HONEYWELL

MULTICS SYSTEM MAINTENANCE PROCEDURES MANUAL

SOFTWARE

MULTICS SYSTEM MAINTENANCE PROCEDURES

SUBJECT

Maintenance Procedures for the Multics System, Including Those Details Required for System Configuration, Startup, Shutdown, Normal Operation (Including BCE), Backup, Dynamic Reconfiguration, Storage System Maintenance, Metering, Bulk I/O Operations, and Crash Recovery

SPECIAL INSTRUCTIONS

This is the fourth revision to AM81, replacing Revision 3, dated May 1985.

Throughout the manual, change bars in the margins indicate technical additions and changes; asterisks denote deletions.

Refer to the Preface for "Significant Changes."

SOFTWARE SUPPORTED

Multics Software Release 12.0

ORDER NUMBER

AM81-04

November 1986

Honeywell

PREFACE

This manual is for Multics system maintainers, system programmers, and machine room supervisors. Its purpose is to provide them with the reference material they need to ensure successful system operation. The manual emphasizes detailed descriptions of system maintenance concepts and functions.

For the most part, Multics operators are expected to use the *Operator's Guide to Multics*, Order No. GB61, which emphasizes step-by-step instructions for performing the most common operations tasks. Only very experienced operators, who have developed an interest in learning more about operating Multics or need in-depth information about a given topic, should refer to the *Multics System Maintenance Procedures* manual.

For detailed descriptions of maintenance commands, maintenance personnel should refer to the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64. Other manuals that may be of interest to them are those that document the major hardware modules and peripheral devices commonly used in a Multics configuration. These manuals are listed in Section 1. For details of Multics software concepts and organization, and for specific usage of Multics commands and subroutines, they should refer to the following volumes of the Multics manual set:

Order Number	Title
AG91	Multics Programmer's Reference Manual
AG92	Multics Commands and Active Functions
AG93	Multics Subroutines and Input/Output Modules

Significant Changes in AM81-04

With Multics release 12.0, BOS becomes totally obsolete. All of the functions which it performed have now been replaced by equivalent functions performed by BCE. Therefore, support for BOS has been completely removed from this manual.

The disk volume recovery procedures in Section 10 and Appendix H have been changed. Instead of using BOS SAVE, RESTOR, SAVE COPY, and TEST, they now use BCE save, restore, copy_disk, and test_disk.

File No.: 1L43

BCE save and restore differ from BOS SAVE and RESTOR in a number of important ways: they allow you to process multiple sets of save or restore information, they include tape error recovery procedures, and they offer improved abort and restart capabilities. A detailed discussion of BCE save and restore has been added to Section 12.

Support for a new hardware I/O system has been added throughout the manual. The main components of this system are:

- the information multiplexer unit (IMU), which performs the same functions as the IOM, but differs from it in that it is controlled by a microprocessor rather than hardwired;
- the integrated peripheral controllers (IPCs), which are the channels in the IMU, and which include IPC-FIPSs, channels which are controlled by a microprocessor and allow use of MSU3380/3390 disk devices and MTU8205/8206/8208 tape devices (IBM compatible devices);
- the maintenance channel adapter (MCA), a microprocessor which controls all IMU initialization and maintenance functions.

A discussion of IMU configuration has been added to Section 3.

The procedure for communicating with the MCA has been added to Section 4.

The procedure for cold booting BCE via the IMU has been incorporated into Section 6.

Information about adding an IMU to the system has been added to Section 11.

The "model" field on the iom config card has been changed to accept one of two values: "iom" or "imu". The value of "nsa" is no longer accepted. Appropriate changes have been made to the manual. Also, the table of time zones described under the clok card has been completely revised. Finally, support has been added for a new config card, the ipc card.

Support for MSU3380/3390 disk devices, which are divided into subvolumes, and MTU8205/8206/8208 tape devices has been added throughout the manual. Note that the existence of subvolumes in 3380/3390 disk devices has necessitated changes to the part and root config cards.

Support has also been added for MTP8021/8022/8023 tape MPCs (which are 611 tape MPCs in PPU cabinets) and MSP8021/8022/8023 disk MPCs (which are 800 disk MPCs in PPU cabinets).

The procedure for moving 451 disk packs in Sections 10 and 12 has been updated.

CONTENTS

Tables

Illustrations

SECTION 1

INTRODUCTION

HOW TO USE THIS MANUAL

This manual is divided into a large number of sections and appendixes.

Section 1 introduces the manual, provides lists of common acronyms and useful hardware documentation, and includes a glossary.

Section 2 gives an overview of system hardware and software.

Section 3 describes the configuration process, including details of setting switches on all major hardware modules.

Section 4 explains how to communicate with the system, and includes discussions of the bootload console and the initializer terminal.

Section 5, which described the bootload operating system (BOS), is obsolete and has been deleted.

Section 6 describes the bootload command environment (BCE).

Section 7 describes the configuration deck and all of the configuration cards.

Section 8 explains how to start the system up and shut it down.

Section 9 describes the hierarchy and volume backup systems.

Section 10 offers advice on recovering from system failures, including information on volume and directory salvaging, and disk unit failures.

Section 11 discusses dynamic reconfiguration.

Section 12 describes how to do storage system maintenance.

Section 13 reviews system messages.

Section 14 explains how to meter and tune the system, and includes suggested guidelines and a glossary of metering terms.

Section 15 discusses the bulk input/output facility, emphasizing procedures for operating the I/O daemon.

Appendix A provides a summary of configuration cards.

Appendix B explains DPU and DMP/VIP operating procedures.

Appendix C provides checklists of switch settings.

Appendix D explains the names of communications channels.

Appendix E discusses system exec_coms.

Appendix F offers a sample system_start_up.ec.

Appendix G discusses volume management.

Appendix H describes alternate procedures for disk volume recovery, thus serving as a supplement to Section 10.

Appendix I describes the Honeywell Error Analysis and Logging System (HEALS) and its use on Multics.

Appendix J discusses Multics disk management, thus serving as a supplement to Section 14.

COMMON ACRONYMS

The following acronyms are used frequently in this manual:

ASCII	American Standard Code for Information Interchange
AST	active segment table
BCE	bootload command environment
* CPU	central processor unit
CU	control unit
DIA	device interface adapter
EBCDIC	Extended Binary-Coded Decimal Interchange Code

EIMA	execute interrupt mask assignment
EIS	extended instruction set
FNP	DATANET 6670 front-end network processor
IMU	information multiplexer unit
IOM	input/output multiplexer
IPC	integrated peripheral controller
MCA	maintenance channel adapter
MPC	microprogrammed peripheral controller
RCP	resource control package
RLV	root logical volume
RPV	root physical volume
SC	system controller
SCU	system control unit
SST	system segment table

HARDWARE MANUALS

The following manuals document the major hardware modules and peripheral devices commonly used in the Multics system configuration.

Order No.	*Title*
AL39	DPS/Level 68 & DPS 8M Multics Processor Manual
AM46	L64/66/68 MTU0400/0500/0600 Operation
AM48	Series 60 MSU0400 Mass Storage Unit Operation
AN37	Series 60 CRU0600/1050 Card Reader Operation
AP88	Series 60 PRU1200/1600 Printer Operation
AT50	Series 60 CCU0400 Card Reader/Punch PCU0120 Card Punch Operation
AT71	L66/68 MSU0402/0451 Mass Storage Unit Operation
AU76	L66/68 PRU1100 Printer Operation
AY03	L66/68 MSU0500/0501 Mass Storage Unit Operation
AY34	L66/68 DATANET 6670 Operation Reference Manual
AY83	L64/66/68 CRU0301/0501 Card Reader Operation
CB64	L66/68 MTU0610 Magnetic Tape Unit Operation
DA33	Series 6000 Equipment Operators Manual
DB28	Series 6000 MTS500 Magnetic Tape Subsystem
DC79	L66/68 Site Preparation Manual
58010010	Information Multiplexer Unit Hardware Operations Manual

GLOSSARY OF TERMS

Included in this glossary are a number of terms commonly used in conjunction with Multics system operation. This is not intended to be an exhaustive or all-inclusive list, but rather a supplemental one oriented toward operations usage. A more extensive glossary of Multics terms is contained in the *Multics Programmer's Reference Manual*, Order No. AG91.

answering service
: the subsystem that controls interactive and absentee users. It runs in the initializer process and handles operations such as dialups, logins, and logouts.

backup
: refers to those systems which ensure that user and system segments and directories can be recovered if they are destroyed due to system failure or user error. See dumping and recovery below.

BCE
: the bootload command environment; a set of programs within Multics initialization that perform functions such as bootloading Multics, dumping main memory, saving and restoring the contents of disk volumes, and initiating emergency shutdown of Multics.

bootload or boot
: to load a fresh copy of a set of programs. Both BCE and Multics can be bootloaded.

Bootloads of BCE and Multics must be discussed jointly. A "cold" boot of BCE and Multics recreates the entire storage system hierarchy on a particular RPV, discarding previous hierarchies, including all user files. A "warm" boot of BCE and Multics maintains the current storage system hierarchy. To do a "cold" boot of BCE and Multics, you first tell BCE to format the RPV with the cold command, then tell Multics to format the file system with the boot -cold command. Obviously, the usual procedure is to do a "warm" boot of BCE and Multics.

The period of time between Multics bootload and shutdown is also spoken of as a bootload or service session.

crash
: a Multics system malfunction that has caused the system to become unavailable to users. Causes of a Multics system crash may stem from either hardware or software troubles. In certain instances, the operator may provoke a system crash (e.g., when the system is in a loop).

daemon
: a system service process that performs such tasks as backup, process creation, network control, and I/O device control.

I/O daemon
: the system service process that controls unit record I/O device operations such as printing and card punching.

dumping
: is the procedure by which the Multics backup systems search out, select, and copy segments and directories from the Multics storage system hierarchy onto tape. The segments and directories selected for dumping are determined by the mode (incremental, consolidated, or complete) in which dumping is performed.

hardcore supervisor
: the part of the system software which performs the supervisory functions of the system. It is also called the hardcore. It can't be changed while the system is running, and includes programs which must be present to bring the system up and programs which run the storage system.

initializer process
: is the control process for the system. When the Multics bootload sequence is started by BCE, the initializer process is created and remains active as long as Multics is running. The initializer process performs various system functions, including answering service operations, operator command service, user request handling, system terminal management, and message routing.

message coordinator
: is a set of programs that manage system terminals and handle message routing. The message coordinator programs run in the initializer process. They allow the initializer to run more than one terminal channel and let the daemons run without attached terminals, sending their messages to the initializer for disposition.

recovery
: the procedure by which the backup systems recover segments and directories that have been dumped onto tape and place them back into the storage system hierarchy. See reloading and retrieving below.

reloading
: is the global recovery of a major portion of the hierarchy when it has been damaged.

retrieving
: is the recovery of individual segments and directories at the request of users.

salvager
: directory - a subsystem that verifies and repairs the directory hierarchy. It is invoked automatically when damage is detected and can also be manually run over all or part of the hierarchy.

  volume - is an *offline* program that operates while the volume is being mounted and not available for use. It examines the volume for damage caused by system malfunctions and corrects the damage if possible. The salvager is invoked automatically in some cases, but it can also be invoked by the operator on instructions from the programming support staff.

scavenger
: an *online* program that examines storage system volumes for damage caused by system malfunctions and corrects the damage if possible. The scavenger operates while the volumes are in service (see also salvager above).

spooling
: is a method of queuing users' print requests when the line printer is either out of service or processing other requests.

syserr log
: is a log of messages, called syserr messages, produced by the Multics supervisor. The syserr messages are written on a reserved area of disk called the LOG partition. Some are also printed on the bootload console. Periodically, these messages are copied into a family of log segments named syserr_log in >sc1>syserr_log.

system_start_up.ec
: an exec_com segment (i.e., a segment containing a list of commands to be executed). It is invoked automatically when the answering service is initialized (by either the startup, multics, or go command). The commands executed may include, but are not limited to, such operations as turning on the message coordinator before starting the answering service, logging in the daemons after the answering service is started, and accepting additional channels if the initializer is to operate more than one terminal.

system control
: another name for the initializer process is the system control process.

user control
: another name for the answering service is user control.

VTOC
: volume table of contents; provides information about all of the segments and directories which reside on a pack, including their location on the pack.

SECTION 2

SYSTEM DESCRIPTION

HARDWARE

A Multics configuration consists of one or more central processor units (CPUs), one or more input/output multiplexers (IOMs) for peripheral interfacing, and one or more front-end network processors (FNPs) for data communications interfacing. Memory in the Multics configuration is provided by one or more system controller units (SCUs) interfacing to memory store units. There can be up to eight SCUs in a Level 68 System, and up to four SCUs in a DPS 8 System. The function and makeup of these units is discussed in the following paragraphs and in the technical manuals referenced in these paragraphs. For an illustration of a Multics system configuration, see Figure 2-1.

MULTICS PROCESSOR

The processor for the Multics system performs all the computational processing within the Multics system configuration. The processor is frequently referred to as the central processing unit (CPU).

The CPU incorporates a manual MODE switch to allow it to selectively function as either a processor with extended capability (MULTICS mode) or a Series 6000 Processor (GCOS mode). A detailed description of the CPU internal organization and the more than 300 machine instructions is given in the *Multics Processor Reference Manual* (Order No. AL39).

MEMORY

The memory system is composed of one or more SCUs that interface directly with CPUs, IOMs, and the memory store units that contain the manipulated data.

A typical Multics system has more than one SCU. Associated with each SCU is an amount of memory, configured into store units.

System Clock

The SCU also contains a calendar clock. It is a 52-bit register that counts one microsecond intervals. You must set the clock to the number of microseconds since midnight January 1, 1901 Greenwich mean time (GMT). Details of how to do this are given in Section 3 of this manual.

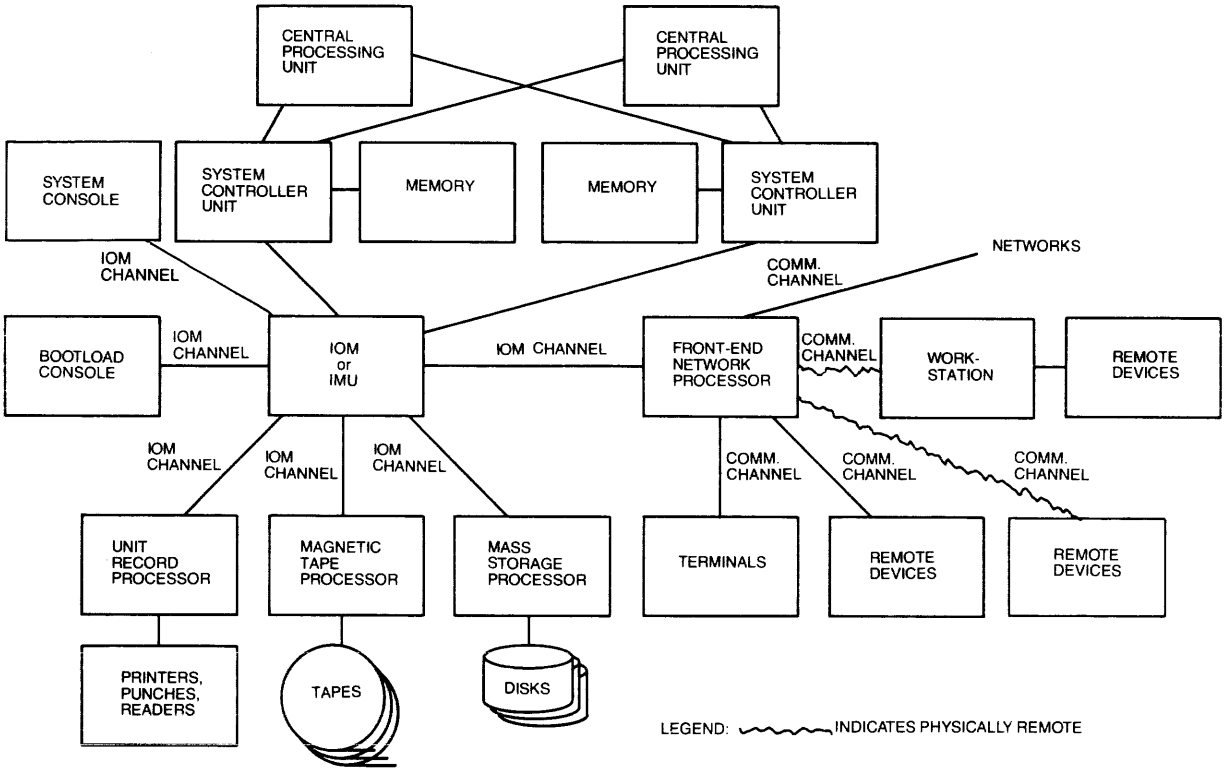


Figure 2-1. A Multics System Configuration

INPUT/OUTPUT MULTIPLEXER

The IOM functions as the I/O processor for the Multics configuration. It handles the transfer of data between the main memory, the FNP, and all peripheral devices including disks, tapes, and unit-record equipment.

Information Multiplexer Unit

The IMU functions much like the IOM for the Multics configuration; i.e., as an I/O processor. It differs from the IOM in that it is controlled by an internal microprocessor rather than being hardwired.

Note: throughout this manual, the term "IOM" refers to both the IOM and the IMU, unless otherwise stated.

FRONT-END NETWORK PROCESSOR

The Multics system requires an FNP. Originally, the DATANET 355 FNP was a partition of the Multics hardware configuration. The DATANET 6670 FNP now replaces the DATANET 355 FNP in Multics configurations. The FNP is a stored program communications processor that receives and processes information from remote terminals for direct input to an IOM data channel. The FNP transmits information to the remote terminals that are connected to the Multics system configuration over private lines or common carrier communication facilities. The DN6670 is described in the *L66/68 DATANET 6670 Operation Reference Manual*, Order No. AY34.

PERIPHERAL SUBSYSTEMS

Peripheral subsystems communicate with the CPU through the IOM using a standard interface. A list of supporting documentation can be found in Section 1.

SOFTWARE

For complete details of Multics software concepts and organization, refer to the following manuals:

Multics Programmer's Reference Manual, Order No. AG91
Multics Commands and Active Functions, Order No. AG92
Multics Subroutines and I/O Modules, Order No. AG93

The Multics software of interest to system maintainers includes:

1. Initializer commands -- for controlling operation of the system such as setting the maximum number of users who can log in, setting the message of the day, and shutting the system down.

2. Answering service -- for accounting purposes and logging users in and out.

3. Backup System -- for copying segments and directories from the storage system onto tape (incremental backup, consolidated backup, complete backup) and recovering segments and directories from tape and placing them back into the storage system (upon user request or after a system failure).

4. I/O daemon -- for processing bulk I/O and controlling remote or local job entry (reading and punching cards, printing output on the high-speed printer).

5. User commands -- such as compilers and text editors.

Storage Hierarchy

You must have an understanding of the layout of the storage system hierarchy in order to be able to take proper action during recovery from a severe system failure. A brief description of the Multics storage system hierarchy and system libraries is given in the following paragraph. Further details are given in the *Multics Programmer's Reference Manual,* Order No. AG91.

The Multics storage system includes both system and user segments. Figure 2-2 shows a portion of the storage system hierarchy nearest to the system root. The hierarchy can be viewed as an upside-down tree with the root at top and directories occurring at each fork in the trunk (i.e., branches of the tree). This diagram shows certain directories always found in the hierarchy. The names of all these directories and the general content of segments inferior to them are listed below.

- >system_control_1 (>sc1)

  is the storage location for most system accounting, authorization, and logging information. (It's the initializer's home directory.) The table printed by the who command, the message of the day, and the absentee queue segments are the only generally accessible segments in this directory. Project administration tables are stored in a directory tree under system_control_1.

- >site

  contains segments and directories which contain per-site information.

- >process_dir_dir (>pdd)

  contains one directory for every process currently in the system. The name of an individual process directory is derived from the unique identification of the process. The process directory is used as a place to store all segments that are intended to have a lifetime no greater than that of the process that creates or uses them.

- >daemon_dir_dir (>ddd)

  contains segments and directories used to support the various I/O system daemon processes.

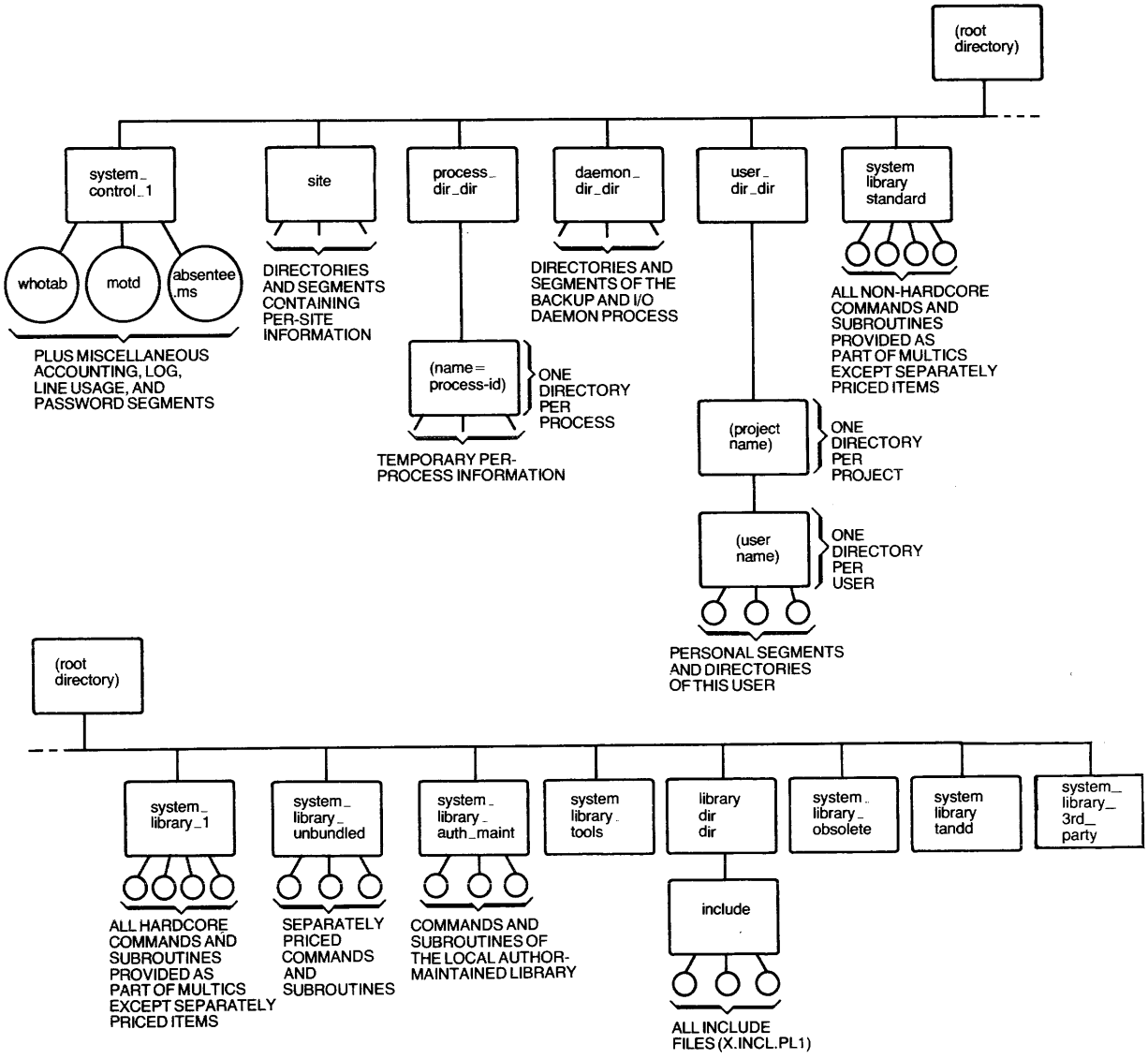


Figure 2-2. Multics Directory Hierarchy

- >user_dir_dir (>udd)

  is the base of a subtree containing all of the personal segments of individual users. The immediate contents of user_dir_dir is a set of directories, one for each project that uses Multics. Contained in a project directory is usually one personal directory for each user working on that project.
- >system_library_standard (>sss)

  contains the library of commands and subroutines which are provided as part of Multics but are not part of the hardcore and are not included on the Multics system tape. These commands and subroutines are documented in the *Multics Commands and Active Functions* manual, Order No. AG92, and the *Multics Subroutines and I/O Modules* manual, Order No. AG93.

- >system_library_1 (>sl1)

  contains the library of commands and subroutines which are part of the hardcore and are included on the Multics system tape. These commands and subroutines are documented in the *Multics Commands and Active Functions* manual, Order No. AG92 and the *Multics Subroutines and I/O Modules* manual, Order No. AG93.

- >system_library_unbundled (>unb)

  contains all the unbundled (separately priced) software supported on Multics.

- >system_library_auth_maint (>am)

  is similar to system_library_standard except that it contains private commands and subroutines provided by programmers of the local installation.

- >system_library_tools (>t)

  contains the commands and subroutines that are used to administer and maintain the system and provide programs that constitute the various system daemon processes for printing, reading and punching cards, etc.

- >library_dir_dir (>ldd)

  contains sources and object archives for system library programs.

- >system_library_obsolete (>obs)

  holds obsolete software programs that are no longer supported by Honeywell. It is the responsibility of the individual site to convert from using obsolete software programs to using new software programs.

- >system_library_tandd (>firmware)

  contains the online T&D firmware program that was previously located in the firmware archive segment >ldd>firmware.

- >system_library_3rd_party (>sl3p)

  contains software written and supported by third party vendors.

For a complete description of these directories and the segments they contain, refer to the *Multics System Administration Procedures* manual, Order No. AK50.

Resource Control Package

The resource control package (RCP) controls and allocates peripheral resources. RCP keeps usage records on the following types of resources:

- Tape drives
- Storage system disk volume attachments
- User I/O disk drives
- Printers, readers, and punches
- Bootload console
- Special devices

RCP allows special privileged attachments for system daemon processes and for CSD test and diagnostic programs.

SECTION 3

CONFIGURATION

DEFINITION

Configuring the Multics system involves setting switches on the Multics processor and its associated devices to connect them into a configuration that can run Multics.

LEVEL 68 SYSTEM VS DPS 8 SYSTEM

The following paragraphs describe the differences between a Distributed Processing System (DPS) 8 and a Level 68 System.

The DPS 8 is a continuation of the Level 68 system. Internally, the DPS 8 processor works differently from the Level 68 processor, but architecturally, they support the same set of instructions and registers. For users, the primary difference between the two processors is that the DPS 8 is faster. For operators and system maintainers, the primary difference is that DPS 8 processors, as well as SCUs and IOMs, do not have maintenance panels. The information that is provided by these panels in a Level 68 system is provided by displays on a terminal in a DPS 8 system. The exact panels which do not exist on DPS 8 boxes are as follows:

```
CPU:  Maintenance, Test and Display Panels
SCU:  Maintenance and Display Panels
IOM:  Maintenance and Test Panels
```

The displays which replace the maintenance panels are produced by the Dynamic Maintenance Panel (DMP), which is part of the processor. Displays from the DMP may be accessed in either of two ways: with a standard VIP terminal attached to the DMP or with a Diagnostics Processor Unit (DPU). The DPU serves as an interface to the DMP for the processor, the SCU and the IOM. Your decision as to whether you should use the DPU or the VIP attached to the DMP will depend on the configuration at your site. You might have one VIP and a patching mechanism to connect it to the desired DMP interface, or a separate terminal for each DMP interface, or some combination of these. You might or might not have a DPU.

The DPU is a Level 6 computer system. A basic DPU subsystem contains a processing unit, a maximum memory size of 128K, a Multiline Communications Processor (MLCP), a Multiple Device Controller, and a DPU Control Panel. A DPU subsystem also has a VIP terminal attached to it (currently a VIP7205). You should not confuse this terminal, which is part of the DPU subsystem, with the terminal mentioned above, which is connected directly to the DMP.

The DPU provides a maintenance capability that includes remote maintenance control of the DPS 8 processor, SCU and IOM. In other words, the switch settings and the contents of various registers for these units may be displayed on the attached terminal.

In addition to the maintenance panels being replaced, the configuration panels on these three units are packaged somewhat differently than they are on the Level 68 machines. For a general view of the DPS 8 processor configuration panel, see Figure 3-6. For general views of the DPS 8 SCU and IOM configuration panels, see Figures 3-3 and 3-9 respectively.

For normal operations, there is little difference in the procedures for setting switches on the configuration panels of any of the DPS 8 units; the switches are still on "standard" configuration panels. However, on the processor, the EXECUTE SWITCHES/EXECUTE FAULT two-position switch is replaced by an EXECUTE FAULT pushbutton. This means that the procedure for executing fault is the same as on the Level 68, but the procedure for executing switches is different. To execute switches to return to BCE, you must use either a DPU or a DMP/VIP. Procedures for operating the DPU and the DMP/VIP are described in Appendix B.

The DPS 8 FNP and memory are the same as the L68 FNP and memory.

You should be aware of the fact that there are some differences between configuring a Level 68 System and configuring a DPS 8 System. These differences are noted where appropriate throughout this section.

DEVICES AND FUNCTIONS

The Multics configuration includes three kinds of devices: active devices, passive devices, and peripherals. The CPUs and IOMs are active devices. The SCUs are passive devices. Printers, tape drives, disk drives, card readers, FNPs, and card punches are examples of peripheral devices.

When active devices reference memory, they generate a 24-bit absolute memory address that is interpreted by the port selection switches (on the processor configuration panel) to produce a request to a specific SCU for a specific word in the memory connected to that controller.

There are three rules that simplify setting up the Multics configuration:

1. Every active device must be able to access all SCUs.

2. Every SCU must have the same active device on the same SCU port, so all SCUs must have the same PORT ENABLE settings.

3. Every active device must have the same SCU on the same port, so all active devices will have the same configuration panel settings.

PERIPHERAL PREPARATION AND OPERATION

All peripherals must be configured in an online and ready state. Refer to the appropriate manual for each piece of peripheral equipment. These manuals are listed in Section 1.

Peripheral devices and terminal devices used by operations personnel vary from site to site, because adding a new device is possible through a general I/O programming technique. It is beyond the scope of this document to describe the operation of each peripheral device or terminal. However, some specific operation of peripheral devices or terminals is discussed when it is significant to the operation of Multics.

SYSTEM CONTROLLER UNIT (6000)

Refer to Figure 3-1 for a general view of the Level 68 6000 SCU configuration panel.

Level 68 6000 SCU Configuration Panel

The Level 68 6000 SCU configuration panel contains the following switches:

```
SYSTEM CONTROL AND MONITOR (CONT&MON/MON/OFF)
SYSTEM BOOT CONTROL (ON/OFF)
ALARM (DISABLE/NORMAL)
MAINTENANCE PANEL MODE (TEST/NORMAL)
STORE A
        MODE (OFFLINE/MAINT/ONLINE)
        SIZE (32K,64K,128K,256K)
STORE B
        MODE (OFFLINE/MAINT/ONLINE)
        SIZE (32K,64K,128K,256K)
EXECUTE INTERRUPT MASK ASSIGNMENT
        (A through D; OFF/0/1/2/3/4/5/6/7/M)
ADDRESS CONTROL
        LOWER STORE (A/B)
        OFFSET (OFF,16K,32K,64K)
        INTERLACE (ON/OFF)
CYCLE PORT PRIORITY (ON/OFF)
PORT CONTROL (ENABLED/PROG CONT/DISABLE)
```

The three positions of the two MODE selector rotary switches, one for STORE A and one for STORE B, function as follows:

ON LINE — Normal operating position; the memory can be accessed by the SCU ports.

MAINT — Maintenance position; the memory cannot be accessed by a 6000 SCU port, but it can be accessed by the 6000 SCU maintenance panel test logic.

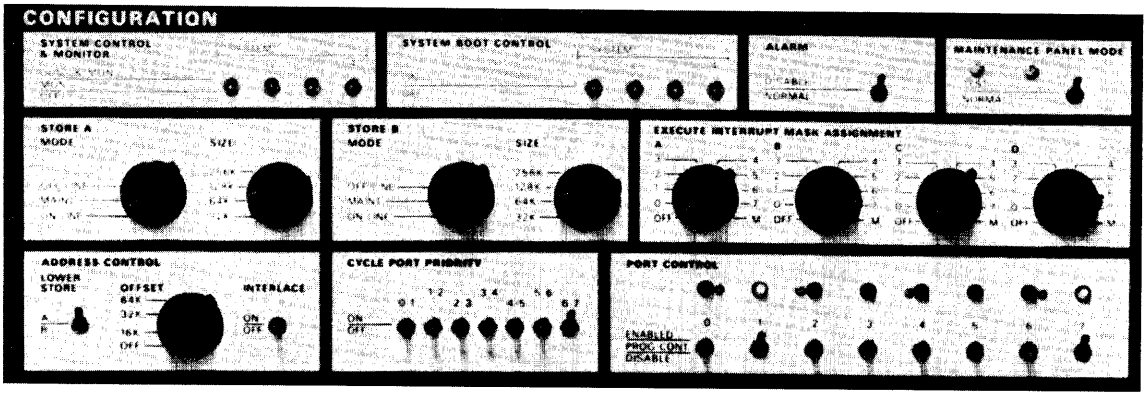


Figure 3-1. Level 68 System Controller Unit (6000 SCU) Configuration Panel

OFF LINE The memory is not accessible by the 6000 SCU. This position effectively removes the memory from the system.

The two SIZE switches (4-position rotary), one for STORE A and one for STORE B, are used in conjunction with the LOWER STORE toggle switch. Reversing the LOWER STORE switch reverses the roles of the SIZE switches. The LOWER STORE toggle switch selects the store to which the lower addresses are routed. The roles of the SIZE switches are as follows:

1. The SIZE switch to which the LOWER STORE switch is set establishes the address boundary between the two memory ports.

2. The combined output of the two SIZE switches establishes the upper address boundary. An attempt to access beyond that boundary produces a nonexistent address illegal action (IA code 0010 (02 octal)).

If STORE A and STORE B are not equal in size, the larger store must be assigned the lower store address range.

The OFFSET rotary switch causes the logical addresses of physical blocks of memory to be exchanged by complementing an address bit. This feature can be used in conjunction with other configuration switches to assign a faulty block of memory to a logically nonexistent upper address range. This 4-position switch causes different address bits to be complemented as follows:

Switch Position	Address Bit Complemented
OFF	No address offset
16K	3
32K	2
64K	1

Address offset only occurs within the memory units since the bit is complemented after the memory has been selected. Therefore, the address offset does not cause an exchange of memory ports; offsets equal to or larger than the memory size are ignored.

The INTERLACE toggle switch enables interleaving between the two stores, A and B. The two stores must have the same size for interleaved operation.

The EXECUTE INTERRUPT MASK ASSIGNMENT (EIMA) rotary switches determine where interrupts sent to memory are directed. The four EIMA rotary switches, one for each program interrupt register, are used to assign mask registers to system ports. The normal settings assign one mask register to each CPU configured. Each switch assigns mask registers as follows:

Position	Function
OFF	Unassigned
0	Assigned to port 0
1	Assigned to port 1
2	Assigned to port 2
3	Assigned to port 3
4	Assigned to port 4
5	Assigned to port 5
6	Assigned to port 6
7	Assigned to port 7
M	Assigned to maintenance panel

Assignment of a mask register to a system port designates the port as a control port, and that port receives interrupt present signals. Up to four system ports can be designated as control ports. The normal settings assign one mask register to each CPU configured.

The eight PORT CONTROL toggle switches are used to enable specific system ports either manually or under program control. The three positions operate as follows:

ENABLE The port is enabled and able to communicate with an active module regardless of the mask bit.

PROG CONT The port is under program control and is turned on or off by the proper bit in the program controlled mask register. This is the normal position for an enabled Multics port.

DISABLED The port is turned off.

There is an indicator light associated with the PORT ENABLE switch. The indicator is on whenever a port is enabled.

The seven CYCLE PORT PRIORITY switches are used to group similar active modules to assure equal access to memory on the same 6000 SCU. Turning these switches on causes system ports to be linked to form groups. System ports within a group have equal access to memory, and no group can link more than five ports. If all ports within a group place continuous access requests, the ports are granted access in cyclic order. Access requests between groups are handled on a priority basis; all access requests by higher priority groups must be satisfied before lower priority access requests are acknowledged. The CYCLE PORT PRIORITY switches link all 6000 SCU ports connected to CPUs, and should link other like devices such as IOMs in their own unique groups. To do this, all switches between CPU ports and between IOM ports are set UP, and others are set DOWN.

The ALARM switch disables the alarm bell when in the DISABLE position.

The MAINTENANCE PANEL MODE (TEST/NORMAL) switch controls the operation of the 6000 SCU maintenance panel. In the NORMAL position, the maintenance panel is disabled. In the TEST position, the 6000 SCU maintenance panel is enabled.

For a summary of the switch settings to be checked before the system is brought up, see Appendix C.

Level 68 6000 SCU Maintenance Panel

Switches on this panel are used when setting the calendar clock. The procedure to set the system clock is described under "Calendar Clock" at the end of this section.

SYSTEM CONTROLLER UNIT (4MW)

In those installations where a 4MW SCU is present, you must be concerned with certain displays and hardware switch settings on the equipment. Refer to Figure 3-2 for a general view of the Level 68 4MW SCU configuration panel, and to Figure 3-3 for a general view of the DPS 8 4MW SCU configuration panel.

System Controller Unit Display Panel (4MW SCU)

The following is a list of switches on the 4MW SCU display panel.

SELECT

The SELECT switch selects what is to be displayed in the 12 display lamps. There are ten possible displays, each of which is explained on the panel.

HISTORY REGISTER

The HISTORY REGISTER switch stores information about the last four 4MW SCU operations. The switches and displays are of little interest to the operator.

On the DPS 8 4MW SCU, the display has been replaced by a display. For details, refer to Appendix F.

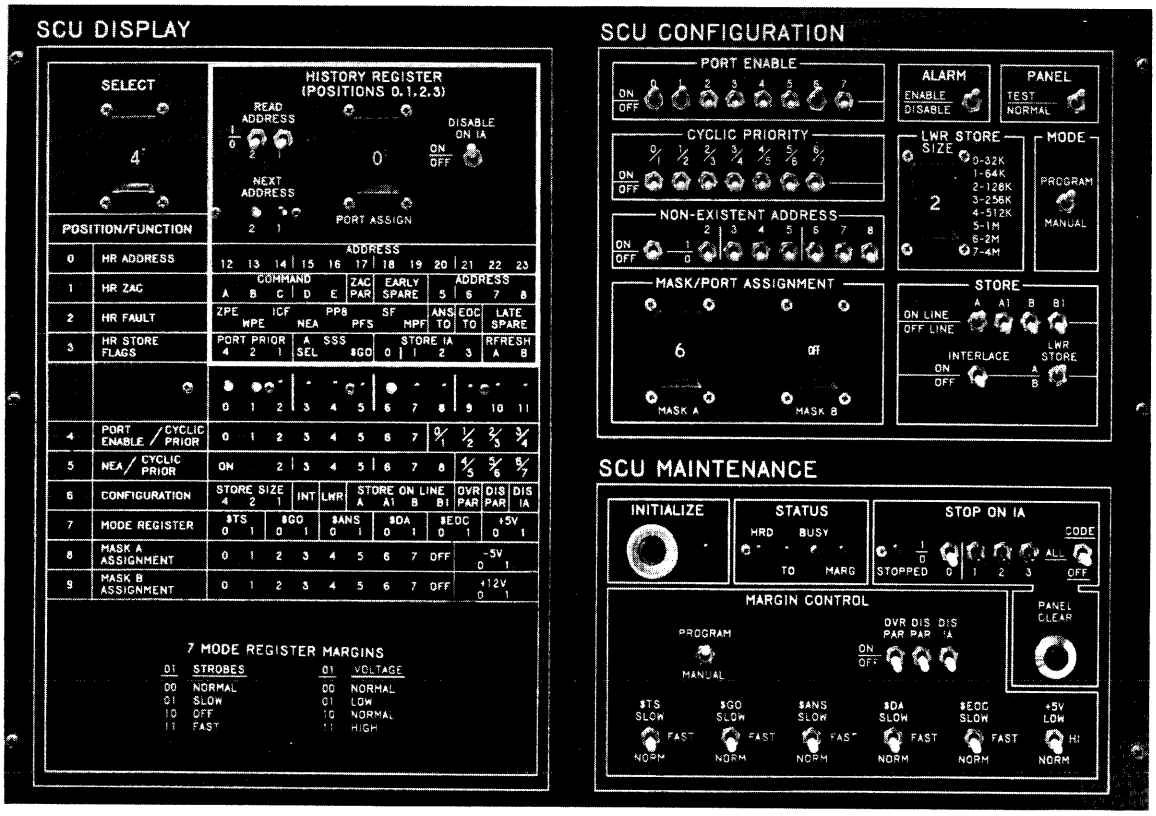


Figure 3-2. Level 68 System Controller Unit (4MW SCU) Panels

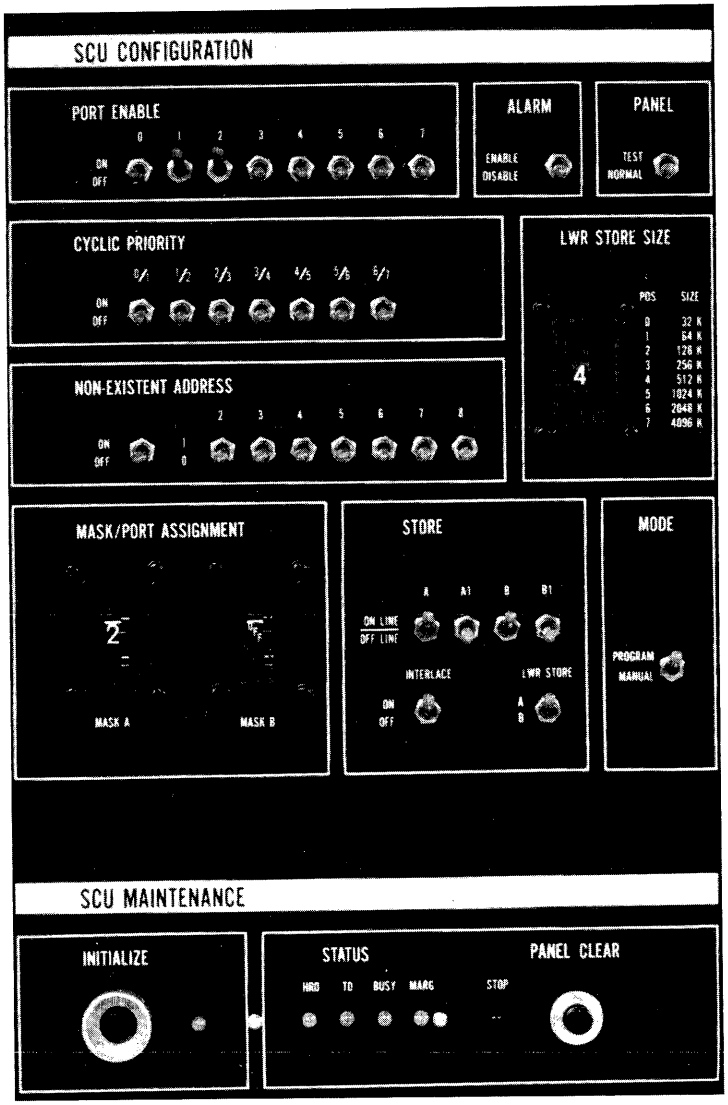


Figure 3-3. DPS 8 System Controller Unit Configuration Panel

System Controller Unit Configuration Panel (4MW SCU)

The switches on the 4MW SCU configuration panel are normally read only when the system is initialized. During Multics operation, internal 4MW SCU registers corresponding to the configuration switches can be changed by software. These registers can always be examined through the use of the SELECT thumbwheel on the 4MW SCU display panel.

The DPS 8 4MW SCU configuration panel is almost identical to the Level 68 panel. For a general view of the DPS 8 panel, refer to Figure 3-3. To set the switches on the DPS 8 panel, follow the instructions given here for the Level 68 panel. Note that with a DPS 8 4MW SCU, the contents of the history registers can be displayed on a terminal screen. For details, refer to Appendix B.

The following switches are found on the 4MW SCU configuration panel:

PORT ENABLE (ON/OFF 0, 1, 2, 3, 4, 5, 6, 7)
These switches are used to enable specific 4MW SCU ports when the system is initialized. They should be ON for each port connected to a configured CPU or IOM, and OFF for all others. The initial settings can be changed under program control. The current settings can be examined by setting the SELECT thumbwheel on the 4MW SCU display panel to position 4.

CYCLIC PRIORITY (ON/OFF 0/1, 1/2, 2/3, 3/4, 4/5, 5/6, 6/7)
These switches have the same function as the CYCLE PORT PRIORITY switches on the 6000 SCU. It is not necessary to set these switches on the 4MW SCU. The proper settings are established under software control during Multics initialization and reconfiguration.

NONEXISTENT ADDRESS (ON/OFF, 2, 3, 4, 5, 6, 7, 8)
These switches are used only when the two store units connected to a 4MW SCU are different sizes. They indicate the first nonexistent addresss of the total memory (stores A and B) present on the controller. The switches have the following values:

Switch Position	Size
2	2048K
3	1024K
4	512K
5	256K
6	128K
7	64K
8	32K

For example, if the controller normally contains 1024K words (configured as A - 256K, A1 - 256K, B - 256K, B1 - 256K) but 128K of Store B1 is defective (or not present), then switches 4, 5 and 6 should be set to indicate that the first nonexistent address is 896K.

ALARM (ENABLE/DISABLE)
This switch is normally left in the ENABLE position.

PANEL (TEST/NORMAL)
This switch is normally left in the TEST position.

LWR STORE SIZE (0/1/2/3/4/5/6/7)
This thumbwheel switch is used to set the size of the lower store unit connected to the 4MW SCU. The meanings of the switch positions are explained on the configuration panel of the 4MW SCU.

MODE (PROGRAM/MANUAL)
This switch must always be in the PROGRAM position.

STORE A, A1, B, B1, (ONLINE/OFFLINE)
These switches enable the store units connected to a 4MW SCU.

INTERLACE (ON/OFF)
This switch enables the interlacing of the two store units connected to a 4MW SCU. It should be ON when upper and lower stores are the same size. Otherwise, it should be OFF.

LWR STORE (A/B)
This switch selects which of the two store units connected to a 4MW SCU contains the lower addressed memory cells. If both store are not the same size, the larger store must be lower.

MASK/PORT ASSIGNMENT A, B (OFF/0/1/2/3/4/5/6/7)
These two thumbwheel switches are analagous to the EXECUTE INTERRUPT MASK ASSIGNMENT switches on a 6000 SCU. When booting, one of them should be set to the port connected to the bootload CPU. The other should be set OFF. These switches can be changed internally under program control. The current settings can be examined by setting the SELECT thumbwheel on the 4MW SCU display panel to positions 8 and 9.

For is a summary of the switch settings to be checked before the system is brought up, see Appendix C.

System Controller Unit Maintenance Panel (4MW SCU)

This panel is of no interest to the operations staff. All switches except the PROGRAM/MANUAL switch should remain in the down position.

On the DPS 8 4MW SCU, the maintenance panel has been replaced by a display. For details, refer to Appendix B.

CENTRAL PROCESSING UNITS

Refer to Figure 3-4 for a general view of the Level 68 processor configuration panel, and to Figure 3-5 for a general view of the DPS 8 processor configuration panel.

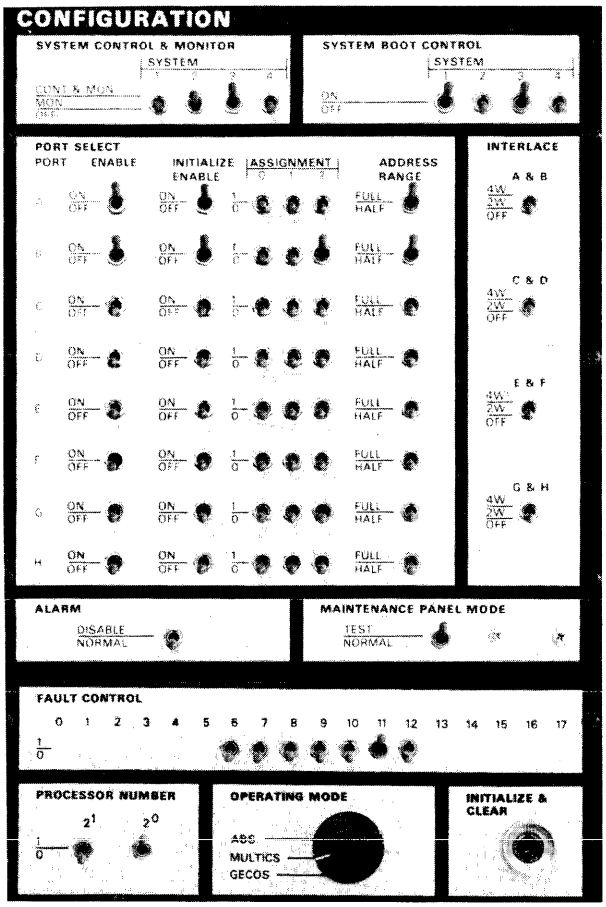


Figure 3-4. Level 68 Processor Configuration Panel

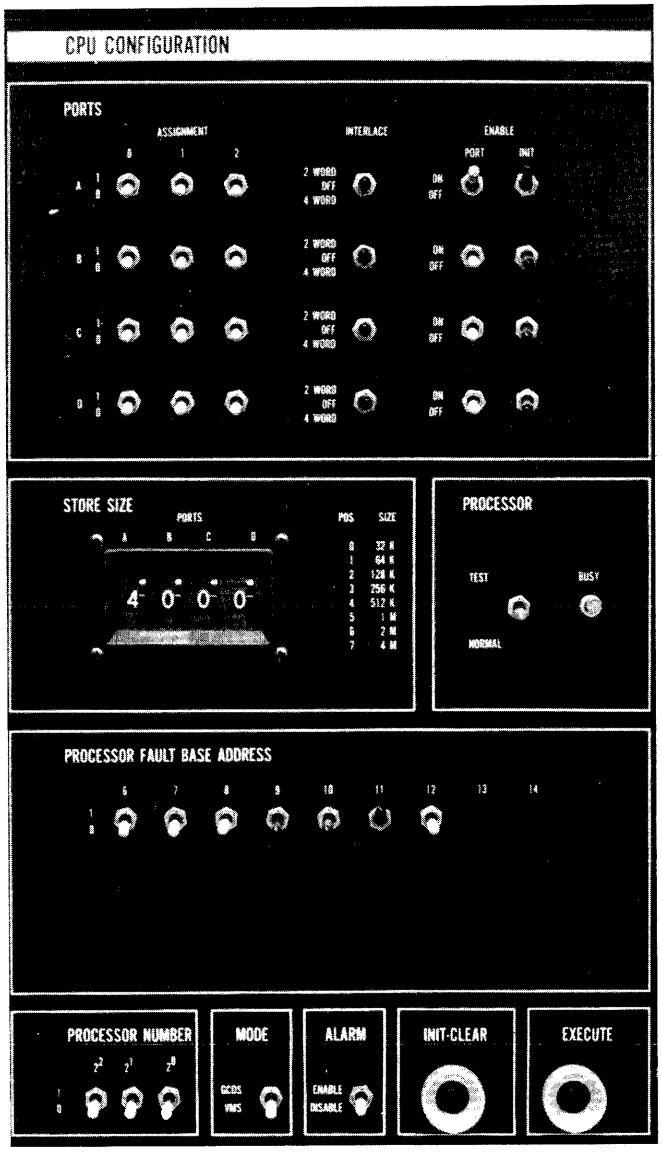


Figure 3-5. DPS 8 Processor Configuration Panel

Configuration Rules

In configuring a Multics system, the following configuration rules apply:

1. Each CPU in the system must be connected to every SCU in the system.

2. Each IOM in the system must be connected to every SCU in the system.

3. Each SCU in the system must be connected to every CPU and IOM in the system.

4. Corresponding ports on all CPUs and IOMs must be connected to the same SCU. For example, port A on every CPU and IOM must be either connected to the same SCU or not connected to any SCU.

5. Corresponding ports on all SCUs must be connected to the same active device (CPU or IOM). For example, if port 0 on any SCU is connected to IOM A, then port 0 on all SCUs must be connected to IOM A.

6. IOMs should be connected to lower-numbered SCU ports than CPUs.

These rules are illustrated in Figure 3-6, where the port numbers for a small Multics system consisting of 2 CPUs, 3 SCUs, and 2 IOMs have been indicated.

A Multics system can support up to 16MW of main memory. This is 16,777,216 36-bit words, and it represents a 24-bit address field. These words are accessed by an absolute address, which can range from 0 to 16,777,215. The main memory on a given system is divided among the configured SCUs so that each SCU contains a contiguous block of storage. The block of storage associated with each SCU is defined by a base address and a size. Both the base address and the size are expressed in units of KW, which is 1024 words. The storage associated with the SCUs on a system can be assigned arbitrarily, but subject to the addressing rules described below.

Level 68 Addressing Rules

1. The base address of an SCU must be a multiple of the value defined by the STORE SIZE patch plug associated with that SCU. Further, this multiple must be in the range 0 through 7. A STORE SIZE patch plug is associated with each pair of CPU or IOM ports (A and B, C and D, etc.). The patch plugs are set to the proper memory size by the field engineers and are not resettable by the site.

2. The size of an SCU must be either the value defined by the STORE SIZE patch plug associated with that SCU or half of that value. Note that the size of an SCU refers to the range of addresses assigned to the SCU. Subject to the limitations to be discussed, the amount of memory actually configured on the SCU may be less than the size.

3. One and only one SCU must have a base address of 0. This SCU is called the bootload SCU.

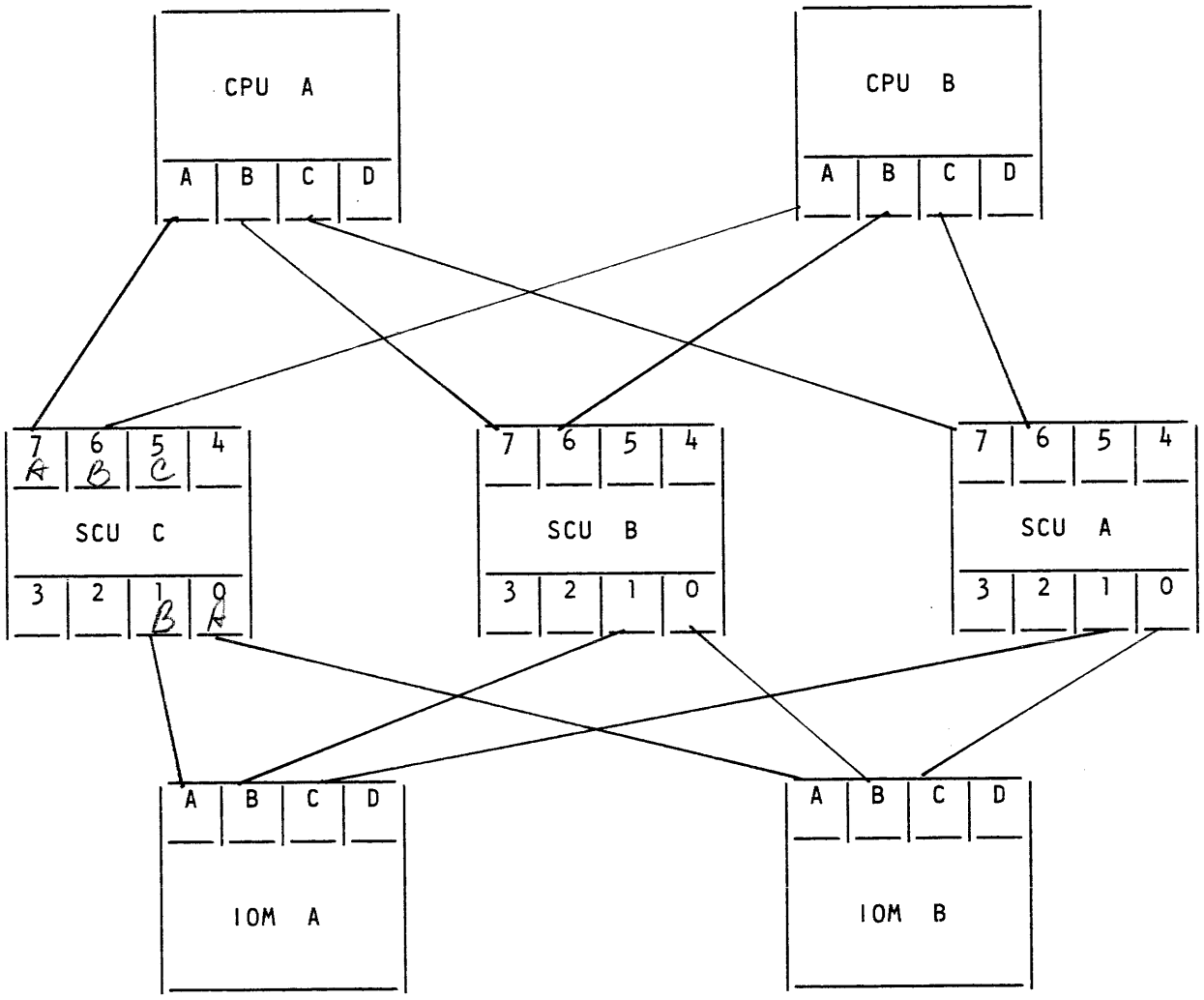


Figure 3-6. Port Numbers for a Small Multics System

4. There can be no overlap in addresses associated with different SCUs. Specifically, the range of addresses defined by the base and size of one SCU must not overlap the range of addresses defined by the base and size of any other SCU, where the size here means the size defined in the STORE SIZE patch plug associated with the port.

5. During bootload, Multics requires a contiguous section of memory beginning at absolute address 0 and sufficiently large to contain all routines and data structures used during the first phase of Multics initialization (i.e., collection 1). The size of the section required varies among Multics releases, and it also depends on the size of the SST segment, which is dependent on the parameters specified by the site on the sst config card (refer to Section 7). However, 512KW is adequate for all circumstances. There can be no "holes" in memory within this region. Beyond this region, "holes" can exist in memory. The examples below will clarify this point.

DPS 8 Addressing Rules

1. The base address of an SCU must be a multiple of the value defined by the STORE SIZE thumbwheel switch associated with that port. Further, this multiple must be in the range 0 through 7. There is a STORE SIZE thumbwheel switch on the central processor and IOM configuration panel for each CPU or IOM port (A, B, C, and D). This thumbwheel switch should be set for each port to be the lowest value that is not smaller than the amount of main memory configured on that port.

2. The size of an SCU is the value defined by the STORE SIZE thumbwheel switch associated with that SCU. Note that the size of an SCU refers to the range of addresses assigned to the SCU. Subject to the limitations to be discussed, the amount of memory actually configured on the SCU may be less than the size.

3. One and only one SCU must have a base address of 0. This SCU is called the bootload SCU.

4. There can be no overlap in addresses associated with different SCUs. Specifically, the range of addresses defined by the base and size of one SCU must not overlap the range of addresses defined by the base and size of any other SCU, where the size here means the size defined in the STORE SIZE thumbwheel switch associated with that SCU on each CPU and IOM.

5. During bootload, Multics requires a contiguous section of memory beginning at absolute address 0 and sufficiently large to contain all routines and data structures used during the first phase of Multics initialization (i.e., collection 1). The size of the section required varies among Multics releases; it also depends on the size of the SST segment, which is dependent on the parameters specified by the site on the sst config card (refer to Section 7). However, 512KW is adequate for all circumstances. There can be no "holes" in memory within this region. Beyond this region, "holes" can exist in memory. The examples below will clarify this point.

Level 68 Processor Configuration Panel (PORT SELECT Panel Area)

The PORT SELECT panel area of the processor configuration panel is identical in both form and function to the PORT SELECT panel area of the Level 68 IOM configuration panel. The purpose of the switches on these panels is to define how the CPUs, IOMs, and SCUs in a Level 68 system are connected, and to define the range of memory addresses associated with each SCU. A Level 68 system may contain a maximum of 7 CPUs, 4 IOMs, 8 SCUs, and 16MW of main memory in aggregate. Each CPU on a Level 68 system has 8 ports for connecting that CPU to SCUs; these ports are identified by the letters a through h. Similarly, each IOM on a Level 68 system has 8 ports for connecting that IOM to SCUs; these ports are identified by the letters a through h. Each SCU has 8 ports for connecting that SCU to CPUs and IOMs; these ports are identified by the numbers 0 through 7.

The following paragraphs describe the functions and settings of the various switches on the PORT SELECT panel area of the Level 68 processor and Level 68 IOM configuration panels. See Figure 3-5. Note that the IMU may not be configured with a Level 68 system.

ASSIGNMENT

These three toggle switches define a 3-bit binary number (ranging from 0 to 7) which determines the base address of the SCU connected to the port. The base address (in KW) is the product of this number and the value defined by the STORE SIZE patch plug for the port. For low-order memory the switches should be set down, to 000; others are set as appropriate.

ADDRESS RANGE

This switch determines the size of the SCU connected to the port. If it is set to FULL, then the size is the value defined by the STORE SIZE patch plug for the port. If it is set to HALF, then the size is half of the value defined by the STORE SIZE patch plug for the port.

PORT ENABLE

These switches indicate which ports are active. The switch for each port connected to an SCU should be ON. The switch for each port that is not connected to an SCU should be OFF.

INITIALIZE ENABLE

These switches enable the receipt of an initialize signal from the SCU connected to the ports. This signal is used during the first part of bootload to set all CPUs to a known (idle) state. The switch for each port connected to an SCU should be ON. The switch for each port that is not connected to an SCU should be OFF.

INTERLACE

This is a 3-position switch that allows interleaving of memory addresses by port pairs in groups of either two or four words. All INTERLACE switches should be set OFF for Multics operation.

The following examples illustrate the determination of the base address and size of an SCU from the switches on the configuration panels:

	STORE SIZE patch plug	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
Example 1:	256KW	3 (0 1 1)	FULL	768KW	256KW
Example 2:	1024KW	0 (0 0 0)	HALF	0KW	512KW
Example 3:	512KW	2 (0 1 0)	FULL	1024KW	512KW

Under the rules defined above, all configuration panels on the system are set identically. That is, all CPU configuration panels are set identically, and all IOM configuration panels are set to match the CPU configuration panels.

The following examples illustrate valid and invalid configurations and configuration panel settings.

Example 4

The following SCUs are configured:

Port	STORE SIZE patch plug	Memory
A	512KW	512KW
B	512KW	256KW
C	1024KW	1024KW
D	1024KW	512KW

The following configurations are all valid for this set of SCUs; the ports are listed in order of increasing base address, which corresponds to the order of mem config cards.

4.1

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
A	0 (0 0 0)	FULL	0KW	512KW
B	1 (0 0 1)	HALF	512KW	256KW
C	1 (0 0 1)	FULL	1024KW	1024KW
D	2 (0 1 0)	HALF	2048KW	512KW

4.2.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
C	0 (0 0 0)	FULL	0KW	1024KW
B	3 (0 1 1)	HALF	1536KW	256KW
D	2 (0 1 0)	HALF	2048KW	512KW
A	5 (1 0 1)	FULL	2560KW	512KW

4.3.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
C	0 (0 0 0)	FULL	0KW	1024KW
A	2 (0 1 0)	FULL	1024KW	512KW
B	3 (0 1 1)	HALF	1536KW	256KW
D	2 (0 1 0)	HALF	2048KW	512KW

4.4.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
D	0 (0 0 0)	HALF	0KW	512KW
C	1 (0 0 1)	FULL	1024KW	1024KW
A	4 (1 0 0)	FULL	2048KW	512KW
B	5 (1 0 1)	HALF	2560KW	256KW

The following are examples of INVALID configurations using the same set of SCUs:

4.5.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
A	0 (0 0 0)	FULL	0KW	512KW
C	1 (0 0 1)	FULL	1024KW	1024KW
B	3 (0 1 1)	HALF	1536KW	256KW
D	2 (0 1 0)	HALF	2048KW	512KW

The above configuration is INVALID because the absolute addresses of SCUs C and B overlap, in violation of addressing rule 4, above.

4.6.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
B	0 (0 0 0)	HALF	0KW	256KW
A	1 (0 0 1)	FULL	512KW	512KW
C	1 (0 0 1)	FULL	1024KW	1024KW
D	2 (0 1 0)	HALF	2048KW	512KW

The above configuration is INVALID--insufficient contiguous memory beginning at absolute address 0 is provided for Multics initialization. Following addressing rule 5, at least 512KW of contiguous memory is required, while only 256KW has been provided (there is a "hole" in main memory in the range 256KW-512KW). Note the difference between this example and valid configuration 2, above. In valid configuration 2, there is a "hole" in memory immediately above the bootload SCU. However, 1024KW of contiguous configured memory beginning at absolute address 0 has been provided, which is sufficient for Multics initialization.

Example 5

The following SCUs are configured:

Port	STORE SIZE patch plug	Memory
A	512KW	512KW
B	512KW	256KW

The following are valid configurations for this set of SCUs:

5.1.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
A	0 (0 0 0)	FULL	0KW	512KW
B	1 (0 0 1)	HALF	512KW	256KW

5.2.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
A	0 (0 0 0)	FULL	0KW	512KW
B	4 (1 0 0)	HALF	2048KW	256KW

Note that there is no valid configuration for this set of SCUs if SCU B is the bootload SCU. The reason for this is that the base address of SCU A must be a multiple of the value determined by the STORE SIZE patch plug, which is 512KW. If SCU B were configured as the bootload SCU, there would be a "hole" in memory between the end of SCU B (256KW) and the beginning of SCU A (whose base address would be at least 512KW). Following addressing rule 5, at least 512KW of contiguous memory beginning at absolute address 0 should be provided for Multics initialization. Thus any configuration for this set of SCUs where SCU B is the bootload SCU would violate addressing rule 5.

One complication can arise because STORE SIZE patch plugs are available only in certain standard sizes (e.g., 128KW, 256KW, 512KW, 1024KW). It is possible for an amount of memory to be configured on an SCU which does not correspond to any standard size patch plug. In this case, a larger size patch plug should be used (e.g., 512KW for an SCU which contains 384KW of memory). For the purpose of setting the switches on the configuration panels, the SCU is treated as if it had the larger amount of memory defined by its associated STORE SIZE patch plug. However, the mem config card must reflect the actual amount of memory configured on the SCU. Further, the NONEXISTENT ADDRESS switches on the SCU maintenance panel should reflect the actual amount of memory configured on the SCU. In applying addressing rule 5, the actual memory configured must be considered rather than the amount of memory indicated by the STORE SIZE patch plug.

The following example should clarify these points.

Example 6

The following SCUs are configured:

Port	STORE SIZE patch plug	Memory
A	512KW	384KW
B	512KW	512KW

The following configuration is valid for this set of SCUs:

6.1.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
B	0 (0 0 0)	FULL	0KW	512KW
A	1 (0 0 1)	FULL	512KW	512KW

Note that the size associated with SCU A for the purposes of setting the switches on the configuration panels is larger than the amount of memory actually configured on SCU A. The NONEXISTENT ADDRESS switches on the maintenance panel of SCU A should be set to reflect the actual amount of memory configured. Similarly, the mem config card for SCU A must reflect the actual amount of memory configured (384KW).

The following configuration is INVALID for this set of SCUs:

6.2.

Port	ASSIGNMENT switches	ADDRESS RANGE switch	Base Address	Size
A	0 (0 0 0)	FULL	0KW	512KW
B	1 (0 0 1)	FULL	512KW	512KW

This configuration is invalid because it violates addressing rule 5, above. According to that rule, there must be at least 512KW of contiguous memory beginning at absolute address 0. In this configuration, there is a "hole" in memory between 192KW and 256KW, even though this "hole" is not reflected in the configuration panel switches.

Note that there was no mention of CPUs or IOMs in the above examples. If the configuration rules defined above are followed, the switch settings on all CPU and IOM configuration panels are identical. Thus for the purpose of setting switches on the configuration panels, the number of CPUs and IOMs configured is not relevant.

For a summary of the switch settings to be checked before the system is brought up, see Appendix C.

.. 12h "Level 68 Processor Maintenance Panel"

Figure 3-7 shows a general view of the Level 68 processor maintenance panel, and part of the display panel. This part of the display panel has been included to show where the EXECUTE SWITCHES/EXECUTE FAULT switch is located.

Switches on the maintenance panel are set by CSD or by the programming staff. They may be set by or under the direction of a responsible person. Switch settings should not be changed while the system is running. If it is necessary to change switch settings without shutting down, enter BCE with the bce initializer command, put the processor in STEP mode, and then change the settings.

The following paragraphs describe the functions and settings of the switches on the Level 68 processor maintenance panel.

The 36 DATA switches are used to enter data or an instruction into the processor.

The ADDRESS toggle switches are used when the maintenance panel command ENTER STORE is executed. The address set into the ADDRESS switches is always present for comparison on a maintenance panel stop-on-address condition or a mode-register-trap-on-address condition. The SEGMENT NUMBER toggle switches are used for maintenance panel stop-on-address condition.

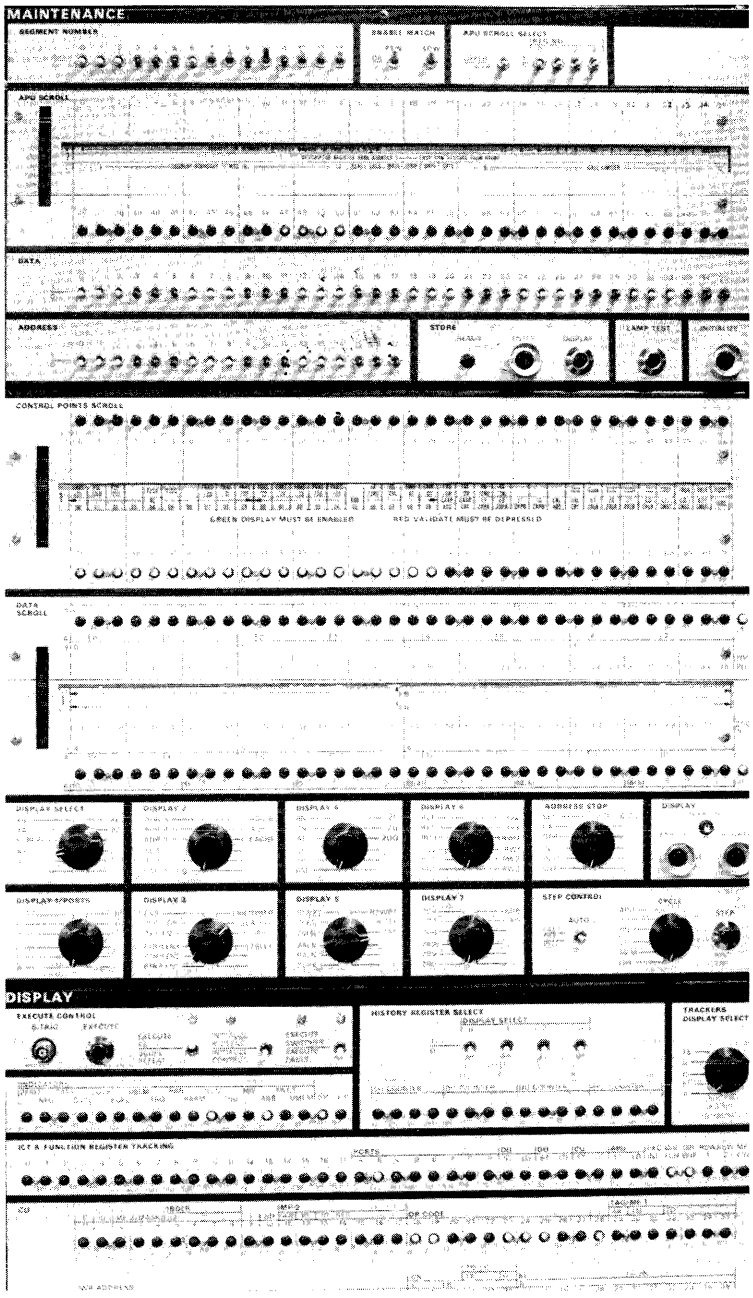


Figure 3-7. Level 68 Processor Maintenance Panel and Part of Display Panel

The DISPLAY ENABLE pushbutton forces an internal condition in the processor which makes registers and control points available for display. The DISPLAY ENABLE pushbutton must be used only when the processor is stopped and the DISPLAY ENABLED light is not lit. The processor should be placed in STEP mode before pressing this button. If DISPLAY ENABLE is pushed while the processor is running, the processor takes indeterminate action that may be fatal to the system.

The DISPLAY ENABLED condition must always be in effect when viewing entries on the DATA SCROLL and those entries on the CONTROL POINTS SCROLL that are printed in green.

The VALIDATE pushbutton must be pressed to view entries on the CONTROL POINTS SCROLL that are printed in red. The VALIDATE pushbutton must be used only when the processor is stopped, else results similar to those resulting from pushing DISPLAY ENABLE while the processor is running may occur.

The CYCLE control knob controls if, and in what way, the processor moves from cycle to cycle. This control knob must be in the OFF position for normal operation.

The ADDRESS STOP control is used for debugging by system programmers. This control knob must be in the OFF position for normal operation.

The HISTORY REGISTER DISPLAY CONTROL switches are used in conjunction with DATA SCROLL position 7 (APU HISTORY REGISTER), position 11 (CU HISTORY REGISTER), position 12 (OU HISTORY REGISTER) and position 13 (DU HISTORY REGISTER) to display the contents of the history registers on the DATA SCROLL. There are 16 registers in the CU, OU, DU, and APU history registers (a total of 64). The individual one of the 16 registers in each of the CU, OU, DU, and APU history registers is selected by setting SELECT COUNT switches 0, 1, 2, and 3 to the number of the desired register.

The CONTROL POINT SCROLL, DATA SCROLL, DISPLAY SELECT, DISPLAY 1, 2, 3, 4, 5, and 6 switches are used to cycle the register, bus, or control point onto the maintenance panel display.

The LAMP TEST pushbutton is used to test the maintenance panel indicator lamps. An unlit lamp indicates either a defective lamp or an open circuit.

The INITIALIZE pushbutton sets all control points to a known state (set to INITIALIZE CONTROL). It is used to initialize the control points without affecting the data registers.

DPS 8 Processor Configuration Panel

The DPS 8 processor configuration panel is identical in both form and function to the DPS 8 IOM configuration panel. The purpose of the switches on these panels is to define how the CPUs, IOMs, and SCUs in a DPS 8 system are connected; and to define the range of memory addresses associated with each SCU. A DPS 8 system may contain a maximum of 6 CPUs, 4 IOMs, 4 SCUs, and 16MW of main memory in aggregate. Each CPU on a DPS 8 system has 4 ports for connecting that CPU to SCUs; these ports are identified by the letters a through d. Similarly, each IOM on a DPS 8 System has 4 ports for connecting that IOM to SCUs; these ports are identified by the letters a through d. Each SCU has 8 ports for connecting that SCU to CPUs and IOMs; these ports are identified by the numbers 0 through 7.

The following paragraphs describe the functions of the various switches on the DPS 8 processor and DPS 8 IOM configuration panels. See Figure 3-6. See "Information Multiplexer Unit" later in this section for a description of IMU configuration.

ASSIGNMENT
: These three toggle switches define a 3-bit binary number (ranging from 0 to 7) which determines the base address of the SCU connected to the port. The base address (in KW) is the product of this number and the value defined by the STORE SIZE thumbwheel switch for the port.

STORE SIZE
: This thumbwheel switch determines the size of the SCU connected to the port.

PORT ENABLE
: These switches indicate which ports are active. The switch for each port connected to an SCU should be ON. The switch for each port that is not connected to an SCU should be OFF.

INITIALIZE ENABLE
: These switches enable the receipt of an initialize signal from the SCU connected to the ports. This signal is used during the first part of bootload to set all CPUs to a known (idle) state. The switch for each port connected to an SCU should be ON. The switch for each port that is not connected to an SCU should be OFF.

INTERLACE
: This is a 3-position switch that allows interleaving of memory addresses by port pairs in groups of four words. (The DPS 8 processor is restricted to NO or 4-word interlace; 2-word interlace is not supported.) All INTERLACE switches should be set OFF for Multics operation.

The following examples illustrate the determination of the base address and size of an SCU from the switches on the configuration panels:

	ASSIGNMENT switches	STORE SIZE thumbwheel switch	Base Address	Size
Example 1:	3 (0 1 1)	1024KW	3072KW	1024KW
Example 2:	0 (0 0 0)	2048KW	0KW	2048KW
Example 3:	2 (0 1 0)	512KW	1024KW	512KW

Under the rules defined above, all configuration panels on the system are set identically. That is, all CPU configuration panels are set identically, and all IOM configuration panels are set to match the CPU configuration panels.

The following examples illustrate valid and invalid configurations and configuration panel settings.

Example 4

The following SCUs are configured:

Port	Memory
A	512KW
B	512KW
C	1024KW
D	512KW

The following configurations are all valid for this set of SCUs; the ports are listed in order of increasing base address, which corresponds to the order of mem config cards.

4.1.

Port	ASSIGNMENT switches	STORE SIZE thumbwheel switch	Base Address	Size
A	0 (0 0 0)	512KW	0KW	512KW
B	1 (0 0 1)	512KW	512KW	512KW
C	1 (0 0 1)	1024KW	1024KW	1024KW
D	4 (1 0 0)	512KW	2048KW	512KW

4.2.	Port	ASSIGNMENT switches	STORE SIZE thumbwheel switch	Base Address	Size
	C	0 (0 0 0)	1024KW	OKW	1024KW
	B	3 (0 1 1)	512KW	1536KW	512KW
	D	4 (1 0 0)	512KW	2048KW	512KW
	A	5 (1 0 1)	512KW	2560KW	512KW

4.3.	Port	ASSIGNMENT switches	STORE SIZE thumbwheel switch	Base Address	Size
	C	0 (0 0 0)	1024KW	OKW	1024KW
	A	2 (0 1 0)	512KW	1024KW	512KW
	B	3 (0 1 1)	512KW	1536KW	512KW
	D	4 (1 0 0)	512KW	2048KW	512KW

4.4.	Port	ASSIGNMENT switches	STORE SIZE thumbwheel switch	Base Address	Size
	D	0 (0 0 0)	512KW	OKW	512KW
	C	1 (0 0 1)	1024KW	1024KW	1024KW
	A	4 (1 0 0)	512KW	2048KW	512KW
	B	5 (1 0 1)	512KW	2560KW	512KW

The following is an example of an INVALID configuration using the same set of SCUs.

4.5.	Port	ASSIGNMENT switches	STORE SIZE thumbwheel switch	Base Address	Size
	A	0 (0 0 0)	512KW	OKW	512KW
	C	1 (0 0 1)	1024KW	1024KW	1024KW
	B	3 (0 1 1)	512KW	1536KW	512KW
	D	4 (1 0 0)	512KW	2048KW	512KW

This configuration is INVALID because the absolute addresses of SCUs C and B overlap, in violation of addressing rule 4, above.

Note that there was no mention of CPUs or IOMs in the above examples. If the configuration rules defined above are followed, the switch settings on all CPU and IOM configuration panels are identical. Thus, for the purpose of setting switches on the configuration panels, the number of CPUs and IOMs configured is not relevant.

DPS 8 Processor Maintenance Panel

On the DPS 8 CPU, the maintenance panel (as well as the test and display panels) has been replaced by a display. For details, refer to Appendix B.

INPUT/OUTPUT MULTIPLEXER

Refer to Figure 3-8 for a general view of the Level 68 IOM configuration panel, and to Figure 3-9 for a general view of the DPS 8 IOM configuration panel.

Input/Output Multiplexer Configuration Panel

The IOM configuration panel contains the following switches:

```
SYSTEM CONTROL & MONITOR (CONT & MON/ MON/OFF)
SYSTEM BOOT CONTROL (ON/OFF)
PORT ENABLE for each port (ON/OFF)
INITIALIZE ENABLE for each port (ON/OFF)
ASSIGNMENT (1/0)
ADDRESS RANGE (FULL/HALF)
INTERLACE (4W/2W/OFF)
ALARM (DISABLE/NORMAL)
MAINTENANCE PANEL MODE (TEST/NORMAL)
IOM BASE (12 through 23; 1/0)
INTERRUPT BASE (6 through 18; 1/0)
IOM NUMBER (1/0)
SOURCE (CARD/TAPE)
CHANNEL NUMBER CODE
  TAPE CHANNEL NUMBER (0 through 5; 1/0)
  CARD CHANNEL NUMBER (0 through 5; 1/0)
SYSTEM INITIALIZE
ZERO BASE S.C. PORT NO. (1/0)
OPERATING MODE (PAGED/EXT GCOS/STD GCOS)
BOOTLOAD
```

The IOM and the processor configuration panels are closely related; in fact, one of the sides of each is identical to the other. Switches common to the IOM and processor configuration panels are set identically.

The DPS 8 IOM configuration panel is almost identical to the Level 68 panel. For a general view of the DPS 8 panel, refer to Figure 3-9. To set the switches on the DPS 8 panel, follow the instructions given here for the Level 68 panel. Note that there is a switch on the DPS 8 panel called the PROGRAM MANUAL switch. This switch should be set to MANUAL, to prevent the configuration from being changed via a Diagnostics Processor Unit.

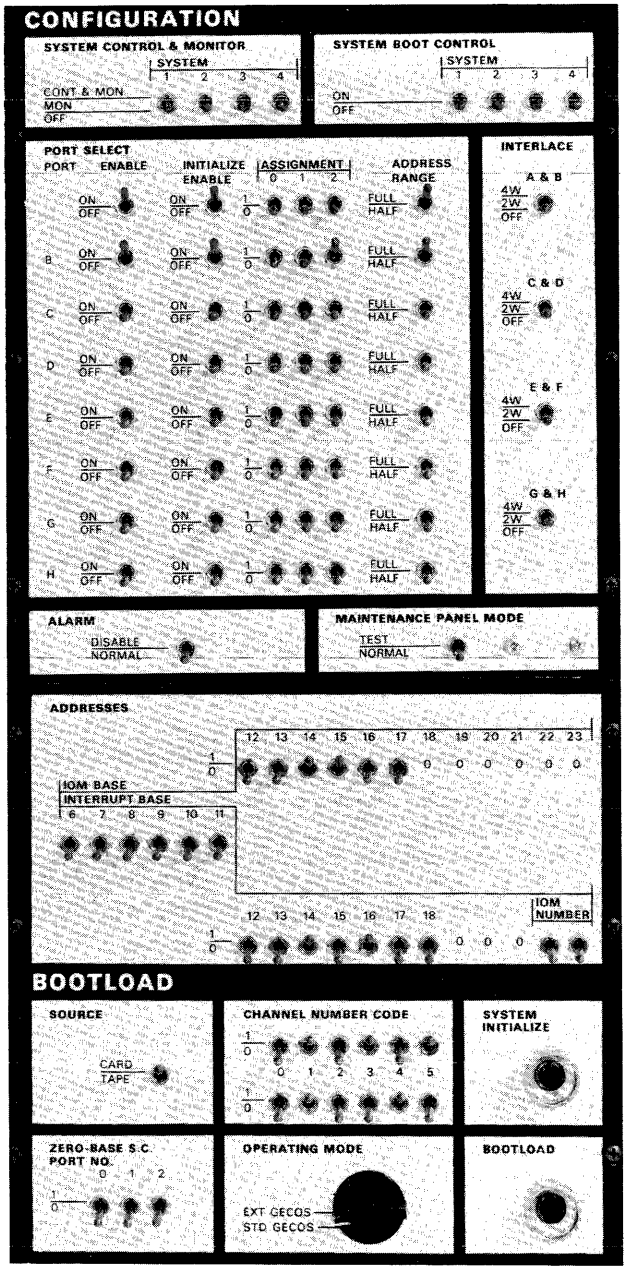


Figure 3-8. Level 68 Input/Output Multiplexer Configuration Panel

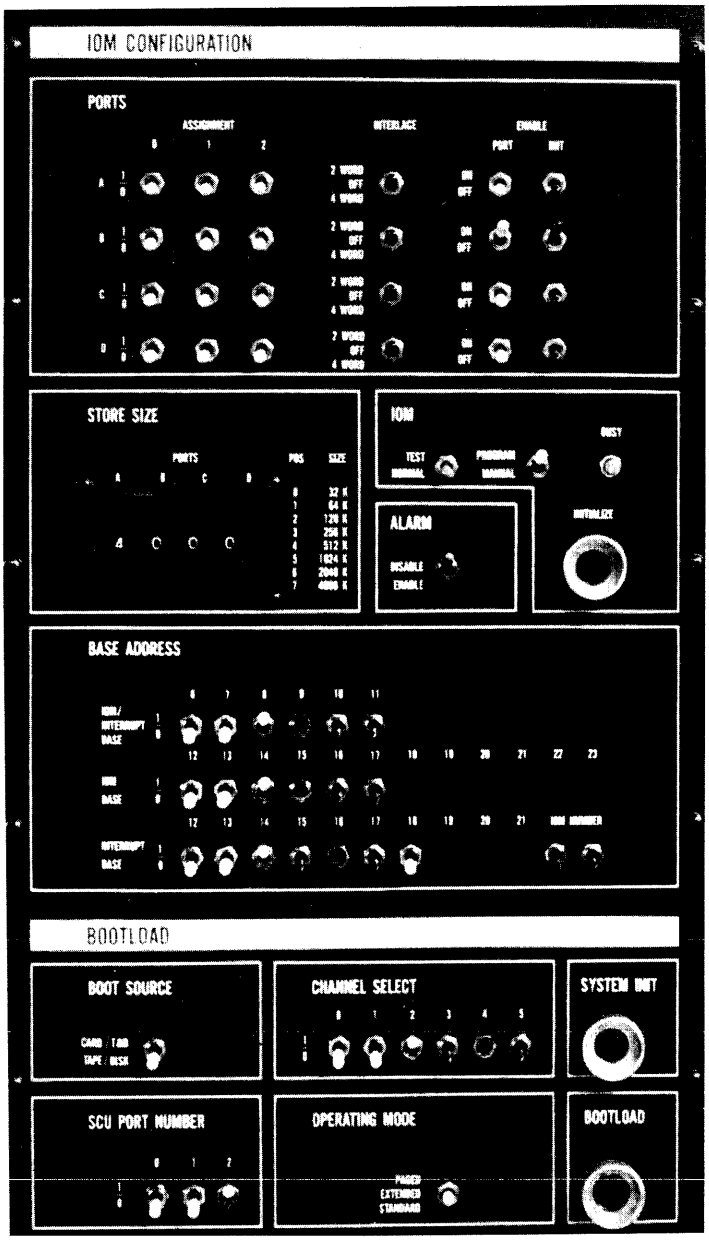


Figure 3-9. DPS 8 Input/Output Multiplexer Configuration Panel

The functions and settings of all switches on the IOM configuration panel are described in the following paragraphs.

For a description of the ASSIGNMENT, INTERLACE, ADDRESS RANGE, and PORT ENABLE switches, as well as the STORE SIZE patch plugs, refer to the previous discussion on the functions of the processor configuration panel switches.

The INITIALIZE ENABLE toggle switch allows the port to be initialized from other ports. This switch is set ON. When OFF, the port initialize signal is inhibited.

The SOURCE toggle switch, in conjunction with the CHANNEL NUMBER CODE toggle switches, selects the source of the bootload (always TAPE) and defines the channel number of the tape subsystem that contains the bootload program.

The ZERO BASE S.C. PORT NO. toggle switches define the port number of the SC through which connects are to be sent to the IOM.

To set up the IOM for bootloading:

1. The SOURCE switch is set in the TAPE position.

2. The tape channel number is set into the TAPE CHANNEL NUMBER switches, labeled CHANNEL NUMBER CODE.

3. The ZERO BASE S.C. PORT NO. switches are set to reflect the SCU port number to which the IOM is connected.

The ALARM toggle switch is used to disable the IOM alarm.

The MAINTENANCE PANEL MODE toggle switch controls the operation of the IOM maintenance panel. For normal operation, this switch is in the NORMAL position. In the TEST position, the maintenance panel options of the maintenance panel are enabled and the TEST portion of the TEST/NORMAL indicator on the operators panel is ON.

The 12 IOM BASE ADDRESS toggle switches are used to set the base address for the IOM. The IOM base address (channel mailbox base address) indicates to software where control words are located in memory. The IOM base address is an 18-bit address, but only bits 0 through 11 are set by the IOM BASE ADDRESS switches. Bits 12 through 17 are all zeros. IOM base addresses for Multics operation are as follows:

IOM A - 1400(8)
IOM B - 2000(8)
IOM C - 2400(8)
IOM D - 3000(8)

The 13 INTERRUPT BASE ADDRESS toggle switches allow the operator to set up a base address for interrupt multiplex words (IMWs). Bits 6 through 11 are shared with the IOM base address, and bits 16 and 17 identify the IOM number. The interrupt base address for Multics operation is 1200(8).

The SYSTEM INITIALIZE pushbutton is used to send a system initialize signal to all enabled ports.

The BOOTLOAD pushbutton is used to initiate the BOOTLOAD sequence.

The OPERATING MODE rotary switch must be set to the PAGED position.

IOM NUMBER must be set as follows:

```
IOM A  -  00
IOM B  -  01
IOM C  -  10
IOM D  -  11
```

All other IOM switches should be in the down (NORMAL) position.

For a summary of the switch settings to be checked before the system is brought up, see Appendix C.

Input/Output Multiplexer Maintenance Panel

Switches on the IOM maintenance panel should only be set by CSD or by the programming staff.

On the DPS 8 IOM, the maintenance panel (as well as the test panel) has been replaced by a display. For details, refer to Appendix B.

Input/Output Multiplexer Operation

In order to use the IOM for Multics, the IOM port must be enabled on the SCUs.

INFORMATION MULTIPLEXER UNIT (IMU)

Unlike other mainframes in the system configuration, the IMU has no configuration panel or switches. Instead, configuration functions are performed by the maintenance channel adapter (MCA), a microprocessor inside the IMU. The MCA uses configuration files stored on diskettes. An IMU can have up to four configuration files. To configure the IMU, use the MCA config command. This command is menu driven. Both the command and the menus are described in the *Information Multiplexer Unit Hardware Operations Manual*, Order No. 58010010. A few of the menu functions are also described here.

To enter IMU bootstrap information, select item 4 of the config command menu. The MCA will prompt you with a configuration topic, and with both its current value and a list of acceptable input values. To keep the current value, respond with a CR.

- "IMU number" is the number which corresponds to the name of the IMU being configured (e.g., "0" = A, "1" = B, etc.).
- "host oper system" is the type of operating system the IMU is running with (e.g., "2" = Multics).
- "Level1-remote maintenance allowed" enables or disables the Remote Maintenance Interface (RMI). This prompt should always be answered by typing "N".
- "The lowest MCA number" is the lowest MCA number connected to the Multidrop Interface (MDI); see Section 4.
- "Total number of MCA" is the total number of MCAs on the MDI.
- "bootstrap enable" indicates whether this IMU can boot the system.
- "bootstrap automatic" enables or disables automatic booting. This prompt should be answered by typing "N".
- "bootstrap scu port number" is the port number of the SCU through which connects are sent to the IMU (i.e., the port on the SCU to which this IMU is connected).
- "bootstrap source" is the device type to boot from. This prompt should be answered by typing "2" (for tape). At present Multics can only boot from tape.
- "bootstrap primary channel number" is the channel number of the bootload tape subsystem.
- "interrupt base address" is the address for the Interrupt Multiplexer Words, and is calculated by the MCA using the "host oper system" and the "IMU number." This prompt should be answered by typing a CR.
- "mailbox base address" is the address where the software places the control words in memory, and is calculated by the MCA using the "host oper system" and the "IMU number." This prompt should be answered by typing a CR.

To enter the memory port configuration, select item 5 of the config command menu. The MCA will prompt you with a configuration topic, and with both its current value and a list of acceptable input values. To keep the current value, respond with a CR.

- "enter memory port number A-D" is the memory port number.
- "memory port enable" indicates whether the IMU uses this port.
- "memory port initialize" enables or disables the acceptance of the system initialize signal from this port.
- "memory port starting address" sets the starting address for this memory. Your response may be selected from the following table.

MEM SIZE	256K	512K	1M	2M	4M
PSA	0	0	0	0	0
OTD	256K	512K	1M	2M	4M
SAD	512K	1M	2M	4M	8M
SRR	768K	1536K	3M	6M	12M
ITE	1M	2M	4M	8M	
BIS	1280K	2560K	5M	10M	
LNS	1536K	3M	6M	12M	
EG	1792K	3584K	7M	14M	

- "memory port size" is the size of the memory on this port. The valid responses are: 256K, 512K, 1M, 2M, 4M.
- "memory port interlace" enables or disables the interlacing of two memory ports. It is recommended that SCU ports not be interlaced on a Multics system; therefore, this prompt should be answered by typing "N".

FRONT-END NETWORK PROCESSOR

The front-end network processor (FNP) provides the logical and physical connection between the system and a remote I/O device. There may be up to eight FNPs on a Multics system.

Front-End Network Processor Operation

The DN6670 configuration panel consists of only a direct interface adapter (DIA) panel. Set switches on the DIA panel as follows:

6000 MAILBOX		octal
FNP A	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↓ ↓ ↓ ↑ ↑	003400
FNP B	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↓ ↓ ↓ ↓ ↓	003700
FNP C	↑ ↑ ↑ ↑ ↑ ↑ ↓ ↑ ↑ ↑ ↓ ↑	004200
FNP D	↑ ↑ ↑ ↑ ↑ ↑ ↓ ↑ ↑ ↓ ↑ ↓	004500
FNP E	↑ ↑ ↑ ↑ ↑ ↑ ↓ ↑ ↓ ↑ ↑ ↑	005000
FNP F	↑ ↑ ↑ ↑ ↑ ↑ ↓ ↑ ↓ ↑ ↓ ↓	005300
FNP G	↑ ↑ ↑ ↑ ↑ ↑ ↓ ↑ ↓ ↓ ↓ ↑	005600
FNP H	↑ ↑ ↑ ↑ ↑ ↑ ↓ ↓ ↑ ↑ ↑ ↓	006100
6000 TERMINATE	↑ ↓ ↓ ↑	3
6000 EMERGENCY	↓ ↓ ↓ ↑	7
HNP MAILBOX	↑ ↑ ↓ ↑ ↓ ↓	454
HNP TERMINATE	↑ ↑ ↓ ↑	2
HNP SPECIAL INT	↑ ↑ ↓ ↓	3

Each FNP may be configured with one or two DIA boards. Each board must be configured on a separate FNP port. The FNP ports on which DIA boards may be configured are 3, 4, 5, and 14. Each board's FNP port must be cabled to an IOM channel. The FNP DIA connections are identified in the Multics config deck by the IOM channel to which the FNP port is cabled. The Multics software determines which FNP port to use in accessing the FNP by references to the IOM channel cabled to the active FNP port. The FNP port number is not recorded in the FNP core image, nor in Multics supervisor databases, nor on any Multics config card.

A FNP with two DIA boards can be cabled to two different IOMs on a single Multics system, or to an IOM on each of two different systems. However, only one of the DIA boards may be used at a time. The prph fnp config card for the IOM channel cabled to the active DIA must have a state of on; the card for the IOM channel cabled to the inactive DIA must have a state of off.

Cabling a FNP to two different IOMs on a single Multics system offers a measure of improved reliability. If the IOM attached to the active DIA board breaks down, the IOM and its attached FNP can be deleted from the system, and the FNP can then be added to the system using the other IOM channel. However, users of the FNP at the time of the IOM failure will have to login again. If their processes had the save_on_disconnect attribute, they will be able to reconnect to their processes and continue the work which was interrupted when the IOM failed. If their processes did not have the save_on_disconnect attribute, work in progress when the IOM failed will be lost.

Cabling a FNP to the IOMs of two different systems allows the FNP to be shifted easily from one system to the other.

Multics requires that each FNP use a paging mechanism to access FNP memory beyond the first 32K words of memory. The paging mechanism on the FNP pager board can be disabled for testing purposes, but Multics requires that it be enabled during normal operations. Contact your CSD representative if your FNP will not operate. Ask him to insure that the paging mechanism is fully operative.

For a summary of switches to be checked before the system is brought up, see Appendix C.

CALENDAR CLOCK

The calendar clock is a 52-bit register in each SCU that contains the number of elapsed microseconds since January 1, 1901 at midnight, Greenwich mean time (GMT). To set the clock for the 4MW SCU, use the BCE clock setting function. To set the clock for the 6000 SC, use the switches on the SC maintenance panel. As explained below, enter the appropriate 12-digit octal number corresponding to the date and time via the switches labeled DATA, and then press the correct buttons.

The calendar clock MUST be set accurately; serious damage to the storage system can result if the setting is incorrect.

Setting Calendar Clock in 4MW SCU

Step-by-step procedures for setting the calendar clock in the 4MW SCU are available in the *Operator's Guide to Multics*, Order No. GB61.

Setting Calendar Clock in 6000 SCU

1. After leaving the "early" BCE command level, the BCE clock setting function is invoked. You must ensure that the clok configuration card specifies the correct time zone. All times entered should be in local time.

2. BCE will ask you a question of the form:..

```
The current system time is DATE TIME.
Is this correct?
```

3. You may reply "abort" to return to the "early" command level, "yes," or "no." If you answer "no," BCE will prompt you for the time with:

```
Enter time:
```

to which you should provide the current local time, in any form acceptable to the convert_date_to_binary_ subroutine. For example:

```
year-month-day hour:minutes
```

Choose a figure that is slightly (a minute or less) in advance of the current time, to allow time for the next steps to be performed.

4. BCE will then respond with:

```
SCU Switches (octal) TTTTTT TTTTTT
```

5. BCE will prompt with:

```
Enter anything after the switches have been set.
```

6. At the CPU, place the STEP CONTROL selector switch on the maintenance panel in the MEM position. *

7. At the SC (which must be in TEST mode), enter the number TTTTTT TTTTTT in the upper row of the DATA switches. Enter all zeros in the lower row of the DATA switches.

8. Press the INITIALIZE and the LOAD CLOCK pushbuttons simultaneously, at the instant when the current time reaches the time that was typed.

9. Turn the STEP CONTROL selector switch on the CPU to OFF and press the STEP pushbutton.

10. Enter "y".

11. BCE will repeat the question in step 2. This should be answered appropriately.

SECTION 4

COMMUNICATING WITH THE SYSTEM

THE BOOTLOAD CONSOLE

You may use the bootload console to issue Multics initializer commands, commands to the daemons, standard Multics commands, and BCE commands when BCE is in operation. *

Effect on System Performance

Under normal circumstances, the bootload console may be used without noticeable effect on the performance of the Multics system. However, if there is a large burst of syserr messages, then the system pauses and waits for the messages to be printed before proceeding with other user or system commands. When messages are being printed, the console is in UNLOCK mode (described below). On a one-CPU system, the system does nothing else while the console is unlocked.

Console 30-Second Timer

The bootload console has a 30-second timer mechanism. When reading input from the console, if no character is typed within 30 seconds, the read operation is terminated. The 30-second timer is controlled by a switch in the maintenance panel on some models of the bootload console. This switch must be set to the ENABLED position during operation of both Multics and BCE.

Use of the Bootload Console

The bootload console operates in two modes: LOCK mode and UNLOCK mode. In LOCK mode, the console keyboard is locked and cannot be used to type input. In UNLOCK mode, either the keyboard is unlocked and ready to accept input, or the bootload console is typing output. If the bootload console is in UNLOCK mode, the keyboard continues to be unlocked after each input request has completed and the system is ready to accept another input line. The console may be returned to LOCK mode when you press the EOM button (or the RETURN key for a CSU6601 console) without typing any other character so that system messages may be printed.

On all but the model CSU6601 console, the bootload console remains in LOCK mode until you press the REQUEST button. When the keyboard is unlocked you may type input. The END OF MESSAGE (EOM) button must be pressed to signal that the input line has been completed. When the system has processed an input line and is ready to accept another, the console keyboard is unlocked. Then you can type the next input line without pressing the REQUEST button. In UNLOCK mode, any output generated as a result of an input line may be stopped by pressing the REQUEST button. However, syserr messages cannot be stopped. Also, when the REQUEST button is pushed, the line currently being typed is finished and one more line of output is typed before the output is stopped.

On the model CSU6601 console, the system prompts you when it is expecting input. The prompt consists of the characters "M->" at the beginning of a line when either BCE or Multics is running. When the CSU6601 console is in LOCK mode, you can unlock the console for input by pressing the RETURN key on the keyboard. After the system responds with the appropriate prompt, you may type a line of input. This line is terminated by pressing the RETURN key, at which time the console is placed in LOCK mode.

Another way that the bootload console may be taken out of UNLOCK mode and put back into LOCK mode is by runout of the 30-second timer. When the bootload console is UNLOCKED and ready to accept input, if you do not type a character within 30 seconds, the timer runs out and the bootload console is placed in LOCK mode. In this event, any read operation is terminated, and any input typed on the current line is lost. Similar actions take place if any of the following events occur:

- You type an invalid character
- You turn the ONLINE/OFFLINE switch to OFF
- You turn the POWER switch on the console maintenance panel to OFF

The maximum number of input characters that can be typed on one line from a bootload console is 84. If this number is exceeded, or if the OPERATOR ERROR button is pushed, the read operation is terminated and all input typed on the current line is discarded. However, the operator console remains in UNLOCK mode.

The bootload console occasionally jams and will not respond to the REQUEST button. The set_system_console command may be used to return the console to its previous (unjammed) state. This command requires access to the highly privileged gate hphcs_, and is documented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64. Note that this is not an initializer command -- it must be executed in a privileged user process or in admin mode in the initializer process.

The substty command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) is useful in cases of an inoperative bootload console when there is no alternate console. It may be used to switch the output from the inoperative console to an initializer terminal that is working.

RCP messages and other syserr traffic are not handled by the message coordinator. This means that message coordinator commands cannot be used to manipulate these messages. For example, the reroute command cannot be used to reroute these messages from the bootload console to an initializer terminal.

The set_system_console command mentioned earlier may be used to stop ring zero from sending syserr messages to the console. The messages are automatically rerouted to the first initializer terminal accepted during the current bootload.

THE MULTIDROP INTERFACE (MDI) FOR IMUS

On systems that have IMUs as I/O multiplexers, you must communicate with the maintenance channel adapter (MCA). The MCA controls the hardware functions of system booting, IMU maintenance (e.g., IPC firmware loading), and some hardware control functions for the IMU. There are three hardware components involved in communicating with the MCA. These are: the MCA, the console channel in the IMU (IPC-CONS), and a console or terminal. These are connected together to form the multidrop interface.

The multidrop interface (MDI) connects MCAs and IPC-CONSs in a daisy-chain configuration. The MDI requires at least one IPC-CONS (connected to a console) and one MCA. Multiple IPC-CONSs and MCAs may be connected; however, only one console may be enabled on the MDI. This console is the master console; all other consoles are slaves. The master console is the one to which all operator communications to all connected MCAs are routed. There are commands that will allow the master console designation to be moved from the current master to a slave; however, that slave IPC-CONS must have a console or terminal connected.

Each MCA must be able to reach a master console. If multiple IMUs are configured, only one is required to have an IPC-CONS. This IPC-CONS must be connected to a console or terminal and be the master console on the MDI. If the bootload console is on the IMU, it may be used as the master. If not, a separate console is required for the MDI. This console must be described in the Multics configuration deck as "alt". All the MCAs and IPC-CONSs on the system may be connected together on one MDI, or each MCA and IPC-CONS in an IMU can be a separate MDI, or the components may be combined in other ways, but they must meet all the hardware and software requirements for an MDI.

Because the master console may be the bootload console, the IPC-CONS in the IMU uses an escape sequence to communicate with an MCA. This convention uses a "#" character to determine if the input is for the MDI. There can be more than one MCA connected to the MDI; therefore, the MCA number must follow the "#" character. To show that the MDI is active, a ">" character is printed by the console indicating the message will be for an MCA. All commands to the MCA are prefixed this way. All output messages are prefixed similarly with "#nn<," where "#" is an indication that this is from an MCA, "nn" is the MCA number, and "<" indicates that this is an output message. For example, to set the date and time in the MCA, type:

```
#01>time 111485,120000
```

The MCA will respond with:

```
#01<Monday November 14, 1985.  11/14/85 (12:00:00)
```

The MCA number is determined by rocker switches on the MCA board in the IMU. These must be set to a number corresponding to the IMU's letter on its iom card in the config deck (e.g., 00 = a, 01 = b, etc.).

This convention of prefixing the input messages with the "#" character only applies when the system does not have the console open for read. If the console is open for read (normal operator input), "#" may still be used to delete a character. If it is necessary to input to the MCA when the console is open for read, an "ESC#nn" prefixes the message.

While the ability to tell the MCA the date and time as described above has no impact on the Multics system, some of the maintenance functions may have adverse effects. Also, by obtaining control of the MCA, control may be obtained over the IMU and therefore over all devices connected to it. There are no explicit commands to read or write devices; however, the potential to compromise data exists. This can be considered a security risk.

To prevent possible adverse effects and avoid the security risk, console input to the MCA can be disabled (locked) by using the BCE lock_mca command. This command locks console input to the MCA; output from the MCA is still displayed on the console. The lock_mca command must be issued before the BCE boot command. If lock_mca is not invoked, the default mode of operation for the MCA is to have its input enabled (unlocked). To unlock an MCA which is locked, use the BCE unlock_mca command. The lock_mca and unlock_mca commands are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

THE INITIALIZER TERMINAL

If more than one terminal channel is connected to the initializer, the output from the various sources (e.g., daemon processes) can be routed to divide the work among several terminals. For example, all the daemons could be handled by one terminal, and the answering service could use another. Or, if all the terminals are inoperative, the system can be run completely from the bootload console.

All terminals attached to the initializer may input initializer commands. (It is possible to restrict a terminal to only certain commands.) It is sometimes difficult to input an initializer command between output messages on an initializer terminal, because the system keeps interrupting. If you type an empty line on an initializer terminal, the system responds:

```
OPER:
```

and suspends output on that terminal channel. When you complete your command, the output is restarted, with no message lost. If you do not finish your command in one minute, the output is restarted. (On the bootload console, no output happens while a read is open.)

Terminals may also be added to the initializer dynamically. To do this, dial a terminal into Multics as if you were going to log in, but instead of typing login, issue a dial command:

```
Multics 9.0: PCO, Phoenix, Az.
Load = 41.0 out of 110.0 units: users = 41

! dial system
```

The dialed terminal gets a message of the form:

```
TN300 405 chn a.h003 dialed to Initializer.
```

Also, a message stating that the terminal has dialed in is routed through the message coordinator from the source "as" on the switch severity1 to wherever the system_start_up.ec has routed "as" messages. This might be the bootload console, an initializer terminal, or nowhere. The message looks like this:

```
1137 as dial_ctl_:  channel a.h013 dialed to Initializer
```

You should then issue a series of commands to accept the terminal channel and to route output to it.

```
! accept a.h003 reply dump otw_
  Ready (User_name)
! define vc2 tty a.h003
  Ready (User_name)
! route dump user_i/o vc2
  Ready (User_name)
```

The response on the dialed terminal is a message saying that the initializer has attached the channel:

```
channel a.h003 attached by Message Coordinator.
```

followed by whatever messages are routed to the terminal channel.

When you are finished with a dialed terminal, or if a curious user tries to dial the initializer without permission, you may disconnect the channel from the initializer and make it available for dialups again by typing a drop command:

```
! drop a.h003
  Ready (User_name)
```

The response on the dialed terminal is a message similar to "please reissue dial command" and at this point the terminal may be redialed, or used for regular logins, or hung up.

SECTION 5

BOOTLOAD OPERATING SYSTEM

The information that was in this section is obsolete and has been deleted.

SECTION 6

BOOTLOAD COMMAND ENVIRONMENT

BOOTLOAD COMMAND ENVIRONMENT DESCRIPTION

The bootload command environment (BCE) comprises a set of programs for performing functions such as bootloading Multics, dumping and patching main memory and disks, and initiating an emergency shutdown of Multics.

BCE is contained within the first two collections of modules on the Multics system tape. It consists of the following major parts:

1. collection zero routines
   a series of programs read in from tape by the IOM which load the other BCE programs into memory and load firmware into the bootload tape controller, if necessary.

2. collection one initialization
   a series of programs that are part of Multics initialization proper that also initialize the bootload command environment.

3. toehold program
   a small program permanently residing in main memory at absolute location 24000 (octal). It communicates closely with Multics to pass control back and forth between Multics and BCE.

4. bootload command utilities
   a series of programs which provide the BCE command level.

5. command programs
   a number of programs that perform the operator directed functions of BCE.

CONFIGURATION REQUIREMENTS

BCE requires a bootload console. In case of bootload console failure, BCE searches the configuration deck for an alternate console. If it finds one, the first console becomes inoperative. If it doesn't find one, BCE crashes. (See the description of the prph opc config card in Section 7.)

BCE requires 512K of contiguous low order memory. All of BCE's functions can be performed within this memory.

Two special regions of the RPV are used by BCE. These two special regions have locations recorded in the label of the RPV. The first is the FILE partition, which contains a simple file system used by BCE to hold BCE exec_coms and ASCII sources of configuration files. The second is the BCE partition, used by BCE to hold the following:

- BCE itself and BCE command programs
- The programs needed to boot Multics
- A saved copy of memory used by Multics when BCE is invoked after a crash

LOADING BCE

BCE is loaded via the IOM. Step-by-step procedures for bootloading BCE are available in the *Operator's Guide to Multics*, Order No. GB61.

Cold Booting BCE

A cold boot of BCE and Multics recreates the entire storage system hierarchy on a particular RPV, discarding previous hierarchies, including all user files. Therefore, you shouldn't do a cold boot unless you're sure you want to discard the existing hierarchy; i.e., you shouldn't do a cold boot unless the Multics storage system is either nonexistent or has been destroyed. Proceed as if you were doing a regular boot. Then, when you get the prompt:

```
Enter rpv data:
```

answer it with:

```
cold Tchan msp_model drive_model drive_number{sv}
```

where:

T
: is the tag of the IOM to which the bootload disk controller (the one controlling the disk drive on which the RPV is mounted) is connected (a, b, c or d).

chan
: is the number (in decimal) of the IOM channel to which the bootload disk controller is connected (e.g., 24).

msp_model
: is the model number (in decimal) of the bootload disk controller.

Valid model numbers are:

```
400     (MSP0400)
451     (MSP0451, DSC0451)
601     (MSP0601)
603     (MSP0603)
607     (MSP0607)
609     (MSP0609)
611     (MSP0611)
612     (MSP0612)
800     (MSP8021, MSP8022, MSP8023)
ipc     (IPC-FIPS)
```

drive_model
: is the model number (in decimal) of the disk drive on which the RPV is mounted. Valid model numbers are:

```
400     (MSU0400)
402     (MSU0402)
451     (MSU0451)
500     (MSU0500)
501     (MSU0501)
3380    (MSU3380)
3381    (MSU3381)
```

drive_number{sv}
: is the number of the disk drive and, if drive_model is 3380 or 3381, the name of the subvolume on which the RPV is located. The valid subvolume names for MSU3380s are a and b. The valid subvolume names for MSU3381s are a, b, and c. A subvolume name must be specified for 3380 and 3381 devices; it must be omitted for all other disk drive types. An example of drive_number for a 451 is "1". An example of drive_number{sv} for a 3380 is "1b".

The system will enter the init_vol request loop. (The init_vol request loop is described under the init_vol command in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.) At this time, you must enter the attributes of the RPV.

Then, when BCE comes to the "early" command level, you must enter the config deck. (The BCE commands to do this are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64). When you are finished, again proceed as if you were doing a regular boot.

Some Special Requests

When the system asks you to enter the boot tape MPC model, you can stop (crash) initialization by typing "shut." If firmware should not be loaded into this MPC for some reason, you can prevent it from being loaded by typing "ipc."

When the system asks you to enter the RPV data, you can abort booting by typing "shut." If firmware should not be loaded into the described MPC for some reason, you can suppress the load by typing "skip" before you type "rpv" or "cold."

Error Recovery during BCE Boot

There are several different points during the boot process at which attempts are made to allow for error recovery. The methods depend on the point within the boot sequence. The following paragraphs discuss the recovery attempts by describing some aspects of the internal operation of the boot sequence.

When you boot BCE directly from the IOM, it executes collection 0 initialization, which reads in collection 1 (BCE proper). A config deck is synthesized from the knowledge of the hardware found during this pass and through questions to the operator. A first pass is made through collection 1 to find the RPV and to read in the last config deck saved in the CONF partition on disk. If an error occurs before this point, most likely a hardware or software failure, the early dump facility is invoked (see "The Early Dump Facility" later in this section). Otherwise, this environment (memory and the synthesized config deck) is saved on disk. The "early" command level is then entered. (Note that time stamps may be wrong at the "early" level.) At this point, you must make sure the config deck (read from disk) is correct. Then you may enter "bce" to actually boot BCE. Initialization continues with a second pass through collection 1. If this pass fails, most likely due to either a hardware problem or an error in the config deck, the saved environment is restored and you are returned to the "early" command level. You may then retry the boot. Eventually this will succeed and BCE will come to the "boot" command level, having saved this new environment and config deck.

Once at the "boot" command level, you may perform BCE functions. To boot Multics, enter "boot". Another pass through collection 1 is made to set up for Multics. If an error occurs during this pass (most likely a hardware problem or a bad config deck), the environment saved above is restored and you are returned to the "bce_crash" command level. Also, if a BCE utility should fail or should encounter a BCE breakpoint, this environment is restored and "bce_crash" level entered. (For a discussion of BCE breakpoints, refer to the description of the BCE probe command in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.) At this time, you may enter "crash" level commands to examine the failed image (or to debug BCE), or "boot" level commands to fix the config deck (if necessary) and to retry the boot of Multics.

An important thing to remember about coming to the "bce_crash" or returning to the "early" command levels is that they use an environment and config deck declared safe on a previous initialization pass. As such, not all devices listed in the "current" config deck (the one visible with the config deck editor) may be accessible at this level. Generally speaking, to access all devices, the config deck must be correct and an initialization pass (the "boot" pass) must be made. If you are in doubt, entering "reinitialize" will run another initialization pass.

Once the "service" pass of collection 1 completes, any further failures of initialization or of Multics itself return to the "crash" command level, used for examining the crash. At this time, the config deck as used by Multics is used. This is done to take into account any reconfigurations performed by Multics. At the "crash" level, you should take a dump and perform an emergency shutdown.

Config Deck and Device Accessibility

When Multics is running, the set of devices that are accessible (to the system as a whole) are precisely those described by the config deck. The config deck is kept up to date with the state of the devices. However, the real state of devices and their accessibility is described by various control tables within Multics. One of the main purposes of BCE is to set up these control tables. Since BCE allows arbitrary text editing on the config deck, it follows that the state of the control tables may not match that of the config deck. The following paragraphs describe some of these subtleties.

When you first boot BCE from tape, collection 0 constructs a config deck based on operator responses and some hardware switches. This config deck is used to construct the control tables at the "early" level. As a result, this config deck and the control tables only describe the bootload tape drive, the RPV, and the bootload processor. Some fields in the config deck will be incorrect, such as CPU model numbers. At the "early" command level, you must make sure that the config deck accurately describes all hardware units. These units are not accessible at this time, however.

An attempted boot to "boot" command level builds control tables describing all of these hardware units. If this boot succeeds, all of these units are accessible from BCE. If it fails, BCE returns to "early" command level with only the initial hardware units accessible.

At the "boot" command level, you may again change the config deck. Any units added, for example, will not be accessible at this time, since the control tables do not describe them. However, if you boot Multics, Multics will be able to access them all, since a Multics boot builds control tables for them all. If this boot fails, BCE returns to the "bce_crash" command level, with these new changes not described in the control tables (but visible in the config deck).

Any changes made to the config deck become reflected in the control tables in only one of two ways. The first is by your booting to the next BCE command level or to Multics. If the config deck is correct, the devices become accessible. The other way is by your entering "reinitialize," which runs a new initialization pass and returns to the "boot" command level. If this succeeds, the devices become accessible. If it fails, BCE returns to "bce_crash" level, without the changes being made.

BCE TOEHOLD

The BCE toehold is a program that resides in main memory. The toehold communicates very closely with BCE and Multics as follows.

When Multics is running, the toehold may be invoked by manually forcing the processor to execute an XED 24000 (octal) interrupt inhibited instruction. The CPU must be in TEST mode when the XED instruction is executed. The toehold saves the processor registers and the 512K of low memory. It then reads in a saved copy of BCE from the RPV and transfers control to it. BCE then enters its command level with a prompt of:

```
bce (crash) TIME:
```

The toehold is also invoked as a result of either the "go" or the "continue" command being issued within BCE. When one of these commands is issued, the toehold restores the memory image that it has previously saved and restarts the program that was originally running.

The toehold contains a flagbox of bits that may be ON or OFF and which can be read and set both by BCE and Multics.

To enter BCE manually on a Level 68 system, execute switches with the DATA switches set to 024000717200 (XED 24000 interrupt inhibited). To enter BCE manually on a DPS 8 system, use the BCE 24000 command.

Step-by-step procedures for executing switches on both a Level 68 system and a DPS 8 system are available in the *Operator's Guide to Multics*, Order No. GB61.

THE EARLY DUMP FACILITY

The early dump facility is a facility within BCE that is capable of saving an image of memory to tape when a system failure occurs during collection 1 initialization. It resides at a fixed location in memory whenever BCE is running (30000 octal). It is invoked automatically whenever a hardware or software error is detected prior to the establishment of the BCE toehold. It can also be entered manually, whenever BCE is present (but definitely *NOT* when Multics is running), by forcing an XED 24004. This is done in a manner similar to forcing a manual return to BCE, except that the value entered into the DATA switches is 024004717200 (XED 24004 interrupt inhibited).

Once entered, the early dump facility may print a flagbox message. It will always prompt with:

```
Enter tape drive number for memory dump:
```

You should reply with the number of a tape drive controlled by the bootload tape controller on which a tape is mounted for writing. Memory will be dumped onto this tape at a density of 1600. After performing the dump, BCE will disable itself.

The tape written by this facility can be read with the read_early_dump_tape (redt) command and analyzed with the analyze_multics (asm) command, both of which are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

BCE COMMAND LANGUAGE

The command language used within BCE is the normal Multics command language (actually the ssu_ request language). Do not confuse this with the command language used at the initializer's ring 1 command level. (Refer to the *Multics Programmer's Reference Manual*, Order No. AG91, for a description of Multics command/subsystem language.) Full support for active functions, iteration sets, etc. is provided.

Commands to BCE may only be issued at the bootload console. Standard typing conventions apply. It is also possible for BCE commands to be placed in exec_coms. Exec_coms are ASCII files containing commands and possible input to commands. They are run with the "exec_com" command.

There are three ways to edit a BCE exec_com. One way is to use the BCE qedx command within BCE. This way is not recommended. A second way is to use the bootload_fs command to copy the exec_com into the Multics storage system, use a Multics text editor to edit the exec_com, and use the bootload_fs command to copy the exec_com back into the BCE file system. A third way is to get a copy of the exec_com from the system library, edit it with a Multics text editor, return it to the system library, and then use the generate_mst command to create a new system tape. Both the bootload_fs command and the generate_mst command are documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

Also, a command may be placed in the flagbox within BCE or Multics for BCE to execute whenever Multics crashes or shuts down.

Whenever BCE is at command level, it responds with a ready message like the following:

```
bce (state) TIME:
```

where "state" is one of the following:

early
: indicates that system is ready to boot BCE.

boot
: indicates that system is ready to boot Multics.

crash
: indicates that Multics has crashed.

bce_crash
indicates that BCE has crashed.

See Figure 6-1 for an illustration of the BCE states and some commands/events that change them.

Some commands have subrequests to them, such as qedx and probe. The conventions for request lines entered for such commands vary from command to command.

BCE COMMANDS

Complete descriptions of the BCE commands are presented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

ABORTING BCE COMMANDS

Whenever you push the REQUEST button on the console (or the RETURN key on the CSU6601) and BCE has not solicited this request from you, the BCE abort routine is entered. This routine allows BCE operations to be aborted to various extents. When called, the abort function prompts you (on the console) with:

```
Abort?
```

You may give various answers to this question. If you hit the REQUEST button accidentally, you may enter "no" or "n" to return to the interrupted operation. Answering "yes" or "y" aborts the operation. If the operation was a sub-request, only the sub-request is aborted. Otherwise, the command in question is aborted, returning either to the exec_com which called it, if one was present, or to BCE command level. Answering "request," "req" or "r" is equivalent to answering "yes." Answering "command," "com" or "c" aborts the current command, regardless of whether a sub-request was in execution or not. Finally, answering "all" or "a" aborts anything in execution, returning to BCE command level.

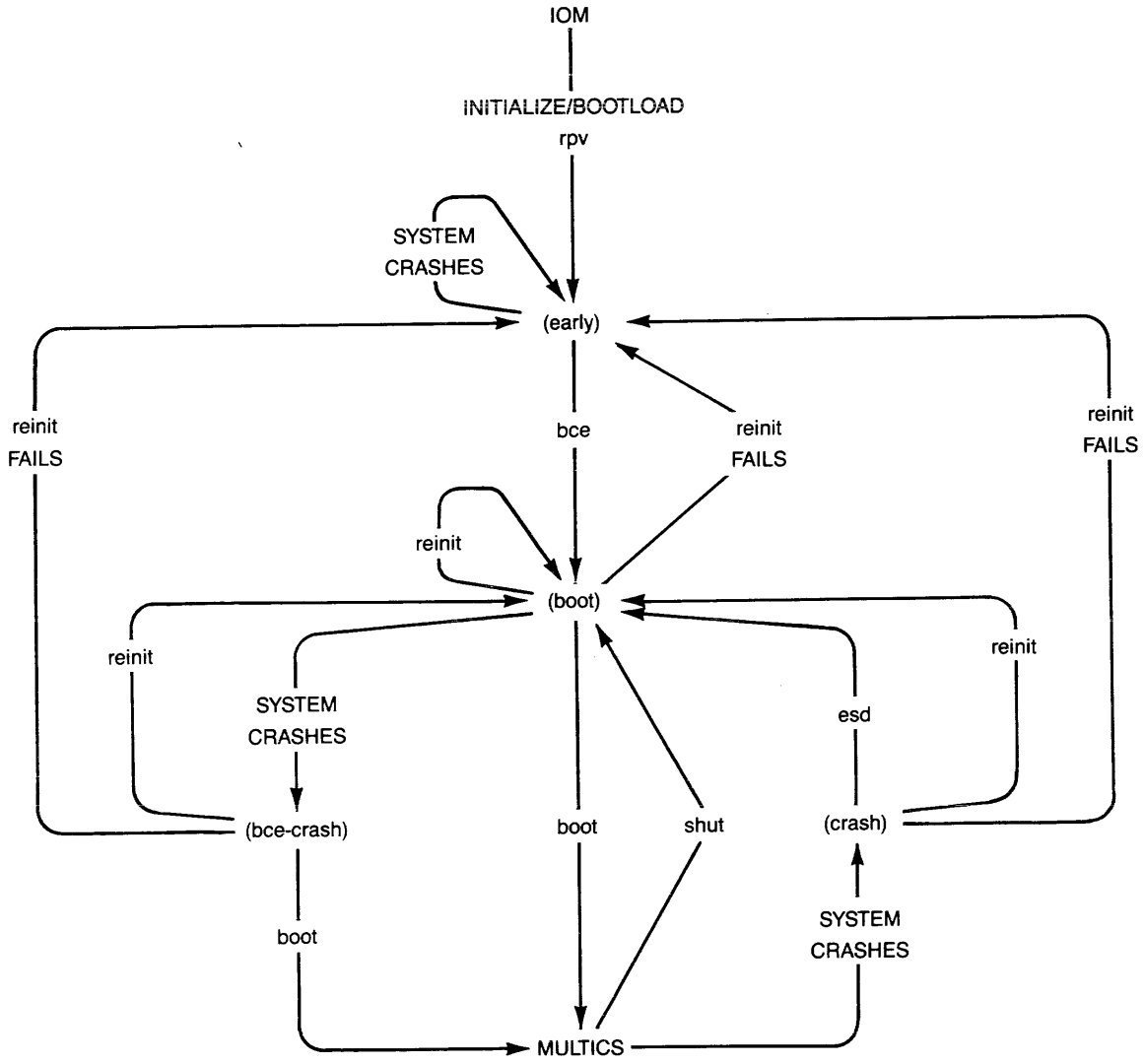


Figure 6-1. BCE States and Commands/Events That Change Them

SECTION 7

MULTICS CONFIGURATION DESCRIPTION

MULTICS CONFIGURATION FILE

The following discussion describes records in the config (configuration) file. Information in this file describes the hardware configuration, tells the system of switch settings and operational readiness of specific hardware and peripheral devices, and sets several system tuning parameters. This information is specified while in BCE, and passed by BCE to Multics; the system software considers this information the configuration under which Multics must run.

Historically, the configuration records were defined by a set of cards in a config deck. Today, the config deck is replaced by a config file in which each line defines a configuration record. The older terminology, however, still appear in the documentation and in the name of the Multics print_configuration_deck command.

Old Terminology	New Terminology
config deck	config file
config card	config record (line in config file)

The configuration file is stored by BCE in the CONF partition. It can be input from a file on the Multics system tape or from the bootload console.

Each configuration record described in this section includes a standard format and a labeled format illustration. In the labeled format, each value on a config record, except the name of the record, can optionally be preceded by a field label. Labeled fields can appear in any order. The interpretation of a config record in labeled form is that all labeled field values are ordered according to standard format; any unlabeled value, then, fill in the missing fields. Thus,

```
iom -state on -port 1 a iom
```

becomes

```
iom a 1 iom on
```

in its standard format. Config records in labeled format are easier to understand than records in standard format because each field is preceded by an identifier.

Each site has its own set of config records that define the hardcore configuration, switch settings, and software tuning parameters used by the site. The format of config records is the same for every site. The particular config records, their order, and their field values vary to conform to the equipment configuration at that site.

Config records can be divided into five categories:

1. configuration of major hardware mainframe modules: cpu, iom, mem
2. configuration of peripheral controllers and devices: chnl, ipc, mpc, prph, udsk
3. descriptions of software parameters: clok, schd, sst, tcd
4. parameters of the storage system: part, root, salv
5. specialized: dbmj, intk, parm, tbls.

General Description of Config Records

All records in the config file contain free-formatted individual fields separated by one or more blank characters. Numbers on config records are usually octal (the part and root records are exceptions). Decimal numbers are represented by placing a decimal point immediately after the number (e.g., 10. indicates decimal ten). In some record fields, numbers 1 through 8 may be represented by the letters a through h, respectively. See examples listed under individual records.

Listing the Config File in BCE

After the config file has been read by BCE at bootload time, it may be listed by typing the BCE config command to enter the config file editor, and then typing "1,$p" at the bootload console. The BCE config command is described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

Listing the Config File in Multics

When Multics is running, you may list the config deck with the print_configuration_deck (pcd) command, described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

Sample Configuration Files

Here is a sample configuration file for a large system, with the records in general format:

```
clok 5 est 24.
iom a 0 iom on
iom b 1 iom on
iom c 2 imu on
cpu a 7 on dps8 70. 32.
```

```
cpu b 6 on dsp8 70. 16.
cpu c 5 on dsp8 70.  8.
cpu d 4 on dsp8 70.  8.
mem a 2048. on
mem b 2048. on
mem c 2048. on
mem d 1024. on
mpc mtpa 611. a 14. 1
mpc mtpb 611. b 14. 1
mpc mspa 612. a 16. 4
mpc mspb 612. a 20. 4
mpc mspc 612. a 24. 4
mpc mspd 612. b 16. 4
mpc mspe 612. b 20. 4
mpc mspf 612. b 24. 4
mpc urpa 8001. a 10. 2
mpc urpb 8001. a 12. 2
prph dska a 16. 4 501. 8 0 22. 451. 2
chnl dska b 16. 4
prph dskb a 20. 4 0 8 501. 8. 0 12. 451. 2
chnl dskb b 20. 4
prph dskc a 24. 4 0 16. 501. 3 0 6 451. 2
chnl dskc a 24. 4
prph dskg c 16. 4 3380. 16.
chnl dskg c 20. 4
prph tapa a 14. 1 610. 6
chnl tapa b 14. 1
prph opca a 31. 6601. 80. on
prph opcb a 24. 6004. 80. alt
prph opcc c 14. 6601. 80. alt
prph prta a 10. 1201. 600. 136.
prph prtb a 11. 1201. 600. 136.
prph rdra a 12. 500.
prph fnpa a 28. 6670. on
prph fnpb b 28. 6670. on
prph fnpc a 29. 6670. on
prph fnpd b 29. 6670. on
ipc fips c 16. 4
ipc fips c 20. 4
root dskc 30 dskb 31 dske 28
part dump dskc 30
schd 400000 4 10 100 2 20.
tcd 200. 600.
sst 3000. 1500. 1000. 200.
udsk dska 2
parm ttyb 61440. ccrf
tbls scav 140.
dbmj 64. 700. 400. 150. 60. 25.
```

Here is the same config file, but with the records in labeled format:

```
clok -delta 5 -zone est -boot_delta 24.
iom -tag a -port 0 -model iom -state on
iom -tag b -port 1 -model iom -state on
iom -tag c -port 2 -model imu -state on
```

```
cpu -tag a -port 7 -state on -type dps8 -model 70. -cache 32.
cpu -tag b -port 6 -state on -type dps8 -model 70. -cache 16.
cpu -tag c -port 5 -state on -type dps8 -model 70. -cache 8.
cpu -tag d -port 4 -state on -type dps8 -model 70. -cache 8.
mem -port a -size 2048. -state on
mem -port b -size 2048. -state on
mem -port c -size 2048. -state on
mem -port d -size 1024. -state on
mpc -ctlr mtpa -model 611. -iom a -chn 14. -nchan 1
mpc -ctlr mtpb -model 611. -iom b -chn 14. -nchan 1
mpc -ctlr mspa -model 612. -iom a -chn 16. -nchan 4
mpc -ctlr mspb -model 612. -iom a -chn 20. -nchan 4
mpc -ctlr mspc -model 612. -iom a -chn 24. -nchan 4
mpc -ctlr mspd -model 612. -iom b -chn 16. -nchan 4
mpc -ctlr mspe -model 612. -iom b -chn 20. -nchan 4
mpc -ctlr mspf -model 612. -iom b -chn 24. -nchan 4
mpc -ctlr urpa -model 8001. -iom a -chn 10. -nchan 2
mpc -ctlr urpb -model 8001. -iom a -chn 12. -nchan 2
prph -subsys dska -iom a -chn 16. -nchan 4 -model 501. -number 8.
   -model 0 -number 22. -model 451. -number 2
chnl -subsys dska -iom b -chn 16. -nchan 4
prph -subsys dskb -iom a -chn 20. -nchan 4 -model 0 -number 8.
   -model 501. -number 8. -model 0 -number 12. -model 451.
   -number 2
chnl -subsys dskb -iom b -chn 20. -nchan 4
prph -subsys dskc -iom a -chn 24. -nchan 4 -model 0 -number 16.
   -model 501. -number 3 -model 0 -number 6. -model 451.
   -number 2
chnl -subsys dskc -iom a -chn 24. -nchan 4
prph -subsys dskg -iom c -chn 16. -nchan 4 -model 3380. -number 16.
chnl -subsys dskg -iom c -chn 20. -nchan 4
prph -subsys tapa -iom a -chn 14. -nchan 1 -model 610. -number 6
chnl -subsys tapa -iom b -chn 14. -nchan 1
prph -device opca -iom a -chn 31. -model 6601. -11 80. -state on
prph -device opcb -iom a -chn 24. -model 6004. -11 80. -state alt
prph -device opcc -iom c -chn 14. -model 6601. -11 80. -state alt
prph -device prta -iom a -chn 10. -model 1201. -train 600. -11 136.
prph -device prtb -iom a -chn 11. -model 1201. -train 600. -11 136.
prph -device rdra -iom a -chn 12. -model 500.
prph -device fnpa -iom a -chn 28. -model 6670. -state on
prph -device fnpb -iom b -chn 28. -model 6670. -state on
prph -device fnpc -iom a -chn 29. -model 6670. -state on
prph -device fnpd -iom b -chn 29. -model 6670. -state on
ipc -type fips -iom c -chn 16. -nchan 4
ipc -type fips -iom c -chn 20. -nchan 4
root -subsys dskc -drive 30 -subsys dskb -drive 31 -subsys dske
   -drive 28
part -part dump -subsys dskc -drive 30
schd -wsf 400000. -tefirst 4 -telast 10. -timax 100. -mine 2 -maxe 20.
tcd -apt 200. -itt 600.
sst -4k 3000. -16k 1500. -64k 1000. -256k 200.
udsk -subsys dska -nchan 2
parm tty b 61440. ccrf
tbls scav 140.
dbmj 64. 700. 400. 150. 60. 25.
```

Note that the parm, tbls, and dbmj records (shown) and the intk and salv records (not shown) do not have a labeled format.

Name: chnl

This record designates additional channels used to access a given disk or tape subsystem through a specified IOM. If a noncontiguous set of channels is used to access a given subsystem as, for example, through different IOMs, a chnl record must be used. Up to three additional channel groups may be specified. See the prph record for more information on disk and tape subsystems.

Format

```
chnl device_name iom1 chn1 nchan1 {...  iom4 chn4 nchan4}
```

where:

1. device_name
   is the name of the disk or tape subsystem for which channels are being specified. It must match the device name of the disk or tape subsystem on a prph record.

2. iomi
   is the tag (a, b, c or d) of the IOM that is to be used.

3. chni
   is the first logical channel (configuration dependent) through which the disk or tape subsystem is driven.

4. nchani
   is the number of logical channels (configuration dependent) to be used.

Labeled Format

```
chnl -subsys device_name -iom iom1 -chn chn1 -nchan nchan1
   {... -iom iom4 -chn chn4 -nchan nchan4}
```

Examples

```
chnl  dska  b  30.  4

chnl - subsys dska -iom b -chn 30. -nchan 4
```

Name: clok

The clok record provides information to system software about how to interpret the readings of the calendar clock in a system controller.

Format

```
clok delta zone boot_delta
```

where:

1. delta

   is the time difference (number of hours earlier than Greenwich mean time). The range of this field should be:

   ```
   -12. <= delta <= +11.
   ```

   If the value of delta is less than zero, a minus sign must be specified. (This field is ignored by BCE, but must still be specified.)

2. zone

   is up to four characters describing the time zone. The following is a list of the acceptable zone names and their corresponding delta and zone values.

clok

Zone Name	delta	zone
Nome Time	+11.	nt
Hawaiian Standard Time	+10.	hst
Yukon Standard Time	+09.	yst
Hawaiian Daylight Time	+09.	hdt
Pacific Standard Time	+08.	pst
Yukon Daylight Time	+08.	ydt
Mountain Standard Time	+07.	mst
Pacific Daylight Time	+07.	pdt
Central Standard Time	+06.	cst
Mountain Daylight Time	+06.	mdt
Eastern Standard Time	+05.	est
Central Daylight Time	+05.	cdt
Atlantic Standard Time	+04.	ast
Eastern Daylight Time	+04.	edt
Newfoundland Standard Time	+03.5	nst
Greeland Standard Time	+03.	gst
Atlantic Daylight Time	+03.	adt
Newfoundland Daylight Time	+02.5	ndt
Azores Time	+02.	at
West Africa Time	+01.	wat
Universal Time	+00.	ut
Universal Time	+00.	z
Greenwich Mean Time	+00.	gmt
Central European Time	-01.	cet
Middle Europe Time	-01.	met
Middle Europe Winter Time	-01.	mewt
British Summer Time	-01.	bst
Swedish Winter Time	-01.	swt
French Winter Time	-01.	fwt
Heure Francais d'Hiver	-01.	hfh
Middle Europe Summer Time	-02.	mest
Eastern European Time	-02.	eet
Swedish Summer Time	-02.	sst
French Summer Time	-02.	fst
Heure Francais d'Ete	-02.	hfe
Baghdad Time	-03.	bt
GMT +4 hours.	-04.	zp4
GMT +5 hours.	-05.	zp5
Indian Standard Time	-05.5	ist
GMT +6 hours.	-06.	zp6
West Australian Standard Time	-07.	wast
Java Time	-07.5	jt
West Australian Daylight Time	-08.	wadt
China Coast Time	-08.	cct
Japan Standard Time	-09.	jst
Central Australian Standard Time	-09.5	cast
South Australian Standard Time	-09.5	sast
East Australian Standard Time	-10.	east
Central Australian Daylight Time	-10.5	cadt

South Australian Daylight Time	-10.5	sadt
East Australian Daylight Time	-11.	eadt
New Zealand Standard Time	-12.	nzst
New Zealand Daylight Time	-13.	nzdt

3. boot_delta

this number reflects the site's normal interval between shutdowns and boots in hours (if the number is decimal, the decimal point must be supplied). If the system was down for more than the specified number of hours, the next time you attempt to boot the system you are informed of the "suspicious" situation and asked if you still want to boot. This control argument can be used to check for incorrect clock settings before damage is done to the storage system. The default is off.

Labeled Format

```
clok -delta delta -zone zone -boot_delta boot_delta
```

Notes

This is the record that is changed when daylight savings (or standard) time is started or stopped.

If the operator attempts to boot BCE with an unacceptable value for zone on the clok record, the system prints the following message:

```
scs_and_clock_init:  The zone on the clok
cbn
record
is not in time_info_.
```

and BCE crashes. If the system was at the "early" state, it will stay there. If it was at the "boot" state, it will go to the "bce_crash" state. If it was at the "bce_crash" state, it will stay there. The operator should correct the clok record and continuing booting. (This means typing "bce" at the "early" state or "boot" at the "bce_crash" state.)

If the operator attempts to set the calendar clock to a time which is more than boot_delta hours after the last shutdown time, the system prints the following message:

```
The current time is more than the supplied boot_delta hours beyond
the unmounted time recorded in the RPV label.  Is this correct?
```

The operator should answer the question, and if necessary, reenter the time.

Examples

```
clok  +05.  est 24.
clok  -02.  eet
clok  +11.  nt

clok -delta +05. -zone est -boot_delta 24.
clok -delta -02. -zone eet
clok -delta +11. -zone nt
```

Name: cpu

The cpu identifies a processor in the system configuration.

Format

```
cpu tag port state {type} {model} {cache_size}
```

where:

1. tag
   is a letter (a through h) corresponding to the processor number (0 through 7) set in the processor configuration switches.

2. port
   is a number (0 through 7) corresponding to the system controller port to which the processor is connected. It is strongly recommended that IOMs be configured on lower-numbered SCU ports than CPUs.

3. state
   is either on or off. On signifies that the processor is configured at the time Multics is bootloaded. Off signifies that the processor can be dynamically added to the configuration at a later time.

4. type
   is either l68, dps8, or dps.

5. model
   is the model number of the processor. The model number is 60. for L68 and DPS processors with no cache and 80. for those with 2K cache. The model number is either 70., 62. or 52. for a DPS8 processor, depending on which submodel you have.

6. cache_size
   is the cache size of the processor expressed in Kilo-words (1kw = 1024 words).

Labeled Format

```
cpu -tag tag -port port -state state
   {-type type -model model -cache cache_size}
```

Examples

```
cpu  a  7   on  dps8 70. 32.
cpu  b  6   on  dps8 62. 16.
cpu  c  5   on  dps8 52.  8.
cpu  d  4   off 168  80.  2.
cpu  e  3   on  dps  80.  2.
cpu  f  2   off dps  60.  0.

cpu -tag a -port 7 -state on -type dps8 -model 70. -cache 32.
cpu -tag b -port 6 -state on -type dps8 -model 62. -cache 16.
cpu -tag c -port 5 -state on -type dps8 -model 52. -cache 8.
cpu -tag d -port 4 -state off -type 168 -model 80. -cache 2.
cpu -tag e -port 3 -state on -type dps -model 80. -cache 2.
cpu -tag f -port 2 -state off -type dps -model 60. -cache 0.
```

Notes

The CPU type, model number and cache size may be optionally specified when the config file is built in BCE. When the system is booted or a CPU is added to the configuration, internal hardware registers are read to determine the actual CPU type, model number and cache size. Then, if these fields have been specified on the cpu config record image, they are checked for correctness. If any of them have been specified incorrectly, a message is sent which sounds the alarm. The discrepant fields are then corrected by the software, and the updated cpu config record image is restored into the config file segment. If these optional fields have not been specified, the software determines what their values should be and updates the cpu config record image in the config file segment without sending a message. If for some reason the CPU model number can not be determined by reading the appropriate config registers within the CPU, the CPU model number field is set to 77..

dbmj

Name: dbmj

The dbmj (Database Management Journals) record sets up dm_journal_seg_, and also sets various limits on synch-held pages.

Format

```
dbmj max_journals max_pages ast1 ast2 ast3 ast4 where:
```

1. max_journals
   is the maximum number of before journals allowed.

2. max_pages
   is the maximum number of database management pages that may be held in memory at any given time.

3. ast1
   is the maximum number of database management segments allowed to have a 4K AST pool entry.

4. ast2
   is the maximum number of database management segments allowed to have a 16K AST pool entry.

5. ast3
   is the maximum number of database management segments allowed to have a 64K AST pool entry.

6. ast4
   is the maximum number of database management segments allowed to have a 256K AST pool entry.

Examples

```
dbmj 64.  700.  400.  150.  60.  25.
```

Name: intk

The intk record is not physically present in the config file. It is a record image set up by BCE in the main memory-resident image of the config file at bootload time, and used to tell Multics whether or not to automatically start up the answering service.

Format

```
intk boot drive p1 p2 ... pN
```

where:

1. boot
   is either warm or cold to specify whether the system is to be brought up to a warm or cold bootload.

2. drive
   is the tape drive from which the system tape is booted.

3. pi
   are arguments typed to the BCE boot command other than warm, cold, or the tape number. These arguments enable special options during system startup.

Example

```
intk warm 3 star
```

Name: iom

The iom record describes an input/output mainframe (IOM or IMU) as part of the system configuration.

Format

`iom tag port model state` where:

1. tag
   is a letter (a, b, c or d) that identifies the IOM or IMU.

2. port
   is the system controller port (0 through 7) to which the IOM or IMU is connected. It is strongly recommended that I/O mainframes be configured on lower-numbered SC ports than CPUs.

3. model

   is either iom, indicating that this I/O mainframe is an IOM, or imu, indicating that this I/O mainframe is an IMU.

4. state

   is either on, indicating that the I/O mainframe may be used by the system, or off, indicating that it may not be used at this time. If off, it may be added to the configuration at a later time.

Labeled Format

```
iom -tag tag -port port -model model -state state
```

Examples

```
iom a 0 iom on

iom c 3 imu on

iom -tag a -port 0 -model iom -state on

iom -tag c -port 3 -model imu -state on
```

Name: ipc

The ipc record is used in the configuration file to associate channel numbers with IPC FIPS controllers. FIPS controllers are only supported for the IMU type of I/O mainframe. The physical channels for this type of I/O mainframe are called IPCs. There must be a separate ipc record for each IPC FIPS controller configured on the system.

Format

```
ipc type iom chn nchan
```

where:

1. type

   is the type of the IPC. Only the "fips" type must be described in the config file.

2. iom

   is the tag (a, b, c, or d) of the IMU to which the IPC is connected.

3. chn
   is the starting logical channel number for this IPC.

4. nchan
   is the number of logical channels for this IPC.

Labeled Format

```
ipc -type fips -iom iom -chn chn -nchan nchan
```

Examples

```
ipc fips a 20. 2

ipc -type fips -iom a -chn 20. -nchan 2
```

Name: mem

The mem record defines the system controllers that are part of the system configuration. There is one mem record for each system controller configured in the system. These mem records must be placed in the config file in the order in which the memories are configured, the lowest-order memory (lowest address) first and the highest (highest address) last.

Format

```
mem port size state
```

where:

1. port
   is a value (a through h) that corresponds to the number of the active module port to which the system controller is connected.

2. size
   is the number of 1024 (2000 octal) word blocks of memory in the controller.

3. state
   is either on or off. On signifies that the memory is actively connected at the time Multics is bootloaded. Off signifies that the memory is available and, while not actively connected, may be brought into the system configuration dynamically at a later time.

Labeled Format

```
mem -port port -size size -state state
```

Examples

```
mem  a 1024.  on
mem  b 1024.  on
mem  c 1024.  off

mem -port a -size 1024. -state on
mem -port b -size 1024. -state on
mem -port c -size 1024. -state off
```

Name: mpc

The mpc record is used in the configuration file to associate channel numbers with microprogrammed peripheral controllers (MPCs) and with physical links to MPCs. On this record, the IOM number, the base channel number, and number of channels for each physical link (PSIA channel) to the MPC is given. There must be a separate mpc record for each MPC configured into the system.

Format

```
mpc ctlr_name ctlr_model iom1 chan1 nchan1 {...  iom4 chan4 nchan4}
```

where:

1. ctlr_name

   is the controller name of the MPC. Controller names must be unique -- no two on the system can be the same. Valid controller names are:

   ```
   mtpx          for tape controllers
   mspx          for disk (mass storage) controllers
   urpx          for unit record controllers
   ```

   where x can be any alphanumeric character that makes the name unique.

2. ctlr_model

is the model number of the MPC. The following is a list of supported MPCs and their corresponding ctlr_name and ctlr_model values.

MPC_type	ctlr_name	ctlr_model
MTC501	mtp	501.
MTC502	mtp	502.
MTC0602	mtp	602.
MTP0600	mtp	600.
MTP0601	mtp	601.
MTP0610	mtp	610.
MTP0611	mtp	611.
MTP8021	mtp	611.
MTP8022	mtp	611.
MTP8023	mtp	611.
MSP0400	msp	400.
DSC0451	msp	451.
MSP0451	msp	451.
MSP0601	msp	601.
MSP0603	msp	603.
MSP0607	msp	607.
MSP0609	msp	609.
MSP0611	msp	611.
MSP0612	msp	612.
MSP8021	msp	800.
MSP8022	msp	800.
MSP8023	msp	800.
URC002	urp	2.
URP0600	urp	600.
URP8001	urp	8001.
URP8002	urp	8002.
URP8004	urp	8004.

The MSP0609 and MSP0612 controller models require two mpc records, one for each half of the controller.

3. iomi

is the IOM to which each link adapter is connected. (See Notes below.)

4. chani

is the starting logical channel number for each link adapter. (See Notes below.)

5. nchani

is the number of logical channels on each link adapter. (See Notes below.)

Labeled Format

```
mpc -ctlr ctlr_name -model ctlr_model -iom iom1 -chn chn1 -nchan
   nchan1 {... -iom iom4 -chn chn4 -nchan nchan4}
```

Notes

Up to 4 physical channels can be described on the mpc record. These channels must be listed in a specific order. The information for the bootload channel (as set on the MPC maintenance panel) must be listed first. The bootload channel is followed by the primary channel on the non-bootload link adapter (LA). Following these two primary channels are the secondary channels for the LAs, in the same order as the primary channels. If any of these channels are not connected to the mpc they should simply be omitted from the mpc record. This ordering of the channels is important to guarantee that the channels will be used in the most efficient manner possible.

For disk and unit record MPCs, up to eight logical channels may be specified, since up to eight logical channels may be configured per physical channel. For tape MPCs, up to two logical channels may be specified. However, only one logical channel should be specified, since only one logical channel should be configured per physical channel.

Examples

```
mpc mspa 611. a 20. 2 a 24. 2
```

```
mpc -ctlr mspa -model 611. -iom a -chn 20. -nchan 2 -iom a -chn 24.
   -nchan 2
```

Name: parm

This record is used to define software parameters. More than one parm record may be used if many parameters are to be specified, although several parameters may be specified on each record.

Format

```
parm parameters
```

where parameters can be chosen from the following:

astk
: enables online maintenance of the SST name table. This increases system overhead, but speeds up dumps.

parm

ccrf

specifies that the system should crash if all available consoles fail or if the bootload console is forcibly detached. If this parameter is not present, the system will continue to run for as long as possible without a console. I/O for the console will be sent to the syserr log. This parameter should be used by those sites which consider it critical that system events be reported in a timely manner.

chwm

causes information to be printed on the bootload console during bootload pertaining to the collection 1 high-water mark; that is, the number of pages used by early initialization.

crwl

is used only for system debugging. If the parameter crwl is put on the parm record, the system returns to BCE on every attempt to crawl out of ring 0, with the message:

```
verify_lock: crawlout stop specified on parm record
```

so that the system staff can take a dump. Typing go causes the system to continue operation.

dirw

causes modified directory pages to be written from main memory whenever a directory is unlocked. Specifying this parameter increases the safety of the system at some cost in increased paging. Process directory pages are not written. This parameter must be specified in order to keep Data Management-protected files consistent across ESD-less crashes.

dris

specifies that the system should not check an IOM's configuration settings for consistency before adding the IOM to the system. For more information, see "Notes on Adding IOMs" in Section 11.

dskq N

where N specifies the maximum number of disk queue elements that can ever be pending (i.e., the maximum number of disk I/O operations that can be queued at any one time). The system enforces limits so that the number of elements per disk drive is no fewer than 5 * the number of disk drives and no greater than 200 * the number of disk drives. If this parameter is not specified, the default is 20 elements per drive.

hcpt

causes information to be printed on the bootload console during initialization concerning utilization of all defined hardcore partitions. A message such as the following will be printed for each drive with a hardcore partition:

```
accept_fs_disk: HC PART on dska_07 used 500 out of
  1000 records.
```

If this parameter is not specified, the information is recorded in the syserr log.

vtb N

where N sets the number of VTOC buffers. Increasing N may reduce the number of VTOC reads. The default number of buffers is 30.

wlim N

where N specifies the maximum number of outstanding writes which may be permitted for a memory flush operation to proceed. Typically, a memory flush operation, which occurs to safeguard pages, will be 1/2 or less of this number. The default is 1/8 of the configured pageable memory (in pages). When memory is reconfigured the wlim value is recalculated.

In addition to the above, the TTYB segment ID may be supplied on a parm record followed by a segment length in words, to set the length of tty_buf. The default length is 6144. words.

Example

```
parm chwm vtb 40.  ttyb 8192.  ccrf
```

Name: part

Part records inform BCE and Multics of the location of the areas of disk used for various partitions.

part

Format

```
part partname subsystem drive{sv}
```

where:

1. partname
   is the name of the partition residing on a particular volume or subvolume. The system consults the label of that physical volume to determine where the records of the given partition reside. The one partition commonly used is:

   dump
   area of disk used to contain dump image.

2. subsystem
   is the name of the peripheral subsystem.

3. drive{sv}
   is the decimal number of the disk drive and, if the drive is a 3380 or 3381, the name of the subvolume on which the partition is located. This is an alphanumeric field; it does not accept a period (.).

Labeled Format

```
part -part partname -subsys subsystem -drive drive{sv}
```

Notes

The part record tells the system (BCE and Multics) on which volume a partition resides; the location of the partition is found in the label for that volume. Information about which drives contain which partitions can be gathered when using the init_vol and rebuild_disk commands (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) or by using the display_disk_label command (described in the *Multics Commands and Active Functions* manual, Order No. AG92). The BCE, CONF, FILE and LOG partitions always reside on the RPV and thus do not need to be specified on the part record.

Examples

```
part dump dskb  1

part -part dump -subsys dskb -drive 1
```

The above records state that the DUMP partition resides on the volume mounted on drive 1 of disk subsystem DSKB.

```
part dump dske 3b

part -part dump -subsys dske -drive 3b
```

The above records state that the DUMP partition resides on the volume mounted on drive 3 subvolume B of disk subsystem DSKE.

Name: prph

The prph record supplies all necessary data about a peripheral device or subsystem. Since different devices require different amounts of additional data, only one device or subsystem may be described on a prph record.

Format

The format of the prph record for various peripheral devices is shown below. For each record, n can be any single alphabetic or numeric character.

```
prph  ccun  iom  channel  model
prph  dian  iom  channel  model
prph  dskn  iom  channel  nchan  model1 d1 {model2 d2...model5 d5}
prph  fnpn  iom  channel  model  state
prph  opcn  iom  channel  model  line_length state {option}
prph  prtn  iom  channel  model  train  line_length
prph  punn  iom  channel  model
prph  rdrn  iom  channel  model
prph  tapn  iom  channel  nchan model1 d1 {model2 d2...model5 d5}
```

where:

1. iom

   is a letter (a, b, c or d) signifying the IOM through which the device is driven.

2. channel

   is the IOM channel through which the device is driven.

3. model

is a model number for a device.

The valid model numbers for combination record units (ccun) are:

```
401.   CCU0401
```

The model number for a direct channel (dian) may be any number, specified at site discretion. If it isn't 0, the -model control argument may be used in RCP commands (e.g., assign_resource) to specify a class of device. In general, this field will be 0, and the device will be specified by name (e.g., diab).

The valid model numbers for disk drives (dskn) are:

```
400.   MSU0400
402.   MSU0402
451.   MSU0451
500.   MSU0500
501.   MSU0501
3380.  MSU3380
3381.  MSU3381
0      drive does not exist (see "Examples")
```

The valid model numbers for FNPs (fnpn) are:

```
6670.   DN6670
```

The valid model numbers for bootload consoles (opcn) are:

```
6001.  CSU6001
6004.  CSU6004
6601.  CSU6601
```

The valid model numbers for printers (prtn) are:

```
 401.  PRT401
 402.  PRT402
 901.  PRU0901, PRU0903
1000.  PRU1000
1200.  PRU1200
1201.  PRU1201, PRU1203
1600.  PRU1600
```

The valid model numbers for card punches (punn) are:

```
120.  PCU0120
121.  PCU0121
201.  CPZ201
300.  PCU0300
300.  CPZ300
301.  CPZ301
```

The valid model numbers for card readers (rdrn) are:

```
201.  CRZ201
301.  CRZ301
500.  CRU0500
501.  CRU0501
1050. CRU1050
```

The valid model numbers for tape drives (tapn) are:

```
500.  MTU0500
507.  MTU0500 (7 track)
600.  MTU0600
610.  MTU0610
630.  MTU0630
8200. MTU8200
 0    drive does not exist (see "Examples")
```

4. nchan

is the number of logical channels to use. The number of channels assigned to a physical channel cannot exceed 8. This number must be less than or equal to the value configured in the hardware patch located in the IOM.

5. di

is the number of drives of type modeli.

6. state

is the state of an FNP or a console. The valid states for an FNP are:

on
: indicates that the FNP may be used by the system.

off
: indicates that the FNP may not be used by the system. The FNP will not be loaded by the answering service even if the CMF calls for it to be loaded.

The valid states for a console are:

on
: specifies that this console is selected as the bootload console and is the primary recipient of I/O. There must be one and only one console with a state of on. In the event of bootload console failure, the system will change the state of this console to inop.

alt
: specifies that this console is to be used as an alternate in the event of bootload console failure. If the bootload console becomes inoperative, the system searches the configuration file for a console with a state of alt. If one is found, its state is changed to on and it becomes the bootload console. When several consoles are specified as alternates, they are selected in the order in which they appear in the configuration file.

io
: specifies that this console exists, but is not to be used as an alternate console. A console with a state of io may be attached as an I/O device.

inop
: specifies that this console is inoperative. Normally, this state is asigned dynamically during console recovery. A console with a state of inop may be attached as an I/O device.

7. line_length

is the number of characters that can be printed on a line. **WARNING:** although you may specify a line length of greater than 80 characters for any kind of bootload console, you must be careful if you do this for a CSU6601 console. Line lengths of greater than 80 characters may cause buffer overflows on CSU6601 consoles. Repeated overflows will result in the console being marked inop (see the description of "state" below) and removed from service. If you suspect this to be a problem, you may bring the console back online by using the set_system_console command, documented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

8. option

must be set to "mask" for any CSU6601 console which doesn't have C.0 firmware. Failure to specify "mask" for such a console will result in loss of that console.

Before the release of C.0 firmware, passwords read from a CSU6601 console were hidden on the printer, but not on the screen. ("The screen" in this discussion refers to both the VIP terminal and the CONRAC monitor, if one is attached to the console.) With the release of C.0 firmware, passwords read from a CSU6601 console which has the firmware are hidden on both the printer and the screen. Passwords read from a CSU6601 console which doesn't have the firmware, but has "mask" specified on its prph record, are hidden on the printer, but not on the screen. (In other words, password masking is exactly the same as it was before the release of C.0 firmware.) A CSU6601 console which doesn't have the firmware and doesn't have "mask" specified on its prph record is treated as if it does have the firmware. This eventually results in loss of the console.

Note that passwords read from a CSU6601/6004 console are always hidden on the printer and never hidden on the CONRAC minotor (if one is attached to the console), regardless of the state of the option field.

9. train

is the print train image number. The valid numbers for print train image numbers are:

train #	device/train description	marketing ID#	CSD T+D#
600.	PRT401/402 94 character set ASCII belt	PRB600	9
600.	PRU0901/0903/1201/1203 94 character set ASCII belt	PRB3600	9
600.	PRU1000/1200/1600 94 character set ASCII belt	PRB0600	9

Labeled Format

```
prph -device ccun -iom iom -chn channel -model model
prph -device dian -iom iom -chn channel -model model
prph -subsys dskn -iom iom -chn channel -nchan nchan -model model1
   -number d1 {-model model2 -number d2...-model model5 -number d5}
prph -device fnpn -iom iom -chn channel - model model -state state
prph -device opcn -iom iom -chn channel -model model -ll line_length
   -state state {-option option}
prph -device prtn -iom iom -chn channel -model model -train train -ll
   line_length
prph -device punn -iom iom -chn channel -model model
prph -device rdrn -iom iom -chn channel -model model
prph -subsys tapn -iom iom -chn channel -nchan nchan -model model1 -number
   d1 {-model model2 -number d2...-model model5 -number d5}
```

Notes

The prph dia record is used to describe direct channels used for special purposes (i.e., other than for connection to FNPs).

Examples

```
prph  dska  a  30.  4  0  4  500.  4  451.  4
prph  tape  a  22.  2  630.  2
prph  prta  a  15.  1201. 600. 136.
prph  prtb  a  14.  1201. 600. 136.
prph  puna  a  17.  301.
prph  rdra  a  16.  1050.
prph  opca  a  20.  6004.  80. on mask
prph  opcb  a  24.  6601.  80. alt
prph  fnpa  a  28.  6670.  on

prph -subsys dska -iom a -chn 30. -nchan 4 -model 0 -number 4 -model 500.
  -number 4 -model 451. -number 4
prph -subsys tape -iom a -chn 22. -nchan 2 -model 630. -number 2
prph -device prta -iom a -chn 15. -model 1201. -train 600. -ll 136.
prph -device prtb -iom a -chn 14. -model 1201. -train 600. -ll 136.
prph -device puna -iom a -chn 17. -model 301.
prph -device rdra -iom a -chn 16. -model 1050.
prph -device opca -iom a -chn 20. -model 6004. -ll 80. -state on
   -option mask
prph -device opcb -iom a -chn 24. -model 6601. -ll 80. -state alt
prph -device fnpa -iom a -chn 28. -model 6670. -state on
```

Each prph dskn record describes a disk subsystem, and each prph tapn record describes a tape subsystem. Note that a subsystem is defined as any group of drives and channels where any of the drives may be accessed via any of the channels. For disk subsystems this may include several IOMs and MPCs. Disk and tape drive numbers within a subsystem start at device number 1, and are consecutive on the prph dskn and prph tapn records. Non-existent devices can be represented by model number 0, as in the first example above. In this example, subsystem DSKA has MSU0500 drives on units 5 through 8, and MSU0451 drives on units 9 through 12. See the chnl record description for an explanation of how to specify several groups of channels for a disk subsystem. Note that no more than 8 channels may be assigned to one subsystem.

Name: root

The root record specifies the location of the physical volumes of the root logical volume (RLV).

Format

```
root subsystem1 drive1{sv} {...subsystemN driveN{sv}}
```

where:

1. subsystemi
   is the name of the peripheral subsystem on which a physical volume of the RLV is mounted.

2. drivei{sv}
   is the decimal number of the disk drive and, if the drive is a 3380 or a 3381, the name of the subvolume on which that physical volume is located. This is an alphanumeric field; it does not accept a period (.).

Labeled Format

```
root -subsys subsystem1 -drive drive1{sv}
   {... -subsys subsystemN -drive driveN{sv}}
```

Notes

All physical volumes of the RLV should be listed on the root record, regardless of whether or not they contain a hardcore partition, unless you are booting to rebuild one or more of them with rebuild_disk or to reload one or more of them with the volume reloader. In these cases, the volume(s) to be rebuilt or reloaded should be omitted from the record. A volume that isn't listed on the root record can be added in ring 1 with the add_vol command. However, you should note that the system will not make use of a hardcore parition contained in a volume added in ring one. The first subsystem/drive pair specified on the root record must be that of the root physical volume (RPV). During a cold boot, the root record must list only the RPV.

Examples

```
root dska 1 dska 2 dskc 0a dskc 1b

root -subsys dska -drive 1 -subsys dska -drive 2 -subsys dskc
   -drive 0a -subsys dskc -drive 1b
```

In these examples, assume that the RLV is located on 451 drives DSKA 1 and DSKA 2, and on 3380 drives DSKC 0 subvolume A and DSKC 1 subvolume B. Information about which drives contain which partitions can be gathered by using the init_vol and rebuild_disk commands (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) or by using the display_label command (described in the *Multics Commands and Active Functions* manual, Order No. AG92).

Name: salv

This record is used to change the options for all salvaging operations that take place before the system reaches ring 4 command level, and for the automatic online salvager.

Format

salv keys

where keys may be chosen from the following:

rbld
: rebuilds all directories

path
: prints pathname of all directories salvaged

debg
additional error messages will be printed for debugging use only

dcf
enables deletion of branches suffering connection failure

Name: schd

The schd record is used to set the scheduling factors and parameters in the system configuration. All of these can also be set with the change_tuning_parameters command, described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

Format

```
schd wsf tefirst telast timax {mine {maxe {maxmaxe}}}
```

where:

1. wsf
   is the working set factor. It is used as a multiplier to compute the amount of main storage that must be available before a process is made eligible, as a function of that process's working set. The wsf is given in units of 256K -- that is, a value of 1000000 octal (256K decimal) sets the wsf to 1.0, while a value of 400000 octal (128K decimal) sets wsf to 0.5. A wsf value of 0.5 means that a process can be made eligible if there is sufficient main storage available to hold half its working set, as determined by its recent usage of main storage. If the schd record is not used, the default is 1.0 (1000000 octal).

2. tefirst
   defines the amount of CPU time for which a process is guaranteed to remain eligible (if necessary) the first time it runs after an interaction. Units are eighths of a second (octal). If the schd record is not used, the default is 2 seconds (20 octal).

3. telast
   defines the amount of CPU time for which a process is guaranteed to remain eligible when it is in the last scheduling queue. Units are eighths of a second (octal). If the schd record is not used, the default is 2 seconds (20 octal).

4. timax

   defines the default amount of CPU time a process remains in the last scheduling queue before being rescheduled (at the end of the queue). Units are eighths of a second (octal). If the schd record is not used, the default is 8 seconds (100 octal).

5. mine

   is an optional parameter that specifies the minimum number of eligible processes. The default is 2 processes.

6. maxe

   is an optional parameter that specifies the maximum number of eligible processes; this parameter must be greater than or equal to "mine." If this option is specified, "mine" must be specified also. The default is 6 processes.

7. maxmaxe

   is an optional parameter that specifies the maximum that "maxe" can be raised to during this bootload. It must be greater than or equal to "maxe" plus 10. The default is "maxe" plus 10.

Labeled Format

```
schd -wsf wsf -tefirst tefirst -telast telast -timax timax
   {-mine mine {-maxe maxe {-maxmaxe maxmaxe}}}
```

Examples

```
schd 400000. 20. 20. 100. 2 6

schd -wsf 400000. -tefirst 20. -telast 20. -timax 100.
   -mine 2 -maxe 6
```

These examples assume a value of 0.5 for wsf; a timing of 2 seconds each for temin and temax; a value of 8 seconds for timax; and arbitrary values of 2 and 6 for mine and maxe, respectively.

sst

Name: sst

The sst record describes the partitioning of the system segment table (SST) in the system configuration.

Format

```
sst ast1 ast2 ast3 ast4
```

where:

1. ast1
   is the number of active 4K segments allowed. The default value is 400.

2. ast2
   is the number of active 16K segments allowed. The default value is 150.

3. ast3
   is the number of active 64K segments allowed. The default value is 50.

4. ast4
   is the number of active 256K segments allowed. The default value is 10.

Labeled Format

```
sst -4k ast1 -16k ast2 -64k ast3 -256k ast4
```

Examples

```
sst  442.  220.  45.  15.

sst -4k 442. -16k 220. -64k 45. -256k 15.
```

tbls

Name: tbls

The tbls record is used to specify the length of certain paged system tables.

Format

```
tbls name1 length1 {... name4 length4}
```

where:

1. namei

   may be selected from the following:

   str

   specifies the length of the system trailer segment. The size of this segment depends in a complicated way on the number of AST entries and the amount of segment-sharing among processes. Since the number of disk drives is a reasonable (though crude) indication of the number of users a system can support, it can provide a simple method of figuring the size. Using this method, length for STR should be 16 pages for six drives or fewer, and should be increased by two pages for every disk drive after the first six. A value of 64K should be adequate for most sites.

   ioat

   controls the size of the I/O assignment table.

   prds

   controls the size of the processor data segment (there is one such segment per configured CPU). The default value is 7. This is always the correct value, except when there are no communications multiplexers enabled. In that case, a value of 4 should be specified.

   scav

   specifies the length of the scavenger's database, in pages. The default value is 70KW. The size of this segment determines the number of simultaneous volume scavenges that can be run. The default value allows one scavenger at a time to run. An additional 69KW is required for each additional simultaneous scavenger.

2. length

   is the number of pages in the table.

If a tbls record is not included in the config file, default values are used as if the record read:

```
tbls str 16 ioat 4 prds 7 scav 70.
```

tcd

Name: tcd

The tcd record describes the allocation of the databases in the system configuration that contain information needed by the traffic controller.

Format

```
tcd apt itt
```

where:

1. apt
   is the number of entries in the active process table; the apt argument sets the maximum upper bound on number of processes logged in.

2. itt
   is the number of entries in the inter-process transmission table; it must be high enough to handle normal message flow. The number for itt should be about double the number of apt entries.

Labeled Format

```
tcd -apt apt -itt itt
```

Examples

```
tcd  75.  150.

tcd -apt 75. -itt 150.
```

Note

The system crashes if the tcd record is omitted.

Name: udsk

The udsk record specifies the number of disk channels and which devices in a subsystem are available in the system configuration for user I/O.

Format

```
udsk subsystem nchan {drive1 count1 ... drive6 count6}
```

where:

1. subsystem
   is the name of the disk subsystem.

2. nchan
   specifies the maximum number of channels that may be used to support user peripheral disk I/O.

3. drivei
   is the first drive number to use.

4. counti
   is the number of drives to use.

Up to six drive/count pairs may be specified to indicate the particular units available for user peripheral disk I/O.

Labeled Format

```
udsk -subsys subsystem -nchan nchan {drive drive1 -number
   count1 ... -drive drive6 -number count6}
```

Notes

If the channels indicated are not all required for user I/O, they are available for storage system I/O. An individual disk drive can be used for either storage system volumes or user peripheral packs. The status of disk drives can be changed dynamically during operation by the set_drive_usage command. If no udsk record appears for a given subsystem, a default of one channel is assumed.

Examples

```
udsk  dska  2

udsk -subsys dska -nchan 2
```

SECTION 8

SYSTEM STARTUP AND SHUTDOWN

OVERVIEW OF SYSTEM STARTUP

There are several steps to bringing up Multics:

- Configure the system. The pack which contains the RPV must be mounted on a drive.
- Mount the RLV (if not already mounted) and all physical volumes of all * logical volumes required at the site for starting
- Boot BCE from the current BCE/Multics system tape
- Boot Multics from BCE
- Start up the answering service and log in the daemons to perform backup, input/output, and any other specialized procedures (such as network interaction).

Step-by-step procedures for bringing up Multics are available in the *Operator's* * *Guide to Multics*, Order No. GB61. What follows is a detailed description of what happens when you boot BCE/Multics.

Bootloading BCE/Multics

A BCE/Multics bootload is the process of loading the programs that make up the bootload command environment, which in turn build up from themselves the Multics operating system. The bootloading process loads the programs into memory, links them so that they may refer to one another, and sets up any necessary databases.

BCE and Multics are loaded from a BCE/Multics system tape. The programs and data on a system tape are divided into groups called collections. The first program on the tape, imbedded in the tape label, is bootload_tape_label. It reads in the first collection of programs, collection 0. Collection 0 reads in collection 0.5, which contains firmware images for the bootload tape controller. Collection 0 determines the correct tape firmware by asking the operator, and loads that firmware. It then reads in collection 1 and prelinks it. This permits programs written in PL/I to be used. Collection 1 enables paging and starts up BCE. Collection 1 then reads in

collection 1.2, which contains BCE exec_coms and files, and disk firmware images. It loads firmware into the bootload disk controller, then reads in collection 1.5, which contains some of the BCE programs. BCE also reads collections 2 and 3, needed to bootload Multics, into the MST partition of the RPV.

When BCE is finished, collection 2 is run to initialize and set up the Multics storage system and the environment to do reloads and other system startup activities. These programs are found in collection 3. Then the initializer process starts running.

THE INITIALIZER PROCESS

When the Multics bootload sequence is started by the BCE boot command, a process is created which is called the initializer process (Initializer.SysDaemon.z). The initializer process is also referred to as the system control process or the answering service process. This process remains active as long as the system is running. It performs many functions for the system, such as:

1. Answering service operations (e.g., login, logout)
2. Operator command service
3. System reloading
4. System terminal management and message routing
5. System accounting
6. User request handling (rcp, new_proc, etc.)
7. System administration

The initializer process is controlled by the bootload console and one or more initializer (message coordinator) terminals. These terminals are used to input commands that control system operation.

The system cannot operate without an initializer process. If an error occurs that makes the initializer process unusable, the system crashes with the message:

```
Attempt to terminate initializer process.
```

INITIALIZER COMMANDS

Once the initializer process begins running, you may issue initializer commands. Initializer commands control the system. They allow you to do the following:

- Get information about the state of the system
- Intervene manually in automatic system functions

- Start and stop nonautomatic system functions
- Change system operating parameters
- Debug the system
- Manipulate storage system volumes
- Control the message coordinator (including device channels, virtual consoles, message routing, sources, and daemons)
- Reconfigure system resources
- Do hierarchy and volume recovery
- Control FNPs and communications channels
- Control user processes

Complete descriptions of the initializer commands are presented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64. The initializer runs first in the administrative ring (ring 1) and then in the user ring (ring 4).

Administrative Ring Commands

When the initializer process enters the administrative ring (ring 1) environment, the first thing the ring 1 environment types is a message of the form:

```
Multics <sysid> - <date> hhmm.t <zone> <day>
```

giving the system identifier from the system tape and the current date, time, and time zone.

In ring 1, the initializer uses only those programs on the BCE/Multics system tape. It doesn't reference any files in the storage system. Only the RLV is known -- the rest of the logical volumes are not mounted until you either issue commands to mount them or you move to ring 4.

If you specified a ring 1 command as an argument to the boot command, this ring 1 command is executed automatically. For example, typing:

```
boot star
```

executes the star (startup) command in ring 1. If you didn't specify any arguments or if startup fails, then the ring 1 program types:

```
Command:
```

and waits for you to type any initializer command allowed in ring 1.

If you type standard, startup, or multics, the initializer leaves the ring 1 environment and executes subsequent commands in the user ring (ring 4) environment.

User Ring Commands

If the initializer process exited from ring 1 because of a startup or multics command, then when it enters the user ring (ring 4) environment, the first thing the ring 4 environment does is initialize the answering service.

Once the initializer process begins operation in ring 4, you may issue ring 4 initializer commands to affect the operation of the system.

If operator authentication is required at your site, i.e., if the require_operator_login installation parameter is turned on, you must sign on before you can enter commands. If operator authentication is not required at your site, signing on is optional. To sign on, use the sign_on command, which is documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64. The sign_off command is also documented in that manual. If the require_operator_login parameter is turned on, Multics will demand identification and authentication of operators from the successful completion of answering service initialization until shutdown. Multics will *NOT* demand authentication in the bootload command environment, in the ring 1 initializer environment, or in the pre-answering service ring 4 initializer environment.

If your site has a timeout period, i.e., if the operator_inactivity_limit installation parameter is set, you will be automatically signed off if you don't enter any commands for more than the specified length of time.

Both the require_operator_login and the operator_inactivity_limit installation parameters are documented in the *Multics System Administration Procedures* manual, Order No. AK50.

The initializer process is normally waiting for an initializer command from you. After the command is typed in, the initializer performs it, types a ready message, and awaits another command.

If operator authentication is not required at your site, and no one has signed on, the ready message looks like this:

```
Ready
```

If operator authentication is required at your site, and no one has signed on, the ready message looks like this:

```
Ready (Not Signed on.)
```

Whether or not operator authentication is required, once someone has signed on, the ready message looks like this:

```
Ready (User_name)
```

Online help is available for all ring 4 initializer commands via the help initializer command. For information on how to use the help initializer command, refer to its description in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64. (See also "Getting Help with Commands" later in this section.)

Full Multics typing conventions are available in the ring 4 initializer environment, except with the reply and intercom commands. These typing conventions include the use of special characters, such as parentheses (for iteration), double quotes (for quoting), and semicolons (for entering multiple commands on a single line). For information on the use of special characters, refer to the *Multics Programmer's Reference Manual*, Order No. AG91.

ADMIN MODE

The initializer process is sometimes used to perform special operations (such as setting access in the root directory) that a normal process cannot perform. For instance, the initializer is the only process that can execute commands before the answering service is brought up. Thus, it must be used to repair problems that prevent all users from logging in.

In order to use the initializer process to execute an arbitrary Multics command, you must enter admin mode. Because the initializer process has special abilities and special limitations, admin mode should only be used by qualified personnel.

To enter admin mode, type the admin command. You must also supply a password to enter admin mode, unless the system administrator has specified that no password is needed.

Once in admin mode, the initializer responds to regular Multics commands instead of initializer commands. However, a special command in admin mode permits execution of initializer commands. Typing:

```
sc_command hmu
```

causes the initializer command "hmu" to be executed.

Admin mode and editing of the message of the day can be entered from the bootload console or from any initializer terminal, but only one console or terminal can be operating in this mode at a time.

Use of admin mode is similar to the use of regular Multics. On an initializer terminal, issuing a quit signal (hitting the BRK key) has the same effect in admin mode as it does in a normal Multics process. (The initializer goes to level 2, the command is suspended, and you remain in admin mode.) This is not true on the bootload console, which doesn't have a BRK key, or any other way of issuing a quit signal.

To exit from admin mode, type the admin_mode_exit (ame) command.

send_admin_command COMMAND

Highly privileged users may send single commands to the initializer to be executed in admin mode by using the send_admin_command command, described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64. If a highly privileged user sends such a command, a message is printed on the initializer terminal, the command is executed, and the system continues in normal operating mode.

If you're in the user command environment, and you want to enter an initalizer command, use the sac command and the sc_command command in combination with the initializer command. For example:

```
sac sc_command maxu auto
```

If you're in the user command environment, and you want to enter an exec command, use the sac command and the ec admin command in combination with the exec command. For example:

```
sac ec admin attended
```

If you're in the user command environment, and you want to make the initializer process execute a Multics command, use the sac command in combination with the Multics command. For example:

```
sac set_system_console -reset
```

If you're in the user command environment, and you want to enter a daemon command, use the sac command and the sc_command command in combination with the initializer reply command, the daemon driver label, and the daemon command. For example:

```
sac sc_command reply prta go
```

Note that you may also enter a daemon command from the user command environment by using the send_daemon_command command (described later in this section).

The send_admin_command command has a number of useful control arguments. One is the -notify_by_mail (-ntmail) control argument. If you use -ntmail, the system will send you mail when your command completes execution. The mail will contain a message stating that your command ran. It will also contain all of the output produced by your command. For example:

```
sac -ntmail word
```

Another useful control argument is -query (qy). If you use -qy, the system will print your command line on your terminal and ask you if you want to send it. This is useful for validating the effects of abbrevs and active functions. For example:

```
sac -qy delete [wd]>test_file

send_admin_command:  delete >udd>m>Smith>test_dir>test_file
send_admin_command:  Do you want to send this admin command line
   to the initializer <wait info>?
```

If the command you want to execute asks a question, you must include the answer request in your command line. For example:

```
sac answer yes delete_dir >udd>m>Smith>test_dir
```

If you don't include the answer request, your command will not be executed.

The following abbrev will ask you if your sac command line is correct, then send it, and print the results from the admin log:

```
.ab sac do "send_admin_command -bfqy &rf2;
   print_sys_log -admin -nhe -dfmt """""" -nfmt """""" -fm
      &1" [calendar_clock]
```

Its use is recommended.

GETTING HELP WITH COMMANDS

There are info segments for the following commands in the following directories:

Commands	Directories
user	>doc>info
privileged user	>doc>privileged
ring 4 initializer	>doc>ss>operator
ring 1 initializer	>doc>ss>r1_initializer
accounting	>doc>ss>accounting
I/O daemon	>doc>ss>io_daemon
BCE	>doc>ss>bce

How you look at these info segments depends on what command environment you're in.*

If you're in the user command environment, you can look at info segments for user commands by using the user help command. For example:

```
help list
```

You can look at info segments for all other commands by using the user help command and specifying the absolute pathname of the info segment. For example:

```
help >doc>privileged>set_system_console
```

Or, you can add one or more directories to your info search list, by adding a line to your start_up.ec. For example:

```
asp info >doc>privileged >doc>ss>rl_initializer
```

Then you can look at info segments in the directories you added to your info search list by using the user help command in the regular way. For example:

```
help set_system_console
```

If you're in the ring 4 initializer command environment, you can look at info segments for ring 4 initializer commands by using the initializer help command. For example:

```
help down
```

If the info segment you want to look at is for an exec command, specify its name in the format "x.command". For example:

```
help x.attended
```

* If you're in the ring 1 initializer, restricted accounting, I/O daemon, or BCE command environments, you can't look at info segments at all. Thus, the only commands you can get help with in the same environment in which you can run them are user, privileged user, and ring 4 initializer commands. For a more detailed discussion of command environments, refer to the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

INITIALIZER USE OF COMMUNICATIONS CHANNELS

When the system is bootloaded, the answering service first performs input/output on the bootload console. When ring 4 is entered and the message coordinator is started, initializer input and output are usually moved to a message coordinator terminal connected to a communications channel.

The initializer process is in charge of all communications channels known to the system. Some of these channels are connected to the answering service and used for logins, some are used by the initializer itself for the message coordinator, and some are given to other processes to use.

The disposition of communications channels is controlled by the CDT. A communications channel connected to Multics may be:

1. Completely unused: not in the CDT

2. Inactive: in the CDT but marked INACTIVE (not to be used unless explicitly enabled by the operator)

3. Active: in the CDT and in use for

   a. message coordinator service

   b. login service

      1) Being listened to by the answering service
      2) Attached to a user process

   c. slave service: attachable by some existing user process, but not available for logins

   d. dial_out (autocall)

Initializer commands are available to move a terminal to and from the above states in various combinations.

MESSAGE COORDINATOR

The message coordinator facility allows the initializer to run multiple communications channels and lets the system daemons run without terminals of their own, sending their messages to the initializer for disposition. The daemons can be logged in automatically (by system_start_up.ec or admin.ec) or at operator request (by means of the login command).

To allow for other activity to occur, such as typing in operator messages, the message coordinator writes messages in a burst followed by a pause. When both the burst size and the delay time are set at the default values, it means that if you want to enter a command, after pressing the carriage-return (or the INPUT-REQUEST) button, at most 20 messages will print before you can enter the command. The set_mc_message_limits command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) allows the system administrator to determine the size of the burst and the duration of the pause.

RCP messages and other syserr traffic are not handled by the message coordinator. This means that message coordinator commands can not be used to manipulate these messages. For example, the reroute command can not be used to reroute messages from the bootload console to an initializer terminal.

Occasionally, the message coordinator may stop operating due to a system problem such as a hung up device. In such a case, you can issue the reset initializer command to restart all channels. Channel restart is also attempted if system control encounters any fault.

Input Delivery and Output Routing

There are two parts to the message coordinator -- delivery of input and routing of output. Input delivery is done by the reply command: any terminal owned by the message coordinator may (subject to permission) issue reply commands directed at any source.

Output routing is more complex. Each daemon process running over the message coordinator attaches one or more switches to a source. In addition, the initializer attaches switches to the source "as". When a process writes information to a switch attached to a source, a message is sent to the message coordinator containing the information, the name of the switch, and the name of the source. The message coordinator looks up the combination of switch name and source name in the message routing table (MRT), where it finds one or more routings for each combination. Each routing specifies a virtual console. The message coordinator looks up each virtual console in the virtual console table (vcons_tab), where is finds a list of one or more destinations for each virtual console. The list of destinations may include terminal channels, logs, and sinks. The message coordinator routes the message to each destination in the list for each virtual console specified by each routing. If the destination is a terminal channel, the message is queued for printing on the terminal. If the terminal is unavailable (e.g., has dropped offline), the message waits in the queue. Otherwise, it is printed immediately. If the destination is a log, the message is written directly into the log. (See the description of system logs in Section 13.) If the destination is a sink, the message is discarded. Figure 8-1 illustrates the message coordinator output routing process.

DEFINING OUTPUT ROUTING

To see how message coordinator output routing is defined in the standard system_start_up.ec, refer to Appendix F. In general, the way to define output routing is as follows:

1. Think about your message coordinator output as a number of groups, each consisting of all of the output you want to handle together and route to the same destination.

2. For each group of output you want to route to the same terminal, use the define command to define one virtual console whose destination is that terminal. Then use the route command to route all of the switches on which that output is written to that virtual console.

   Note: you may want to define a second virtual console whose destination is the same terminal, and route the switches on which some portion of the output is written to the second virtual console. This makes it easy to reroute that portion of the output to a different terminal in the future if you so desire, since the only thing you have to change is the destination of the second virtual console.

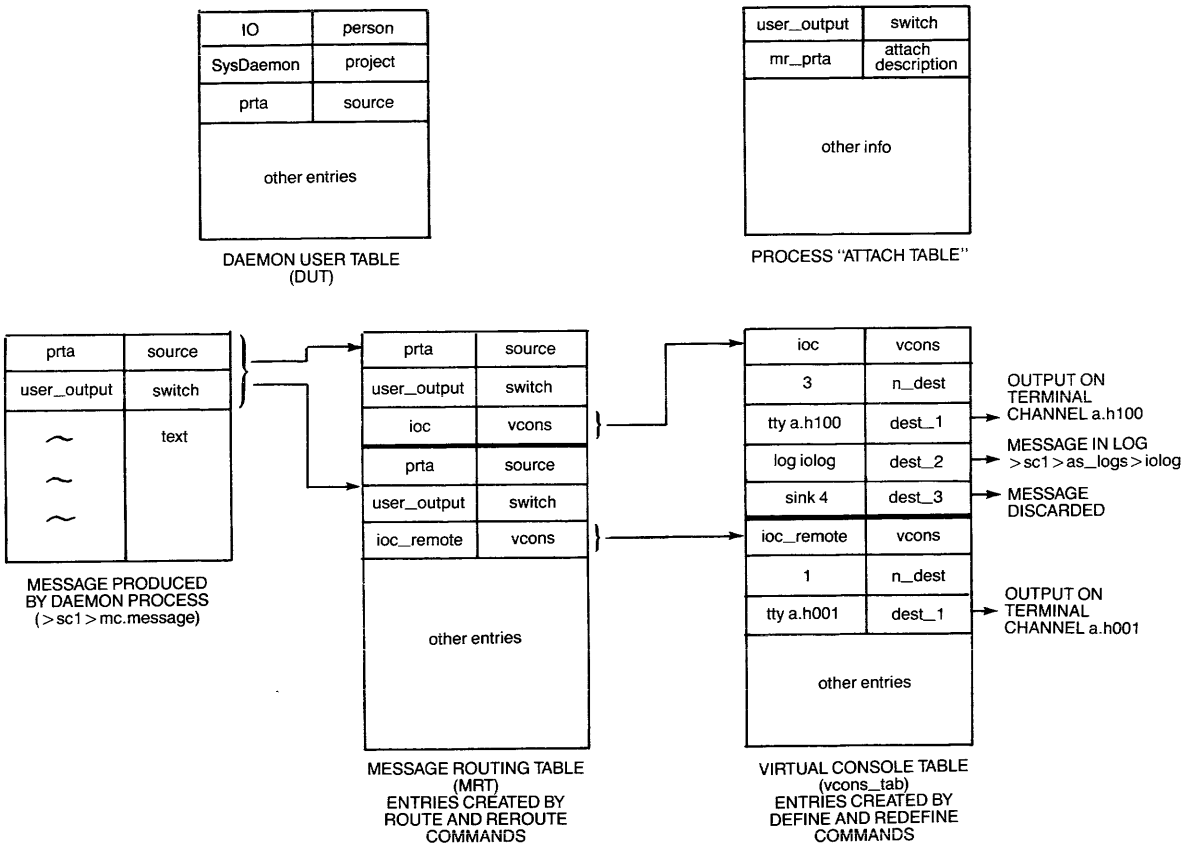


Figure 8-1. Message Coordinator Output Routing

3. For each group of output you want to route to the same log, use the define command to define one virtual console whose destination is that log. Then use the route command to route all of the switches on which that output is written to that virtual console.

   Note: NEVER define two virtual consoles whose destination is the same log.

Figure 8-2 illustrates a typical message coordinator output routing definition.

Operating Daemon Processes

Input and output of the daemon processes are passed through the initializer on their way to and from the terminal channel. To cause a daemon to be logged in from the initializer, type:

```
! login Person_id.Project_id source_id {login control_args}
```

The daemon logs in and attaches its input and output to the message coordinator as a source with name source_id. For example, an I/O daemon can be logged in by typing:

```
! login IO.SysDaemon io2
  Ready (User_name)
  1721 as LOGIN  IO.SysDaemon dmn io2 (create)
```

The Dumper is logged in so that a complete dump may be started by typing:

```
! login Dumper.SysDaemon cdl
  Ready (User_name)
  1721 as LOGIN  Dumper.SysDaemon dmn cdl (create)
```

To log out a daemon, issue the logout command from the initializer, giving the Person_id, Project_id, and source_id of the daemon. Thus, to log out the dumper, the sequence is:

```
! logout Dumper.SysDaemon cdl
  Ready (User_name)
  2231 as LOGOUT Dumper.SysDaemon dmn cdl 12:11 $23.45 (logout)
```

To log out all daemon processes when the system is being shut down, type:

```
! logout * * *
```

Occasionally, you will need to send a quit to a daemon process. A special command is required because the ATTN or INTERRUPT button on an initializer terminal is connected to the initializer, not to the daemon, and is ignored by system control. To send a quit to a daemon, type:

```
! quit source_id
```

The daemon will accept the quit command after it has been passed from the initializer.

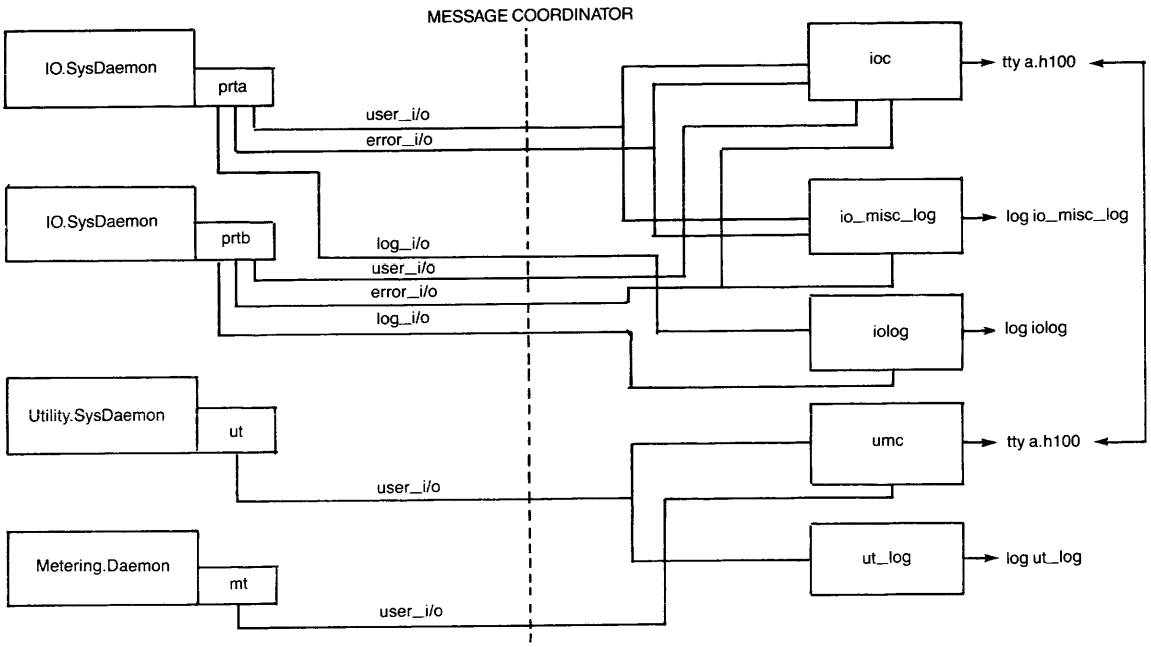


Figure 8-2. Typical Output Routing Definition

The example below shows how the system intermixes output lines from various sources on a single console, and how the user replies to a request for input from a source.

```
! reply cord coordinator
  Ready (User_name)
  1953 cord IO coordinator initialized
! reply prtb driver
  Ready (User_name)
  1953 prtb Enter device name and optional request type:
  --> prtb
! reply prtb prtb printer
  Ready (User_name)
  1954 cord New driver for device prtb request type printer
     (series =10000)
  1954 prtb prtb driver ready at MM/DD/YY 1954.1 EST DAY
  --> prtb
! reply prtb go
  Ready (User_name)
! reply bk start_dump sys_dirs xyz 1 60
  Ready (User_name)
  1955 bk Begin at MM/DD/YY 1955.6 EST DAY
  1955 bk >user_dir_dir
  1955 bk Type primary dump tape label
  --> bk
! reply bk IC-75
  Ready (User_name)
```

The above set of replies are written out only to serve as an example. In practice, the commands and replies necessary to start the daemon functions are contained in the segment admin.ec, and can be invoked by using the exec (x) command. The admin.ec segment is usually modified by each site to suit its requirements. Using the admin.ec that is distributed with the system, the above functions -- starting the I/O daemon coordinator, the I/O daemon driver for prtb, and the incremental backup daemon -- can be accomplished using the following commands:

```
! exec io
  Ready (User_name)
! exec inc xyz IC-75
  Ready (User_name)
```

(xyz stands for the user's initials or name.) Complete descriptions of the exec commands are presented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

send_daemon_command COMMAND

Users with the appropriate access may control the operation of system daemon processes by using the send_daemon_command command, described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64. This command allows you to log daemons in and out over specified message coordinator source ids, send command lines to daemons, and signal QUITS in daemon processes.

Use of the send_daemon_command command is somewhat different from use of the send_admin_command command (described earlier in this section). Access to send_admin_command gives a user complete control of the initializer process, as well as all daemons controlled by the message coordinator. Access to send_daemon_command may be restricted to give a user access to a single daemon process without giving her control of other daemon processes or the initializer process.

The ability to restrict access to the send_daemon_command command is controlled by the validate_daemon_command installation parameter. If this parameter is enabled, you can use the message coordinator access control segments (MCACS) to restrict use of the send_daemon_command command; i.e., you can use them to restrict which daemon(s) a user may control with this command.

If the validate_daemon_command parameter is not enabled, it is impossible to restrict use of the send_daemon_command command by users with "rw" access to the >sc1>admin_acs>send_daemon_command.acs ACS segment. Users with such access may use the command to control any daemon. For more information about the validate_daemon_command installation parameter and MCACS segments, refer to the *Multics System Administration Procedures* manual, Order No. AK50.

If you have "c" access, you may use send_daemon_command to log a daemon in and out. For example:

```
send_daemon_command login rp Repair.SysDaemon -auth system_high
```

or:

```
send_daemon_command logout rp Repair.SysDaemon
```

If you have "r" access, you may use send_daemon_command to send a command to a daemon. For example:

```
send_daemon_command reply prta go
```

If you have "q" access, you may use send_daemon_command to send a QUIT signal to a daemon. For example:

```
send_daemon_command quit prta
```

When a privileged user sends a command to a daemon via the send_daemon_command command, a message is printed on the initializer terminal, the command is executed, and the system continues normal operation.

MESSAGE COORDINATOR DATABASES

The main databases used by the message coordinator are:

mc_anstbl
: one entry per terminal channel

MRT
: message routing table

vcons_tab
: virtual console table

mc.message
: incoming messages for message coordinator

<source>.message
: input messages for other sources

<channel>.queue
: queued output messages for devices

These tables are all completely reconstructed each time the message coordinator is started. All of these segments are kept in >system_control_1. Their ring brackets should be 4,4,4 and access should be rw for the initializer and daemon processes, and null for everybody else.

STARTUP COMMANDS

The system executes an exec_com called system_start_up.ec when the answering service is started. It executes this exec_com in three parts: some commands are executed before the answering service is started, some are executed after the answering service is ready but before communications channels are listened to, and some are executed after communications channels are listened to. The startup command causes all three parts to be executed, along with complete initialization of the answering service. The multics command causes initialization to stop before part 2 of system_start_up.ec has been executed; the go command executes part 2 of system_start_up.ec, listens to communications channels, and executes part 3 of system_start_up.ec.

Normally, the system_start_up.ec performs some message coordinator initialization before starting the answering service, and logs in the daemons after the answering service is ready. If the initializer is to operate more than one message coordinator terminal, the additional channels are accepted (by commands in system_start_up.ec) at this time.

An example of the system-supplied system_start_up.ec is found in Appendix F.

UNATTENDED AND AUTOMATIC MODES

The system may be operated in several modes: attended or unattended, manual or automatic. In unattended mode, the system assumes that no tape mount requests can be honored, and that the backup and I/O daemons are unavailable.

In automatic mode, the system automatically performs dumping, and it reboots after crashes. Manual mode, indicated by invoking the boot command, allows manual and automatic dumping options, but requires positive user instructions to reboot.

Setting Automatic Mode

In order to operate the system in automatic mode, invoke the auto exec_com as part of the bootload process, instead of typing boot. If the auto exec_com has been used, you have the option of turning off automatic mode with the command:

```
x auto off
```

and may then reenable it with the command:

```
x auto on
```

If the auto exec_com has not been used to boot the system, neither of these commands has any effect.

Setting Unattended Mode

A system function is provided in admin.ec so you can conveniently set the system into unattended mode. Typing:

```
x unattend
```

invokes the following steps:

```
sc_command reconfigure delete device tape_(01 02 03 04 05 06 07 08)
sc_command word login Unattended service
set_flagbox unattended true
set_flagbox auto true
set_flagbox rebooted false
```

Installations with more or fewer than eight tape drives must modify the text of admin.ec. These steps leave backup and I/O daemon functions running; some sites may wish to modify the text of admin.ec to log some daemon processes out.

Returning to Attended Mode

When the system has been placed in unattended mode, you may revert to attended operation by typing:

```
x attend
```

which performs the following steps:

```
sc_command reconfigure add device tape_(01 02 03 04 05 06 07 08)
sc_command word login
set_flagbox unattended false
```

Other operations such as logging in daemons may be added.

SYSTEM SHUTDOWN

System shutdown is the process of ceasing Multics service. There are several steps involved in an orderly shutdown, so that the different classes of users can be logged out in the correct order. The step-by-step procedure for shutting down the system is available in the *Operator's Guide to Multics*, Order No. GB61.

Shutdown Failure

If shutdown fails (does not type the "shutdown complete" message) when the shutdown command is issued at the initializer terminal or the bootload console, you should enter BCE. If the bootload console keyboard does not unlock, you must enter BCE as described under "Returning to BCE" in Section 10. Then you should attempt standard recovery procedures. You should always attempt emergency shutdown. See Section 10 for details on recovering the system.

SECTION 9

THE MULTICS BACKUP SYSTEMS

The Multics backup systems augment the reliability of the online disk storage system. They ensure that user segments and directories can be recovered from tape if they are destroyed due to system failure or user error.

The backup systems perform the following functions:

1. dumping

   The dumping mechanism searches out, selects, and copies (dumps) onto tape segments and directories from the Multics storage hierarchy. The frequency of dumping and the length of time for which tapes are kept are determined at individual sites.

2. retrieval

   Retrieval is the recovery of individual segments and directories. It occurs during normal Multics operation.

3. reloading

   Reloading is the reconstruction of a major portion of the hierarchy when it has been damaged.

There are two major Multics backup systems, hierarchy and volume. The hierarchy system tree-walks the hierarchy to locate the data it must dump, while the volume system scans the physical volumes used by the storage system. The goals and general structure of both systems are the same, but the mechanism, cost, and benefits differ.

The volume backup system is organized around the concept of physical volumes. The volume dumper dumps segments and directories, as specified by physical volume and VTOC index. In contrast, the hierarchy backup system is organized around the storage system directory hierarchy. The hierarchy dumper dumps segments and directories as specified by pathname. Similarly, the volume reloader recovers a single physical volume, while the hierarchy reloader recovers some portion of the directory hierarchy. Both retrievers, volume and hierarchy, recover segments and directories. The dump volumes produced by the volume dumper can not be read by the hierarchy reloader or retriever and vice versa.

In the discussions below an overview of dumping and recovery is presented, followed by details of the operation of the volume and hierarchy systems. Descriptions of the commands used by both backup systems appear in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

DUMPING

The dumping mechanism operates in three modes -- incremental, consolidated, and complete. These modes are distinguished by three criteria used to select segments and directories for dumping. What is dumped is site-controllable. Usually, only information that has changed and will be valuable in the future is dumped. Thus per-process information, static libraries, and re-creatable segments and directories are not dumped. All other sections of the hierarchy should be included in the search route of the backup system.

Multiple dumper processes, registered as SysDaemon, Daemon, or both, are allowed.

The backup system does not guarantee that segments are dumped in a consistent state. For example, it is possible that while the incremental dumper is dumping a segment, another process might be writing into that same segment. Thus, an inconsistent copy of a segment might be produced. However, the modifications that cause a segment to be inconsistent also cause another dump of the segment to be produced on the next pass of the incremental dumper. Therefore, unless the system crashes before the next incremental dump, a consistent copy is eventually produced.

The high production rate of incremental and consolidated dump tapes makes the retention of these tapes for long periods of time impractical. Therefore, incremental and consolidated tapes are kept for some short time, perhaps 3 weeks. Complete dump tapes are retained for a longer time, perhaps 6 months, with the exception of one complete dump tape per month that might be held for a period of 1 year.

Incremental Dumps

Incremental dumping is the principal technique used to keep the backup systems abreast of changes to online storage. It is the purpose of an incremental dump to discover modifications to online information not reflected in backup tape storage. The incremental dump locates and dumps all segments and directories modified more recently than they have been dumped. The net effect of the incremental dumping scheme is to limit the amount of information that can be lost to those modifications that have occurred since the last incremental dump.

Incremental dumping is triggered periodically by software timers. In order to minimize the time span during which modifications to online storage can go unnoticed by the backup system, incremental dumps should be produced frequently. On the other hand, because backup competes with ordinary users and exerts a considerable drain on system resources, it becomes economically desirable to lower the frequency of incremental dumps. Therefore, the time interval between the incremental dump cycles at an installation is chosen as a compromise between these two considerations. This does not imply that an incremental dump necessarily finishes its search within a single time interval. In fact, if the incremental dumper is given no scheduling advantage, several intervals might be required to complete an incremental dump during hours of heavy system load. If an incremental dump is not completed before the next incremental dump is scheduled to begin, the "next" dump is deferred until the prior incremental dump is completed.

Consolidated Dumps

A consolidated dump locates and dumps segments and directories that have already been dumped by an incremental dump cycle. Since a consolidated dump catches modifications accrued over a period of time encompassing many incremental dumps, it effectively consolidates the most recent information from a group of incremental tapes and thereby facilitates the reloading of this information by decreasing the number of tapes that must be processed. Also, since tape is susceptible to operational, hardware, and software errors, a consolidated dump provides the installation with a second tape copy of the segments and directories dumped during an incremental dump.

Complete Dumps

A complete dump dumps every segment and directory in the storage system without regard for modification time. Unlike incremental and consolidated dumps, which attempt to keep the backup tapes up-to-date with the contents of the storage system, complete dumps are somewhat different in purpose.

A complete dump establishes a checkpoint in time, essentially a snapshot of the entire Multics storage hierarchy. If it should ever become necessary to recover a major portion or the entire contents of online storage, then the tape with the most recent complete dump marks a cutoff point beyond which no older dump tapes need be inspected.

RETRIEVAL

A user who notices that a segment or directory has been lost or damaged can submit a request for the retrieval of that segment or directory. The problem the user faces is determining which dump operation produced the dump of the segment or directory to be retrieved. Usually the most recently produced copy is wanted. In the case of a damaged segment, however, the damaged version is likely to have been dumped as well, and hence the most recent dump may not be wanted. It is to be hoped that a user knows approximately when a segment was lost or damaged, and whether the segment has been recently modified. Using these two pieces of information, the user can make a reasonable guess as to when the last usable copy of the segment was online.

Once an estimate has been made as to the time frame, this estimate can be verified by examining the corresponding hierarchy dump map. This operation is automatic for volume retrieval, although the user can still specify the time frame if desired. The hierarchy dump map indicates the tape reel on which the dump was written. A feature of the dump map that is sometimes helpful is the printing of the date and time modified attribute for the segment, which effectively points to the next most recent dump of the segment.

The user can specify that a single segment, a directory without its subtree, or a directory including its subtree be recovered.

Using cross retrieval, a user can specify that a segment or, for hierarchy retrieval only, a directory be retrieved with a different pathname. A single segment can be cross retrieved by the volume retriever to any point in the storage system hierarchy. For hierarchy retrieval only, a directory subtree can be cross retrieved to any point in the hierarchy.

VOLUME BACKUP

* The volume backup system is designed to allow recovery from most disk failures while the system is available for users. Incremental and consolidated volume dumping are significantly faster, though less flexible, than their hierarchy counterparts. It is not intended for general users; nor is it designed to be used for archival storage, or intersite file transmission. It is assumed that the hierarchy backup system will be used for these activities.

The personids "Volume_Dumper", "Volume_Retriever", and "Volume_Reloader" must be registered. These personids should be registered on the Daemon project with
** the multip and daemon attributes. For sites using AIM, the authorization for these personids must also be set at system_high and the home directories must be pre-created at system_high.

To set up the volume backup system, log in the Repair Sysdaemon, or, if running in special session using the initializer, execute the following command:

```
ec >tools>setup_volume_reloader
```

This exec_com will create all directories, segments and message segments necessary for running the volume backup system. This exec_com will also set suggested access on the directories and segments created. Not all the access that is set is required. If a site wishes, the access created for *.SysMaint.* and *.SysAdmin.* may be removed.

A site will need a sufficient number of tapes to accommodate the entire file system and any incremental and consolidated dumps until a subsequent complete dump is taken. It is suggested that a new site start with 100 - 300 reels of tape for volume backup.

The Volume Backup LSS

The commands for the volume backup system are available under a Limited Service Subsystem (LSS). Within this LSS, there is one LSS command table per volume backup daemon. These command tables restrict the volume backup command set as follows.

The LSS command table for the volume dumper (Volume_Dumper.Daemon) restricts its available command set to:

```
complete_volume_dump
consolidated_volume_dump
delete_volume_log
display_pvolog
display_volume_log
dmpr_unlock_pv
end_volume_dump
incremental_volume_dump
merge_volume_log
preattach_dump_volumes
purge_volume_log
rebuild_pvolog
recover_volume_log
set_volume_log
set_volume_wakeup_interval
verify_dump_volume
volume_cross_check
volume_dump_trace_off
volume_dump_trace_on
wakeup_volume_dump
```

The LSS command table for the volume retriever (Volume_Retriever.Daemon) restricts its available command set to:

```
list_retrieval_requests
retrieve_from_volume
```

The LSS command table for the volume reloader (Volume_Reloader.Daemon) restricts its available command set to:

```
display_volume_log
merge_volume_log
recover_volume_log
reload_volume
verify_dump_volume
```

In addition, all three LSS command tables will allow these commands:

```
exec_com
help
home_dir
logout
system
user
```

All the commands listed above are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64, with the exception of list_retrieval_requests, exec_com, help, home_dir, logout, system, and user, which are documented in the *Multics Commands and Active Functions* manual, Order No. AG92. (Note that "help" and "logout" here refer to the standard Multics commands by those names, not the initializer commands by those names.)

If your site runs the volume backup system under the supplied LSS, the project_start_up.ec for the Daemon project (daemon_project_start_up.ec) selects the appropriate command table for each volume backup Person_id based on the result of the [user name] active function.

Also, if your site runs the volume backup system under the supplied LSS, you must be careful about making changes to admin.ec. If you change admin.ec so that commands not contained in the command tables are sent to the volume backup daemons, your operators will not be able to use the "x" command(s) affected by the change.

You may extend the command set available to any one of the volume backup daemons by modifying the appropriate command table and creating the new command set via the make_commands command (documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64).

You may allow the full Multics command set to be available to the volume backup daemons by not executing the enter_lss command in daemon_project_start_up.ec. This means that the volume backup system will not run as an LSS.

Volume Dumping

The volume dumping subsystem produces dumps, usually on magnetic tape, that are used by the volume reloader and retrieval subsystems.

The volume dumper can operate in any user ring and at any AIM level with no loss in efficiency or function. The volume dumper requires re access to the hc_backup_ and rcp_sys_ gates. It does not require access to the phcs_ and hphcs_ gates.

The volume dumper can operate in either a single or multiprocess mode. Thus, volume dumping can be partitioned among many processes for either performance or security reasons. Partitioning is done by specifying different physical volumes in different dump control segments, and can be effected at either the logical or the physical volume level. If done, the partitioning must be the same for both incremental and consolidated dump operations. If this is not done, certain record keeping functions will fail. The dump control segment should specify at least all public volumes and may specify any registered volume. If a volume is not mounted, a message to that effect is printed, and the physical volume is skipped.

The structure of the dumper's output is determined by the I/O module used to create it. In the default case, it is a magnetic tape written in Multics standard tape format. The order of data on a dump volume is as follows:

1. dump information data record.

2. contents segment record containing the contents segment of the previous dump volume in the same dump mode.

3. volume log segment record containing the volume log of the physical volume being dumped prior to this dump pass.

4. dump record consisting of the VTOCE of the object and the object if the object is nonnull.

5. repeated instances of item 4 as required.

6. volume log segment record containing the volume log of the physical volume being dumped after the dump pass completed.

7. repeated instances of items 3–6 for each physical volume dumped.

Items 1 and 2 always appear at the beginning of each dump volume. Items 3 through 6 define the dump of a physical volume and may span multiple dump volumes. If any of the individual records specified as items 3 through 6 cannot fit on a dump volume, the record is rewritten in its entirety on the next dump volume.

As the dumper runs, it records information about its operation for use by the volume reloader and volume retriever subsystems. This information consists of the segments discussed below.

VOLUME DUMPER ACCOUNT SEGMENT

A site may choose to charge for volume dumper services. An accounting segment is created in the directory:

```
>system_control_1>volume_backup_accounts
```

for each dump volume written, with the name dump_volume.account. If this action would overwrite an existing segment, the older copy is renamed to dump_volume.account.1, and so on. Entries in the account segment consist of the unique ID pathname of the segment or directory dumped and the number of records.

VOLUME DUMPER CONTENTS SEGMENT

The contents segment contains a unique identifier for each segment and directory written on a dump volume. A contents segment is created in the directory:

```
>daemon_dir_dir>volume_backup>contents
```

for each dump volume used. It is written on the next dump volume (of the same dump mode) and can then be deleted if necessary. The segment name is of the form dump_volume_name.contents. The contents segment is used by the volume retriever to bypass searching an entire dump volume when a specific segment or directory is needed. Use of the contents segment is explained more fully in the discussion of volume retrieval below.

VOLUME DUMPER CONTENT NAMES SEGMENT

The content names segment is a multisegment file that contains the name space (the total set of names) of each directory that is dumped. The content names segment is created in the directory:

```
>daemon_dir_dir>volume_backup>contents
```

with the name dump_volume_name.content_names, if the -names control argument is given with the incremental_volume_dump, consolidated_volume_dump, or complete_volume_dump commands. The content names segment is used by the volume retriever to bypass recovering a directory from a dump volume during branch retrieval. Use of the content names segment is explained more fully in the discussion of volume retrieval below.

VOLUME DUMPER CURRENT DUMP WORKING SEGMENT

The current dump working segment segment is used by the volume dumper to maintain a memory of what has been done and what is left to do. It is created in the working directory, with the first invocation of the incremental_volume_dump, the consolidated_volume_dump, or the complete_volume_dump command. The segment name is of the form dump_control_file_name.dump_type.control.

VOLUME DUMPER DUMP CONTROL FILE

Volumes are specified via a dump control file as either physical volumes or logical volumes. The format of this segment is:

```
lv,<logical volume name>
  or
pv,<physical volume name>
```

Each line must specify no more than one name and may not have any blanks. Logical volume names are translated into a list of physical volume names. Physical volumes are dumped in the order that they appear in the control list.

What is dumped is controlled by the volume specifications in the dump control file, and whether the owner of the segment has enabled or disabled the incremental and complete volume dumping switches. If both switches are off, the segment is not dumped and cannot be recovered. For more information, see the descriptions of the volume_dump_switch_off and volume_dump_switch_on commands in the *Multics Commands and Active Functions* manual, Order No. AG92.

VOLUME DUMPER PHYSICAL VOLUME LOG SEGMENT

In order to determine when a dump volume no longer contains useful information, a physical volume log segment is maintained for each dump volume. The physical volume log segment contains a record of all physical volumes that have information on the dump volume. It is created in the directory:

```
>daemon_dir_dir>volume_backup>pvolog
```

VOLUME DUMPER VOLUME LOG SEGMENT

As the volume dumper operates on a specified physical volume, it records information about its dumping in the volume log segment. There is one valid volume log for each physical volume that has been dumped. The volume log contains a record of every dump volume that contains information that was dumped from this physical volume. Both the volume reloader and the volume retriever subsystems use the volume log segment to determine which dump volumes created by the volume dumper should be used as input. It is created in the directory:

```
>daemon_dir_dir>volume_backup
```

for each physical volume dumped; the name of this segment is of the form physical_volume_name.volog.

The set of dump volumes necessary to logically reconstruct the physical volume is referred to as a reload group. Normally, a volume log contains two reload groups. The display_volume_log and set_volume_log commands can be used to display the contents of a volume and to set the number of reload groups it contains, respectively. The purge_volume_log command can be used to clean up a volume log. Should a volume log be lost it can be recovered using the recover_volume_log command. Should two volume logs exist for the same physical volume they can be merged via the merge_volume_log command.

AUTOMATIC TAPE MANAGEMENT

The volume dumper can manage its own tapes (uses the -auto control argument) if a tape pool is established. To set up and manage a tape pool (also known as the volume pool segment), use the create command to create a segment in the Dumper default directory >ddd>volume_backup called Volume_Dumper.volumes. Use the manage_volume_pool command (fully described in the *Multics Commands and Active Functions* manual, Order No. AG92) to add tape volumes to the volume pool for the dumper's use. Volume names must be of the form:

```
AAnnnnn or Annnnn
```

where A = any alphabetic character and nnnnn is an integer that will fit into 18 bits.

These volumes are allocated and freed as required by the Volume Dumper when the -auto control argument is used. The number of volumes required is dependent upon the frequency of dumping, the size of the storage system in use, and the number of reload groups (see "Volume Dumper Volume Log Segment" above). The use of the volume_cross_check and purge_volume_log commands is recommended to ensure consistency of the databases and to free volumes when inconsistencies have prevented them from being freed earlier.

DUMP MODES

The volume dumping subsystem operates in one of three ways:

1. incremental volume dump -- for each physical volume specified, those segments and directories that are stored on it are dumped if they have been modified since the last incremental volume dump

2. consolidated volume dump -- for each physical volume specified, those segments and directories that have been incrementally dumped since the last consolidated volume dump are dumped

3. complete volume dump -- for each physical volume specified, all segments and directories are dumped

Incremental Mode

The incremental volume dumper operates cyclically using a default time interval of one hour unless otherwise specified. That is, the dumper process is awakened at one-hour intervals. It incrementally dumps all physical volumes specified in the dump control segment, and then goes blocked to wait for the next wakeup. If the real time required to complete a dump cycle exceeds the wakeup interval, the dump cycle repeats immediately. The incremental volume dumper should be run whenever the Multics system is operational. Incremental volume dumping may be partitioned among several different processes, if desired, by specifying different physical volumes in different control segments. The dump control segment should specify at least all public volumes and may specify any registered volume. If a volume is not mounted, a message to that effect is printed, and the physical volume is skipped.

The incremental volume dumper is controlled by bit maps that the system maintains on a per physical volume basis. These bit maps designate the VTOC entries on a physical volume that are to be dumped. The segment or directory described by the VTOC entry is accessed in cooperation with the Multics supervisor in a manner that bypasses the storage system access control list and ring control.

Consolidated Mode

The consolidated volume dumper operates as a single-pass dump, using the same control segment as the incremental volume dump. It is effectively a merge operation of all the incremental volume dumps produced since the last consolidated volume dump. There is no system requirement that it be run, but its operation significantly reduces the amount of input that must be scanned during a volume reload or retrieval operation. The consolidated volume dumper is controlled by bit maps in the same way as for the incremental volume dumper. The segment or directory described by the VTOC entry is accessed in cooperation with the Multics supervisor in a manner that bypasses the storage system access control list and ring control.

Complete Mode

The complete volume dumper also operates as a single-pass dump but it dumps everything. The complete volume dump of a physical volume is the logical equivalent of a BCE save of a physical volume. The complete dumper constructs a temporary bit map of the VTOC entries currently in use before it starts dumping. The segment or directory described by the VTOC entry is accessed in cooperation with the Multics supervisor in a manner that bypasses the storage system access control list and ring control.

The three modes of volume dumping are invoked as separate commands with similar control arguments. The three commands are:

```
incremental_volume_dump
consolidated_volume_dump
complete_volume_dump
```

These commands are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

ADDING TO A DUMP CONTROL FILE

If it becomes necessary to add a physical volume to an existing Volume Dumper control file, the following procedure should be observed to ensure that the reload group is consistent:

- add the volume name to the end of the control file
- start the dump. The Volume Dumper asks if a restart should be performed. This is because it will have determined that the last volume dumped does not match the last volume name in the control file. This is the mechanism used to automatically restart a dump after a system interruption. Answer no to this question. The dump then continues from the first volume in the control file.

HANDLING ERRORS WHILE VOLUME DUMPING

Disk errors, tape errors, or file system errors can occur while volume dumping. The following description explains how to handle each of these types of errors, including fatal process errors and system crashes. In general, you can restart the volume dump after one of these errors by retyping the command line with which you started the dump and including in that line the -restart control argument.

The -restart control argument can be used to restart complete, consolidated, and incremental dumps. For complete and consolidated dumps, the default for -restart is to restart from the last volume dumped. For incremental dumps, you must specify which physical volume to restart from.

As the dumper walks through the volumes specified in the dump control file, it announces which volume is being dumped. Use this volume name with the -restart control argument, e.g.:

```
incremental_volume_dump -control control_file_name
    -operator operator_initials -restart restart_volume_name
```

Disk Errors

For disk errors (e.g., "dev inop" or the disk drive drops offline), if the dump terminates or is terminated by the operator, use the -restart control argument to indicate that dumping should resume with the specified volume. If the volume is unreadable, restart from the next physical volume in the sequence to be dumped. Transient disk errors usually cause no problems other than the immediate one.

Tape Errors

For tape errors, e.g., if the tape drive drops out of ready, use the -restart control argument and specify the last volume name announced by the dumper. If the tape breaks or is damaged during a complete dump, restart the entire dump. If the tape damage occurs during an incremental or consolidated dump, there is no recovery technique.

File System Errors

For file system errors (e.g., connection failure or "RQO"), correct the cause of the error and then use the -restart control argument.

Other Errors

For system crashes or fatal process errors by the dumper, use the -restart control argument.

Volume Retrieval

Although not intended for archival storage, the volume backup subsystem does provide a highly automatic retrieval system. Requests to the retriever, made by using the enter_retrieval_request command (see the *Multics Commands and Active Functions* manual, Order No. AG92), are queued for later processing. When you want to start retrievals you need only log in a process, normally the volume retriever, and issue the retrieve_from_volume command.

The retrieve_from_volume command examines the specified queue for retrieval requests. For each request, a determination is made as to whether the segment or directory to be retrieved exists online. If so, the volume ID of the latest dump volume is determined. If the volume ID is not available, then the set of all volume identifiers that might contain the segment or directory is determined. If the segment or directory is not online, then a recursive search is made for the dump volume containing the first superior online directory and the process is repeated.

As a result of the above operation there exists a temporary set of dump control segments in the process directory that contain entries for segments and directories that may be on that dump volume. The control segments and their associated volumes are processed in reverse chronological order. When a segment or directory is recovered, it is removed from all control segments.

Segments and directories are recovered as the result of a unique identifier (uid) match. The uid is a unique bit pattern associated with a particular segment or directory. It does not change if the segment is renamed. However, if a segment is copied, or a segment or directory is moved in the hierarchy, the new segment or directory acquires a new uid. This also occurs if a segment or directory is reloaded using the hierarchy backup system.

In order to know the uid of a segment or directory, as well as to be able to determine the requestor's access to recover it, the directory entry for that segment or directory must exist. If it does not, it is recovered first. Once the directory entry is recovered, the segment or directory can be recovered. Thus, for deleted segments and directories, retrieval is usually a two-step operation. In certain cases this is not true, because directories and the segments they describe usually reside on different physical volumes. Thus, a directory entry may be lost and retrieved without having to recover the segment or subtree.

The volume retriever obeys the access control rules of the system. Thus it will not retrieve a ring 1 segment from a ring 4 request, and it will not retrieve a segment to which the requester has no access.

As noted above, the volume retriever uses a uid match to find the requested segment or directory. If a segment has been copied or moved in the hierarchy or reloaded using the hierarchy backup system, it is not retrievable from any prior dump volumes under its former pathname.

Determining the set of dump volumes to be searched uses some of the segments created and maintained by the volume dumper. The volume log segment is used to determine the dump volumes that may contain the desired segment or directory and the order in which they should be searched. The volume log is searched for in the directory >ddd>volume_backup (unless the -wd control argument has been specified). In order to decrease the number of dump volumes that must be searched, the contents segment for each dump volume, if available, is searched for a uid match. If a uid match is not found, the dump volume is not searched. If the contents segment is not available, the dump volume is searched. If the contents names segment is available, it is used to further decrease the number of directories scanned during branch retrieval. If the retriever can determine, using the contents names segment, that the branch name was not in the directory when it was dumped, then the directory is not temporarily recovered and thus the number of dump volumes scanned is reduced. If the contents names segment is not available, the tape is searched.

A site may choose to charge for retrievals. To this end a retrieval account segment that contains the requestor's name and the number of segments and directories recovered is created in the directory:

```
>system_control_1>volume_backup_accounts
```

At this time no further processing is done; accounting is off by default.

For more information on volume retrieval refer to the description of enter_retrieval_request in the *Multics Commands and Active Functions* manual, Order No. AG92.

Volume Reloading

The volume reloader is used to reconstruct the contents of a physical volume. It uses as input the reload group indicated by the volume log segment. When invoked, the volume reloader determines from the volume log the set of dump volumes that defines a reload group. The reload group is processed in reverse chronological order so that once a segment or directory is recovered, subsequent copies can be skipped. When a volume reload has completed, the resultant physical volume is a logical image of its former self, less any changes that were introduced by the operator (information unrecorded by the dumper or unreadable due to operational errors).

Volume reloading of a physical volume that is of the root logical volume (RLV), but not of the root physical volume (RPV), is accomplished by bringing the system up to ring 1 initializer command level prior to accepting any physical volumes, recovering the associated volume log via the recover_volume_log command, and issuing the reload_volume command. If the volume to be reloaded is the RPV, the system must be cold booted to ring 1 command level using a spare disk pack. See "Disk Volume Recovery Procedures" in Section 10 for a detailed discussion of this.

The disk pack used by the volume reload facility is accessed as an I/O disk and must have been initialized via the init_vol initializer command, or restored from BCE, or in some way restructured. The disk pack must be initialized with the name and other parameters of the physical volume that is to be rebuilt. The parameters that describe the organization of the pack are compared with those stored in the volume log and any mismatches are reported to you so you may decide whether to continue the reload or reinitialize the physical volume. Volume reloading of any physical volume can be carried out while the system remains operational for users, although the logical volume that contains the physical volume to be reloaded is not available.

The volume reloader uses the user disk facility of the system to read and write the disk pack it is rebuilding. Thus, a user disk drive must be available (see the set_drive_usage command description in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64, and the description of the udsk card in Section 7).

The volume reloader can run in any user ring, at any AIM level, but it is normally run at a system-high AIM level in ring 1. The volume reloader requires re access to the following special gates:

```
hc_backup_
rcp_sys_
```

The volume reloader creates a control segment (in the working directory) for each physical volume that it reloads. The control segment is given the name of the physical volume plus the suffix "control"; e.g., dska_1.control. This control segment contains information about already reloaded objects and allows the reload operation to be restarted should it be interrupted by a system failure.

If the volume being reloaded contains partitions, the config cards describing the partition must be removed from the configuration deck prior to booting. Once the reload is complete, the system must be shut down and the configuration deck updated. For more information, see Section 7.

When a volume reload is complete, in all cases except that of the RPV the physical volume can be used as a direct replacement of the original volume by remounting the logical volume, if required, and adjusting the disk table, or putting the new disk pack on the old disk drive. In the RPV case, you must take additional steps: shut the system down; either move the disk pack to the drive specified on the root config card, or change the card; and reboot BCE.

HIERARCHY BACKUP

The Multics hierarchy backup system protects against the destruction of information maintained by the Multics storage system. The hierarchy backup system preserves recent copies of all segments and directories known to the storage system on magnetic tape, and recovers these copies when needed.

The hierarchy backup system performs the following functions:

1. hierarchy dumping

   The hierarchy dumping mechanism copies segments and directories from the Multics directory hierarchy onto tape.

2. hierarchy retrieval

   Hierarchy retrieval is the recovery, during normal Multics operation, of specified segments and directories that have been copied onto tape.

3. hierarchy reloading

   Hierarchy reloading is the recovery of the entire or partial contents of online storage in order to resume Multics operation (generally done after a system failure).

Users are normally concerned only with system hierarchy dumping and hierarchy retrieving, since reloading is a system function performed when the need arises. The frequency of hierarchy dumping and the length of time that hierarchy dump tapes are preserved are installation-determined parameters. Examples given in the following text are typical values. The operator must check with the local installation procedures to find out the parameters for the particular site.

The Hierarchy Backup LSS

Some of the commands for the hierarchy backup system are available under a Limited Service Subsystem (LSS). Within this LSS, there is one LSS command table for the two hierarchy dumpers (Backup.SysDaemon and Dumper.SysDaemon). This command table restricts the hierarchy dumpers' available command set to:

```
backup_cleanup
catchup_dump
complete_dump
end_dump
start_dump
wakeup_dump
```

and:

```
exec_com
help
home_dir
logout
system
user
```

All the commands listed above are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64, with the exception of list_retrieval_requests, exec_com, help, home_dir, logout, system, and user, which are documented in the *Multics Commands and Active Functions* manual, Order No. AG92. (Note that "help" and "logout" here refer to the standard Multics commands by those names, not the inititalizer commands by those names.)

If your site runs the hierarchy dumpers under the supplied LSS, the project_start_up.ec for the SysDaemon project (sysdaemon_project_start_up.ec) selects the appropriate command table for the two hierarchy dumper Person_ids based on the result of the [user name] active function.

Also, if your site runs the hierarchy dumpers under the supplied LSS, you must be careful about making changes to admin.ec. If you change admin.ec so that commands not contained in the command table are sent to the hierarchy dumper daemons, your operators will not be able to use the "x" command(s) affected by the change.

You may extend the command set available to the hierarchy dumper daemons by modifying the command table and creating a new command set via the make_commands command (documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64).

You may allow the full Multics command set to be available to the hierarchy dumper daemons by not executing the enter_lss command in the sysdaemon_project_start_up.ec. This means that the hierarchy dumpers will not run as an LSS.

Note that while the volume backup LSS restricts the commands available to all three of the volume backup daemons (the volume dumper, the volume retriever, and the volume reloader), the hierarchy backup LSS only restricts the commands available to two of the hierarchy backup daemons (the two hierarchy dumpers). Thus, access to the hierarchy retriever (Retriever.SysDaemon) and the hierarchy reloader (Reloader.SysDaemon) must be restricted to trusted individuals.

Hierarchy Dumping

The hierarchy dumping mechanism searches out, selects, and copies onto tape segments from the Multics directory hierarchy. At the same time, it produces a map indicating the segments and directories included in each dump cycle. The hierarchy dumper operates in three modes: incremental, consolidated and complete. Usually, only information that has changed and will be valuable in the future is dumped. Thus per-process information, static libraries, and re-creatable segments and directories are not dumped.

It is recommended that all incremental hierarchy dump tapes, consolidated * hierarchy dump tapes, and complete hierarchy dump tapes be recorded in a physical "pencil and paper" log (the Multics operations log). This log will be self-explanatory if you adopt conventions for distinguishing between the different kinds of dumps, and stick to them.

The maps produced by the hierarchy dumper should also be preserved in a reverse chronological log. This will aid you later during any hierarchy retrieval that may be required.

Any ring 1 processes may be used to perform hierarchy backup operations. However, we recommend that you use the following User_ids:

```
Backup.SysDaemon
Dumper.SysDaemon
Retriever.SysDaemon
Reloader.SysDaemon
```

Whatever processes are used for hierarchy backup operations should not be used for any other purposes.

Two system processes are generally employed by the hierarchy dump system for the purpose of dumping. These are: Backup.SysDaemon and Dumper.SysDaemon. The Backup.SysDaemon process is used to produce incremental and consolidated dumps. The Dumper.SysDaemon process is used to produce complete hierarchy dumps. Complete hierarchy dumps can be produced concurrently with incremental or consolidated hierarchy dumps. The names of the processes may be anything selected by the site; Backup and Dumper are used at most sites.

INCREMENTAL MODE

The incremental hierarchy dumper locates and copies all segments and directories that have been modified more recently than they have been dumped. For any given segment or directory this criterion is determined by comparing the date and time modified attribute and the date and time dumped attribute. The period of time for incremental hierarchy dumping is short and is determined by each site (the default is one hour).

CONSOLIDATED MODE

The consolidated hierarchy dumper locates and copies segments and directories that have been modified after some specified time in the past. For example, an installation might choose to run a consolidated hierarchy dump every midnight to copy all segments and directories modified since the previous midnight. Consolidated hierarchy dumps collect the most recent copies of segments and directories modified since the specified time in order to reduce the time needed to reload them from tape. Also, the consolidated hierarchy dumper provides additional copies of those segments or directories that have been backed up.

Normal incremental hierarchy dumping continues after consolidated hierarchy dumping terminates. The process asks you to mount new tapes when the consolidated hierarchy dump is complete.

COMPLETE MODE

A complete dump dumps every segment or directory without regard to time modified. The complete dumper does not interact with the incremental or consolidated dumpers.

A complete hierarchy dump establishes a checkpoint in time, essentially a snapshot of the entire Multics storage hierarchy. If it should ever become necessary to perform a complete hierarchy recovery, then the most recent complete hierarchy dump marks a point in time beyond which no older incremental or consolidated hierarchy dump tapes need be inspected.

Another purpose of complete hierarchy dumping involves tape retention strategy. The high production rate of incremental and consolidated hierarchy dump tapes makes the long-term retention of these tapes difficult. Therefore, incremental and consolidated hierarchy dump tapes may be kept for a short period; for example, three weeks; and complete hierarchy dump tapes may be kept for a longer time; for example, six months to a year.

Hierarchy Retrieval

The hierarchy retrieval system is used to recover segments and directories from tapes produced by the hierarchy dumper. Hierarchy retrieval occurs during normal Multics system operation.

A user who notices that a segment or directory has been lost or damaged can submit a request for hierarchy retrieval to the Multics operations staff. It is necessary to determine which hierarchy dump tape of the segment or directory to retrieve. Usually the most recent tape is desired. In the case of a damaged segment, the damaged version may have been dumped. In this case, an earlier dump tape is desired. Hopefully, a user can inform the operations staff of approximately when a segment was lost or damaged, and whether the segment had been recently modified. Using these two pieces of information, it is possible to make a reasonable guess as to which hierarchy dump tape should be processed.

Once a conjecture has been made as to which hierarchy dump tape is to be retrieved, it can be verified by examining the corresponding hierarchy dump map. The map indicates the segments and directories written on that tape. A feature of the dump map that is sometimes helpful is the date and time dumped attribute for the segment, which points to the approximate location of the next most recent copy of the segment to be recovered.

The user can specify that a single segment, a directory without its subtree, or a directory with its subtree be recovered. (A directory for which the subtree is not recovered contains only the links and access control information associated with the directory itself.)

A user can also specify that a segment or directory be recovered with a different pathname. This is called cross retrieval. Segments and directories can be cross-retrieved to any point in the storage system hierarchy.

Hierarchy retrievals may be done by operators, but they should have programming staff assistance. The step-by-step procedure for performing a hierarchy retrieval is available in the *Operator's Guide to Multics*, Order No. GB61.

Hierarchy Reloading

Hierarchy reloads are necessary when some hardware or software problem has damaged major portions of the storage system hierarchy.

In the event that segments are lost or destroyed, the programming staff should be notified immediately and the operator should proceed under their supervision.

Operators should be instructed to keep all salvager output, maps, and online printouts for the programming staff. Also, any problems encountered during the reload should be reported to the programming staff.

Hierarchy reloads may be performed from the initializer only if the reload command is issued as the first command to the initializer after a BCE boot command has been given. For this purpose, the initializer enters a special environment after a boot called the administrative ring.

Hierarchy reloads are discussed in more detail in Section 10 and in Appendix H.

BACKUP COMMANDS

Complete descriptions of the backup commands are presented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

SECTION 10

RESPONDING TO SYSTEM PROBLEMS

MULTICS SYSTEM FAILURES

This subsection describes Multics system failures. It includes information on how Multics crashes (i.e., returns to BCE), how Multics takes a dump, how you can examine a crashed system, and how Multics performs an emergency shutdown.

Understanding System Failures

Multics fails when it stops providing services to some or all users. There is a cycle of three steps which usually takes place whenever Multics fails. These steps are crashing, dumping, and emergency shutdown. Once these three steps are complete, the system can be recovered. Usually, the only recovery needed is rebooting. Sometimes, more recovery is needed during or after the reboot.

CRASHING

Multics is designed to detect system failures by continually checking for errors. It detects errors in two ways: by intercepting hardware faults in critical system code and by checking critical system databases for consistency. Multics responds to serious errors by stopping operations in a controlled way. The process of stopping operations is called crashing.

In order to crash, Multics must do two things: it must save its state and it must transfer control to another system. The system it transfers control to is BCE. It transfers control to a copy of BCE which is saved as part of system initialization. Ordinarily, BCE will work correctly even in the face of the problem that stopped Multics. There are two reasons for this. First, the saved copy of BCE is generally safe from damage due to transient disk errors. Second, BCE uses less of the hardware than Multics. For example, a serious hardware failure of a tape channel might force Multics to crash. Since BCE does not use the tape hardware, it could still operate. Information on how to recognize when Multics has crashed is available in the *Operator's Guide to Multics,* Order No. GB61.

Multics does not always succeed in detecting serious errors. When it doesn't, it * fails without crashing. There are two kinds of failures which don't crash the system. One is a result of the system looping in low level code (i.e., in the hardcore). The other is a result of the initializer process hanging. The following events are all signs that the system is looping:

- The system stops responding to your commands

- One or all user terminals hang
- Neither the system administrator nor any users can log in
- Users who are logged in can't log out
- You receive a large number of messages which look something like this:

```
pxss: notify timeout, event = 144163153167,
  processid = 367256147361
```

- On a L68 system, the lights on the control panels of all of the processors are steadily lit (i.e., they stop blinking)
- On the system indicator panel (if your site has one), all of the lights are steadily lit except for the bottom one

The following events are all signs that the initializer process is hanging:

- The rest of the system seems to be fine, but the initializer doesn't respond to your commands (i.e., the bootload console doesn't respond, or it responds, but the commands you issue don't seem to get executed)
- Users can't log in and out

When Multics fails without crashing, operators or system maintainers must manually crash the system using the "execute fault" or "execute switches" mechanisms described later in this section. Sometimes it is impossible to get the system to crash. The most common reason for this is a power failure that destroys the contents of main memory. In this case, dumping and emergency shutdown cannot take place, and the system must be rebooted from scratch.

DUMPING

A dump is a selective copy of information from a crashed system that is saved for later examination. Two facts about the way Multics is designed ensure that dumps are an accurate image of the crashed system. First, as stated above, Multics saves its state before it transfers control to BCE. Second, when it returns to BCE, it leaves things exactly as they were. This permits BCE to take a dump of the crashed system which reflects its exact state when the error was detected. You can examine the dump with the analyze_multics subsystem. Note that it is also possible to examine the crashed system in place with the BCE probe subsystem. (The analyze_multics and BCE probe subsystems are described later in this section under "Examining a Crashed System.") Sometimes it's impossible to get a dump. In this case, you should attempt to examine the crashed system with BCE probe and proceed with emergency shutdown.

EMERGENCY SHUTDOWN

The next step after the dump is emergency shutdown. During an emergency shutdown, Multics is restarted in a very limited way to clean up file system operations that were in progress at the time of the crash. (The system returns to BCE when emergency shutdown finishes.) Thus, ESD can only succeed if the system crashed, preserving the Multics image. If ESD succeeds, then all page control operations are left in a consistent state. This means that no data in user segments is lost, and no free pages are incorrectly marked in use. However, higher level operations are not cleaned up. For example, directories can be left in inconsistent states, and changes to directories can be lost. The directory salvager corrects these inconsistencies. Sometimes, it's impossible to get an emergency shutdown. In this case, you should continue by rebooting the system.

How Multics Crashes

The following sections contain detailed descriptions of Multics code.

NOTES ON THE MULTICS OPERATING ENVIRONMENT

At any given time, each configured Multics CPU is running some process, and each process is running some program. When there's an error, the process which detects the error is called the crash process. It just means that her process happened to detect the error. This does not mean that the user who owns the process caused the carsh. Some system messages refer to the owner of the crash process as the control process.

When a process detects an error, it is the program being run by that process which makes the decision to crash. Different programs execute in different environments. In some environments, programs are allowed to make external calls to other programs. In other environments, they are not. So Multics provides different ways for programs to crash.

SYSERR CRASHES

The most common way that Multics crashes is by a system program which can make external calls discovering some type of fatal error. The program reports the error via a call to syserr with a severity code of CRASH (1). (See syserr_constants.incl.pl1 for the standard syserr severity codes.) Any program running in the normal PL/I runtime environment of the supervisor can call syserr. When one does, syserr types a message on the bootload console (via a call to ocdcm_) that describes the surface cause of the crash. An example of a syserr crash message is:

```
0802.1 lock: AST lock locked at dir unlock time.
```

The beeper is always turned on for crash messages. Syserr calls privileged_mode_ut$bce_and_return (pmut$bce_and_return). This program sets the scs$sys_trouble_pending flag to the value of scs$processor, stores the process_id of the crash process into scs$trouble_processid, and issues a sys trouble connect.

Sys Trouble Connects

A sys trouble connect is a way for a program to make the CPU on which it is running enter the low level crash handler known as sys_trouble. A program issues a sys trouble connect by executing a CIOC (connect) instruction whose target is the CPU on which it is running. Sys trouble connects are handled as explained below under "Sys Trouble Connect Handling."

RING ZERO DERAIL CRASHES

A second way that Multics crashes is by a ring zero system program which can't make external calls (and is not part of the fault handler) discovering some type of fatal error. The program "reports" the error by executing a DRL (derail) instruction. This mechanism allows ALM programs which cannot call syserr to crash with an informative message. At compile time, these programs use drl_macros.incl.alm to set up DRL instructions. Each DRL instruction's effective address is the location in the object segment of a 32 character message. At run time, one of these DRL instructions will be executed if there's an error, causing a derail fault. The derail fault handler in fim.alm checks the ring of execution (the PRR) at the time of the derail fault. If it's zero, the derail fault handler transfers control to fim_util$drl_fault_trouble. This program sets the scs$sys_trouble_pending flag to the code "trbl_r0_drl_flt", stores the process_id of the crash process into scs$trouble_processid, and issues a sys trouble connect (defined above). The eventual result is a flagbox message of the form: "module: reason." When the system returns to BCE, BCE prints this flagbox message. Note that ring zero DRL instructions are also used for BCE/Multics breakpoints; a DRL with effective address -1 is interpreted as a breakpoint.

INVALID FAULT CRASHES

A third way that Multics crashes is by the program fim.alm or the program fim_util.alm detecting an invalid fault. These two programs can detect a number of faults in inappropriate circumstances, such as page faults in wired environments. Since neither of these programs can print a message, when they detect an invalid fault, they set the scs$sys_trouble_pending flag to one of a number of sys trouble codes for invalid faults. (There is one code for each kind of invalid fault.) Sys trouble codes are defined in sys_trouble_codes.incl.pl1 and sys_trouble_codes.incl.alm. Later on, in the sys trouble connect handler, the code is used to select a flagbox message. In addition to setting the flag, fim.alm and fim_util.alm store the process_id of the crash process into scs$trouble_processid, and issue a sys trouble connect (defined above).

EXECUTE FAULT AND UNEXPECTED FAULT CRASHES

A fourth way that Multics crashes is via an execute fault or an unexpected fault. An execute fault is a manual crash, caused by an operator or system maintainer. It is the preferred way of causing a manual crash, especially on a multi-processor system. As mentioned earlier, the system must be manually crashed when it is looping or when the initializer process is hanging. To do an execute fault on a Level 68 system, the operator sets the EXECUTE SWITCHES/EXECUTE FAULT switch on the display panel of any processor to EXECUTE FAULT, then presses the EXECUTE button on the display panel. On a DPS 8 system, the operator presses the EXECUTE button on the configuration panel of any processor. On a DPS 8 system, an execute fault may also be done by using the appropriate DPU/DMP VIP mode EX command. The step-by-step procedure for executing fault is described in the *Operator's Guide to Multics,* Order No. GB61.

An unexpected fault is either a trouble fault or a fault which is undefined by the hardware. A trouble fault is signalled by the hardware when it encounters an error signalling some other fault. This can be caused by CPU hardware problems or by corrupted data in the fault vector. An undefined fault is always the result of CPU hardware errors.

The handler for execute faults is wired_fim$xec_fault. The handler for unexpected faults is wired_fim$unexp_fault. These programs set the scs$sys_trouble_pending flag to the sys trouble code for execute fault or unexpected fault, respectively. Sys trouble codes are defined in sys_trouble_codes.incl.pl1 and sys_trouble_codes.incl.alm. Later on, in the sys trouble connect handler, the code is used to select a flagbox message. In addition to setting the flag, wired_fim$xec_fault and wired_fim$unexp_fault store the process_id of the crash process into scs$trouble_processid, and issue a sys trouble connect (defined above). The machine conditions are put directly into prds$sys_trouble_data.

CHECK-STOP CRASHES

A fifth way that Multics crashes is by a system programmer using the check-stop debugging technique. To use this technique, a system programmer sets the first nine DATA switches on the maintenance panel of the bootload processor to an octal 123. (Note that this can only be done on a Level 68 CPU.) He sets the rest of the switches to one of a number of unique codes which each specify a step in the initialization process. These codes are defined in real_initializer.pl1.pmac. This program checks the switches before each step in intialization, and crashes the system if it finds the code for that step in the switches. It crashes the system by calling syserr with a severity code of CRASH (1). (See syserr_constants.incl.pl1 for the standard syserr severity codes.) Syserr types a message on the bootload console that describes the surface cause of the crash, and calls pmut$bce_and_return. This program sets the scs$sys_trouble_pending flag to the value of scs$processor, stores the process_id of the crash process into scs$trouble_processid, and issues a sys trouble connect (defined above). A check-stop crash returns to BCE in a restartable manner.

hphcs_$call_bce CRASHES

A call to the privileged gate hphcs_$call_bce will crash the system (i.e., force a return to BCE) via a call to pmut$bce_and_return. This program sets the scs$sys_trouble_pending flag to the value of scs$processor, stores the process_id of the crash process into scs$trouble_processid, and issues a sys trouble connect (defined above). Note that the initializer bce command calls this gate.

SYS TROUBLE CONNECT HANDLING

All of the ways that Multics can crash described so far end by issuing a sys trouble connect. The program which is called to process sys trouble connects is known as sys_trouble. It stops all processors other than the bootload processor, and transfers execution to the BCE toehold.

There are several different kinds of connects. The following paragraphs describe how the system handles connects in general, and then how it handles sys trouble connects in particular.

When the system sends a connect to a processor, the target takes a connect fault. The fault vector specifies that the processor should transfer control to prds$fast_connect_code on a connect fault. (The source of this code is fast_connect_init.alm.) The fault vector also stores the machine conditions in prds$fim_data. If the scs$sys_trouble_pending flag is non-zero, the "fast connect code" transfers control to wired_fim.alm. This module is the handler for faults which the system can legitimately take while running with the PRDS (the processor data segment) as a stack. On a connect fault, wired_fim is entered at wired_fim$connect_handler. If the scs$sys_trouble_pending flag is non-zero (which at this point means that there's a sys trouble connect), wired_fim.alm transfers control to sys_trouble$sys_trouble.

Remember that the system got to this point because a processor sent itself a sys trouble connect. The processor which did this is usually the first one to enter sys_trouble$sys_trouble. However, some other processor may notice that the scs$sys_trouble_pending flag is non-zero and enter sys_trouble$sys_trouble first.

Whichever processor enters sys_trouble$sys_trouble first notices that it is the first processor to do so, and "broadcasts" (sends) connect faults to all of the other processors. This broadcast causes the other processors to enter sys_trouble via the path explained above. They do not perform the broadcast. The system uses scs$trouble_flags to ensure that the broadcast only occurs once. Each bit in scs$trouble_flags corresponds to a processor. If a processor finds its bit turned off, it broadcasts. If a processor finds its bit turned on, it clears its bit and doesn't broadcast. The way a processor determines whether or not its bit is turned on is by comparing the value of prds$processor_tag with the value of scs$trouble_flags. The first processor to enter sys_trouble$sys_trouble finds its scs$trouble_flags bit turned off, and broadcasts. It also turns on the bits correspoding to all of the other processors. So when the other processors enter sys_trouble$sys_trouble, they find their scs$trouble_flag bits turned on

and don't broadcast. At this point, all of the processors, including the first one, are in sys_trouble, executing the same code. Each processor copies the machine conditions of the connect fault from prds$fim_data (in the per-processor data segment PRDS) to prds$sys_trouble_data.

BCE only runs on one processor. The procesor it runs on is the bootload processor. The bootload processor is initially defined at bootload time as the CPU on which BCE last executed; however, should this processor be deleted, reconfiguration will assign the responsibility of being bootload processor to some other CPU. Each processor must determine whether or not to enter BCE. The criteria it uses is whether it is the system-defined bootload processor, *NOT* whether it is the processor on which the crash was detected. The identity of the bootload processor is defined by scs$bos_processor_tag. (The name is left over from the time when BOS was the primary crash handler.) The way a processor determines whether or not it is the bootload processor is by comparing the value of prds$processor_tag with the value of scs$bos_processor_tag.

If there was an invalid fault, execute fault, or unexpected fault crash, and the scs$sys_trouble_pending flag has been set to one of the sys trouble codes defined in sys_trouble_codes.incl.pl1 and sys_trouble_codes.incl.alm, then sys_trouble copies the correct message into flagbox.message, and sets the bit flagbox.alert in the flagbox segment. Similiarly, if there was a ring zero derail crash and the scs$sys_trouble_pending flag has been set to "trbl_r0_drl_flt", sys_trouble copies the derail message into flagbox.message and sets the bit.

The bootload processor proceeds to enter BCE. The other processors execute a DIS instruction in sys_trouble. A DIS (Delay until Interrupt Signal) instruction is an effective HALT instruction. If the BCE go command, which restarts Multics, is subsequently executed, connects will be rebroadcast to all CPUs, "interrupting" them out of their DIS state. The machine conditions for this subsequent fault will be put in prds$fim_data. Note that the copying of the machine conditions to prds$sys_trouble_data, described above, prevents this possible subsequent connect fault from overwriting the first set of machine conditions (the "sys trouble data").

The actual entry of BCE happens as follows. The bootload processor loops for a while to allow all pending I/O operations to finish. This loop is inhibited, so the processor may take lockup faults. Therefore, the lockup fault vector is temporarily set to wired_fim$ignore_fault. Once the loop is completed, an instruction pair consisting of an SCU and a TRA is picked up from location TOE_HOLD_MULTICS_ENTRY (3) * 2 of the BCE toehold. (The toehold begins at location 24000 octal in absolute memory. See toehold_save_dcls_.incl.pl1 for more information.) The instruction pair is patched into the derail fault vector. Finally, a DRL instruction is executed. This causes the instruction pair that was just put into the derail fault vector to be executed. The SCU instruction stores the SCU data in the toehold. The TRA instruction transfers control to the toehold in absolute mode.

EXECUTE SWITCHES CRASHES

The final way that Multics crashes is via an execute switches. This is a manual crash, caused by an operator or system maintainer. It should only be done if an execute fault has been unsuccessful. As mentioned earlier, the system must be manually crashed when it is looping or when the initializer process is crashing. Executing switches is a way of forcing a processor to enter the toehold without having to execute any Multics code. Therefore, this method of crashing the system may get the system to BCE even when none of the methods described above succeed.

Executing switches is done by using the processor's execute switches facility. This facility is a way of telling the processor to execute an instruction pair at a specified absolute location. The toehold has three instruction pairs in it which can be executed in this way. When the processor executes the pair at TOE_HOLD_CRASH_ENTRY*2 (24000), it returns directly to BCE. When the processor executes the pair at TOE_HOLD_ESD_ENTRY*2 (24002), it enters ESD directly. When the processor executes the pair at TOE_HOLD_DUMP_ENTRY*2 (24004), it requests an "early dump" -- a tape dump of the first 512K of memory. This is only useful during collection one initialization.

To execute switches on a Level 68 system, the operator sets the DATA switches on the maintenance panel of any processor to "02400N717200", where N is 0, 2, or 4 as described above. To execute switches on a DPS 8 system, the operator uses either the BCE 24000, 24002, or 24004 command. The step-by-step procedure for executing switches on either system is described in the *Operator's Guide to Multics,* Order No. GB61.

The execute fault method of manually crashing the system ensures that all processors are stopped via sys trouble connects. But the processor on which switches are executed doesn't do anything to stop the other processors. Thus, it is *CRUCIAL* that the operator stop the other processors before executing switches. Obviously, this is not a problem on a one-CPU system.

How Multics Takes a Dump

There are two kinds of Multics dumps: early and normal. Early dumps are taken by collection zero initialization and put on magnetic tape. They are produced in two situations. The first is when collection one fails (in which case, it asks you for the number of a tape drive on which it can write the dump). The second is when the operator executes switches specifying location 24004, requesting an early dump. Early dumps are memory images of the first 512K of memory, which is all the memory used during collections zero and one of initialization. The Multics read_early_dump_tape command is used to read these dumps into the file system for analysis.

Normal dumps are taken with the BCE dump command and put in the DUMP partition of the RPV. They are partial snapshots of the Multics virtual memory, including the databases of processes and segments within processes which contain information relevant to the crash. The privileged Multics copy_dump command is used to read normal dumps into the file system for analysis.

THE BCE dump COMMAND

The BCE dump command writes selected segments from selected processes in the crashed system to the DUMP partition of the RPV disk. The selection of exactly which processes and which segments get dumped is controlled by the control arguments supplied to the dump command. The dump command scans the APT (the Active Process Table, located in the segment tc_data) of the crashed system, and selects processes based on the criteria specified by the control arguments. For each process it selects, the dump command then scans the descriptor segment of that process and selects segments based on the criteria specified by the control arguments. Normally, all of the supervisor databases are dumped, as well as supervisor data in running processes. Pure segments (procedures and fixed data) are never dumped, because they cannot contain clues to what went wrong. (It's possible that a severe hardware failure could damage pure segments, but the need to dump them for this reason is extremely rare and does not justify the vast expense of dumping them regularly.)

The layout of the DUMP partition is as follows. First there is a header, which describes what segments (by number and length) have been dumped, and contains the machine conditions from the return to BCE. Then there are segment images. For each process dumped, the segment images are in order by ascending segment numbers. Figure 10-1 depicts the layout of the DUMP partition following execution of the dump command.

After the system is rebooted, the dump has to be copied out of the DUMP partition into permanent storage. This is done by the privileged Multics copy_dump command, which is usually part of Utility.SysDaemon's start_up.ec. The copy_dump command uses the gate hphcs_ and therefore is generally executed by Initializer.SysDaemon or Utility.SysDaemon. It determines whether the DUMP partition contains a valid dump. If it does, the information in the DUMP partition is copied into one or more segments in the directory >dumps. These segments have the name date.time.n.dump_no, where date is in the form MMDDYY, time is in the form HHMMSS, n is a number, starting at 0, incremented by one for each segment of a multi-segment dump, and dump_no is a number incremented by one each time a dump is taken. The information in these segments is in the same format as it is in the DUMP partition. However, since the DUMP partition can be much longer than the maximum length of a segment, each 255K of dump information is placed in its own segment.

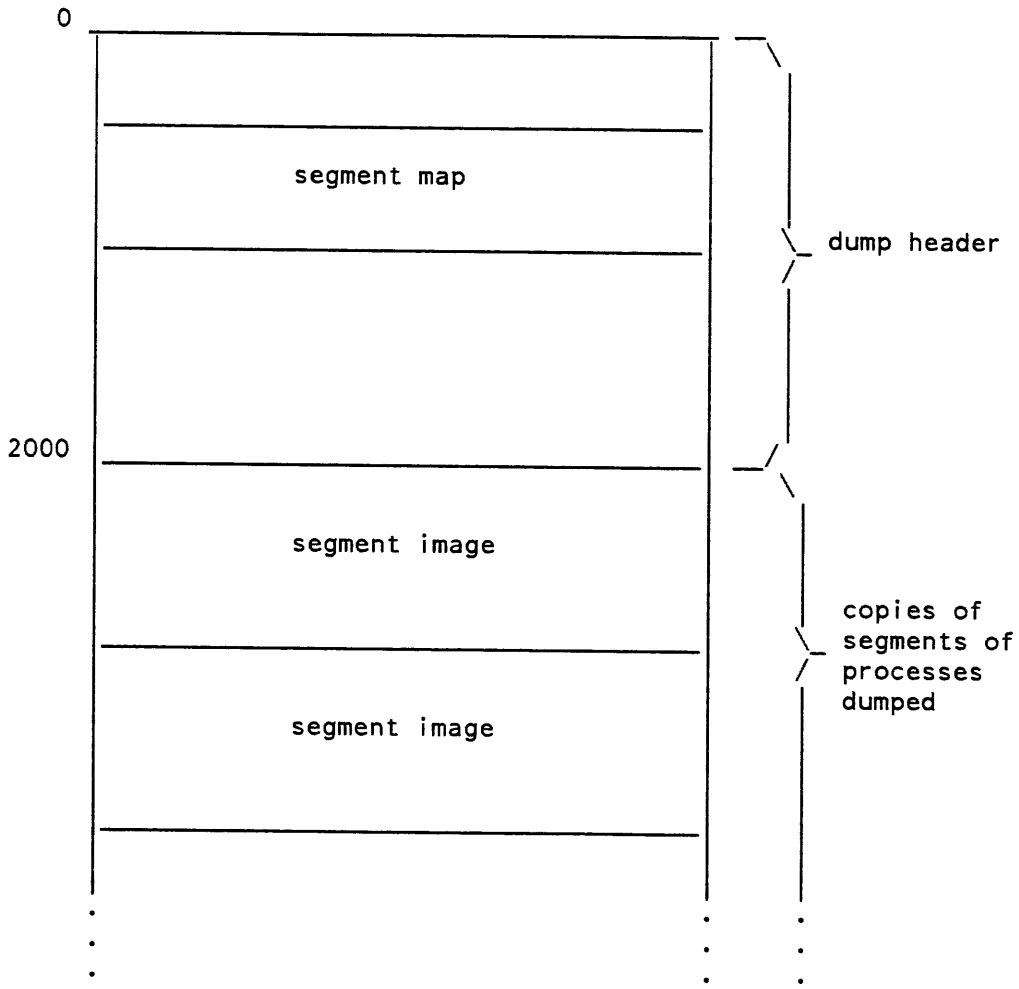


Figure 10-1. Layout of the DUMP Partition

Examining a Crashed System

To examine a crashed system, use either the analyze_multics subsystem (AZM) or the BCE probe subsystem. Normally, you will reboot the system and examine the crash under AZM. Using the BCE probe subsystem is only necessary when the BCE dump command fails, when the system crashes repeatedly and you can't reboot it, or when you strongly suspect a hardware problem and want to see more information before rebooting.

To examine a crash with AZM, type:

```
azm
sld <dump number>
```

To examine a crash with the BCE probe subsystem, type:

```
probe -crash
```

The most useful azm requests are:

`display, d`	displays a selected portion of a segment in a dump
`events`	displays significant events which occurred just prior to the crash, in reverse chronological order
`machine_conditions, mc`	displays all or parts of machine conditions, based on the given pointer
`stack, sk`	traces a given stack
`why`	provides a brief description of the immediate cause of the crash

The most useful probe requests are:

`display, ds`	displays a set of locations in a specified mode
`mc`	displays, in interpreted form, the SCU data found within the machine conditions at the specified address
`stack, sk`	displays a stack trace starting at the given address

For a complete description of the azm and BCE probe commands, refer to the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

LOCATING THE RELEVANT PROCESS

When Multics crashes, it is executing on behalf of some process. This process is called the crash process (or the "return to BCE" process). During initialization, the crash process is always the initializer. When the azm and probe commands start up, they use the information stored in the BCE toehold to find the crash process. Then they select it. Selecting a process means setting things up so that virtual addresses are interpreted within that process' address space. This means that any virtual addresses that you type in or that azm or probe types out are relative to the crash process. In some rare cases, it will be impossible for azm and probe to determine which process returned to BCE. These cases are beyond the scope of this manual.

EXAMINING THE TOEHOLD MACHINE STATE

The most important thing to examine is the machine state from the return to BCE, known as the "toehold machine state." This machine state, as well as all of the items stored in the toehold at BCE entry time, is described by the structure mc_state in the include file toehold_save_dcls_.incl.pl1. The most important part of the toehold machine state to look at is the machine conditions, known as the "toehold machine conditions." These are a standard set of Multics machine conditions. Depending on the way in which the system returned to BCE, different information in the toehold machine conditions will be relevant.

Examining the Toehold Machine Conditions for Execute Switches Crashes

For crashes via execute switches, the toehold machine conditions will reveal what the system was doing when the operator executed switches. To see these machine conditions in AZM, type:

```
mc -dump
```

To see these machine conditions in the BCE probe subsystem, type:

```
mc toehold|2760
```

Note that the constant "2760" is correct for MR12.0. If your site modifies toehold_save_dcls_.incl.pl1, this number may change.

By looking at the value of pointer register 6 (PR6), you can determine what stack was in use at the time of the crash and investigate the circumstances further. There are some other items stored in the toehold at BCE entry time which may be useful in investigating certain crashes, including, for example, history registers. To display the history registers in AZM, type:

```
hregs -dump
```

To display the history registers in the BCE probe subsystem, type:

```
ds <address of hregs in toehold>
```

Note that AZM interprets history registers, while the BCE probe subsystem merely dumps them in octal.

Examining the Toehold Machine Conditions for Non-Execute Switches Crashes

For all crash mechanisms other than execute switches, the toehold machine conditions will describe the machine state at the time of the derail fault that returned to BCE. This machine state depends almost entirely on the text of the program (sys_trouble or pmut) that executed the derail fault. For this reason, it usually isn't neccessary to look at the entire set of machine conditions. PR2 will be useful if you're having trouble finding prds$sys_trouble_data in the dump, since it will contain the address of prds$sys_trouble_data.

As explained above, prds$sys_trouble_data will contain the machine conditions of each process at the time of the crash. PR6 in these machine conditions will show what stack the system was running on when it crashed. If the system crashed via the syserr, ring zero derail, or hphcs_$call_bce mechanism, you should trace the relevant stack to determine the circumstances surrounding the crash.

If the system crashed via the invalid fault mechanism, the prds$sys_trouble_data machine conditions will show which program crashed the system.

If the system crashed via the execute fault or unexpected fault mechanism, the prds$sys_trouble_data machine conditions are the machine conditions of the fault itself. For an execute fault, these machine conditions should allow you to determine why the system was doing whatever provoked you to crash it in the first place. For an unexpected fault, these machine conditions (together with the crash history registers) should allow you to diagnose the problem. This kind of fault is almost always caused by either a corrupt fault vector or a CPU hardware error.

EXAMINING OTHER MACHINE CONDITIONS

Most of the time, system crashes are associated with faults or interrupts. To find out why the system crashed, you have to find the machine conditions for the problematic fault or interrupt. If you've gotten this far and still haven't been able to figure out why the system crashed, you can try looking at all of the places where the system stores machine conditions. These are:

```
pds$fim_data
pds$signal_data
pds$page_fault_data

prds$fim_data
prds$interrupt_data
```

To see which faults store machine conditions in which of these areas, see the programs initialize_faults_data.cds and initialize_faults.pl1. Once you find the relevant machine conditions, you must look at the SCU data and possibly the history registers to determine the reason for the fault.

After performing the basic analysis described so far, you may need to examine other processes beside the crash process to get a complete understanding of the crash. Such an examination is beyond the scope of this manual.

How Multics Performs an ESD

The emergency shutdown operation is initiated by the BCE esd command, which forces a transfer of control to the Multics emergency shutdown procedure and restarts the Multics memory image. The emergency shutdown procedure attempts to flush main memory contents onto the disk volumes and to shut all disk volumes down. If emergency shutdown completes without error, no information in user segments is lost. If the system crashed in the middle of a directory update, the directory may be left in an inconsistent state. If a user references the directory before a manual salvage is done, the directory will be salvaged automatically.

RECOVERING FROM SYSTEM FAILURES

This subsection describes automatic and manual recovery from Multics system failures, including information about what to do when recovery fails.

In general, recovering from a system failure involves the following steps:

1. Returning the system to BCE.
2. Taking a dump of Multics.
3. Shutting down the file system as well as possible, which usually means performing an ESD.
4. Rebooting Multics.
5. If the failure prevents a successful boot of Multics, resolving the problem which caused the failure, be it software or hardware.

Automatic Recovery

Normally, automatic recovery procedures are enabled after a system failure. There are 36 switches set up in the BCE toehold for communication between Multics and BCE. These switches are set either by the BCE set_flagbox command or by the privileged Multics set_flagbox command (both of which are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64). One of these switches means "automatic reboot mode is on." When the system is running in automatic reboot mode and returns to BCE, the flagbox bce_command variable is set to a command that tests the "crashed" indicators to discover whether the system failed or shut down normally. If the test indicates a system failure, automatic recovery procedures begin. These procedures do the following:

1. Take a dump of Multics (using the BCE dump command).
2. Perform an emergency shutdown (using the BCE esd command).

If the system is running in unattended mode, these procedures may also:

3. Bring the system up again (using the BCE boot command). Minimally required salvaging is done automatically as the system is brought up.

(See the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64, for descriptions of the BCE dump, esd and boot commands. See the *Operator's Guide to Multics*, Order No. GB61, for descriptions of the step-by-step procedures to follow when you're making use of automatic recovery procedures, in both automatic and manual modes.) The operator should record the crash according to your site's policy as soon as possible after recovery procedures begin.

When the system reboots automatically after a crash, the operator may not be present. Therefore, the following lines appear in the standard system_start_up.ec:

```
&if [and [get_flagbox unattended] [get_flagbox rebooted]]
&then sc_command delete device tape_(01 02 03 04 05 06 07 08)
```

so that until the operator explicitly reattaches the tape drives or does an "x attend," no user process hangs waiting for a tape. In order to prevent the system from cycling in a tight loop of boot-crash immediately-recover-boot, the following lines appear:

```
&if [and [get_flagbox unattended] [get_flagbox rebooted]]
&then set_flagbox auto false
```

These command lines turn off automatic reboot mode, thus preventing repeated attempts to reboot without operator intervention. If these lines are omitted from system_start_up.ec, the reboot loop terminates when the BCE dump command finds the DUMP partition full. If the system attempts to reboot itself repeatedly, this may be a sign of some system problem that does not prevent answering service startup, but crashes the system later. If this happens at your site, one solution is to modify system_start_up.ec so it submits a deferred absentee job that enables rebooting if the system stays up for at least an hour.

Note that when the system is running in manual mode and returns to BCE, automatic recovery procedures do not run. However, the operator may explicitly ask for the same procedures to run.

Note also that automatic recovery procedures can't run until the system returns to BCE. So when you have a failure which doesn't crash the system (a loop or a hang), you have to force the system to return to BCE before automatic recovery procedures can begin.

Manual Recovery

If you prefer to recover the system manually, or if automatic recovery procedures fail, you may dump Multics, perform ESD, and reboot the system yourself. Step-by-step procedures for recovering the system manually are available in the *Operator's Guide to Multics*, Order No. GB61.

When to Perform Emergency Shutdown

Emergency shutdown (ESD) should always be performed after a system crash, as long as the hardware (especially memory and disk controllers) is operational, no disk packs have been moved, and the contents of memory have not been disturbed.

DO NOT attempt to perform an ESD if memory has been cleared or powered off, disk controllers are broken, or disk packs have been moved. Also, since an ESD destroys the current memory image, you should only perform one after you've dumped Multics or examined the crash with BCE probe, or after you've decided that dumping Multics is impossible.

Doing ESD from the Switches

If the system is hung, you understand why, and there is no way to get a dump, or if you are in BCE and have lost the bootload console, you may perform an emergency shutdown from the switches. To do this on a Level 68 system, execute switches with the DATA switches set to 024002717200. To do this on a DPS 8 system, use the BCE 24002 command (described in Appendix B).

Recovery Failures

Note: if any one of the automatic recovery procedures fails, you will have to finish it yourself, then perform any remaining procedures manually. The automatic recovery procedures will not restart after you've fixed the one that didn't work; they cease to run automatically once one of them fails.

SYSTEM DOESN'T CRASH

When the system doesn't crash, it doesn't return to BCE. A failure which doesn't crash the system is usually the result of the system looping or the initializer process hanging. When this happens, you must crash the system manually by executing fault (or by executing switches, if executing fault doesn't work). Step-by-step procedures for executing fault and executing switches on both a Level 68 system and a DPS 8 system are available in the *Operator's Guide to Multics*, Order No. GB61. BCE senses this manual intervention and does not perform the automatic operation specified in the flagbox bce_command. You may invoke automatic recovery by typing "ec rtb" or you may recover the system manually.

You will not be able to return to BCE if the system can't do disk I/O to the RPV for any reason (for example, because one or more MPCs are broken).

Sometimes a system crash sounds the IOM alarm. BCE cannot be successfully entered until the IOM alarm is manually reset from the IOM panels. (*DO NOT* use the INITIALIZE button on the bootload console to reset the system after an IOM alarm.) The step-by-step procedure for resetting the IOM alarm is available in the *Operator's Guide to Multics*, Order No. GB61.

DUMP FAILURE

If the system fails while it's taking a dump of Multics, the first thing you should do is try the dump again. (If the dump failed because the previous copy_dump command was not successful or not reached, and if the DUMP partition is still full, * you may save the new dump on any other disk that contains a DUMP partition by using the -drive control argument with the dump command.) This allows the new dump to be taken without losing the old one. To try the dump again, type "ec dump." If this works, continue with an ESD. If it doesn't work, give up and continue with an ESD.

EMERGENCY SHUTDOWN FAILURE

An emergency shutdown can succeed completely, succeed partially, or fail. If it succeeds completely, you will receive the following messages (and possibly others):

```
begin emergency shutdown part 1
emergency shutdown part 1 complete
shutdown complete
```

In this case, continue by rebooting Multics.

If the ESD succeeds partially, you will receive the following messages:

```
begin emergency shutdown part 1
emergency shutdown part 1 complete
```

Then you will receive some combination of the messages described next.

```
shutdown_file_system:  Error deactivating.  Quote may be bad.
```

This message indicates that the system failed trying to update its permanent record of quota. You will only see this message once per crash. If you get this message, run a quota salvage after you reboot Multics.

```
disk_emergency:  dskX_NN inoperative:  shutdown of dskX_NN
   suspended.
```

This message indicates that the disk volume mounted on drive dskX_NN could not be shut down due to disk errors. You will see this message once for each broken drive, each time you try the ESD. If you get this message, move the pack or reset the drive, and then retry the ESD by typing "esd."

```
shutdown_file_system:  from demount_pv on OCTAL_PVTX.
   ERROR_MESSAGE
```

This message indicates that a disk volume could not be shut down. Some previous error message should have reported the disk drive and/or volume name. Normally, if a drive is inoperative, the system will have detected the problem, printed a more informative message, and given up on the drive before this point. Thus, this message should appear very rarely. If you get one or more messages like this without having gotten any messages from disk_emergency or elsewhere, you should make sure that all drives are ready, and retry the ESD. You can use the test_disk BCE command to try to identify the failing drives.

```
shutdown_file_system:  N locks set.
```

This message indicates that some system databases were left inconsistent, even though all disk volumes were shut down. You may see this message once, each time you try the ESD. You should ignore it.

```
shutdown complete except for devices suspended.
```

If you received any of the "disk_emergency" or "shutdown_file_system: from demount_pv" messages described above, this message will be the last one you receive before the system returns to BCE. It replaces the "shutdown complete" message.

As you can see from these messages, what usually happens when the ESD only succeeds partially is that some disk volumes get shut down, but others don't. In this case, you may be able to shut down the rest of the volumes by trying the ESD a number of more times. To do this, keep typing "esd." Then continue by rebooting Multics.

If the ESD fails completely, the system will hang or crash.

When you can't get a successful ESD, you should reboot the system. If you had problems with disks or disk MPCs, or if either the RLV or the RPV were tight for space, you should volume salvage the RLV when you reboot, by doing a "boot rlvs." If you try to reboot without salvaging and fail, try again with salvaging. If you don't salvage, you may want to stop in ring 1 and invoke the salvage_dirs command (described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64).

If you are running the standard system_start_up.ec, the volume scavenger is run automatically to correct volumes with inconsistencies and volumes which weren't shut down. For more information about salvaging and scavenging, refer to "Salvaging" later in this section.

Note: sometimes the system crashes before the storage system has been accessed. If a premature emergency shutdown is attempted, the following message is printed:

```
esd: Storage system not enabled.  esd not performed.
```

and the storage system hierarchy is untouched.

AUTO REBOOT DISABLED

When auto reboot is disabled, automatic recovery procedures run as far as rebooting Multics, then stop. Auto reboot can be disabled for two reasons. The first reason is because the auto_reboot flag is off. The auto_reboot flag may be turned off by the set_flagbox command executed while Multics is running. When this happens, the exec_coms print a message and exit after recovering (i.e., after doing a dump and an esd). In this case, you should just reboot, using ec auto star.

The second reason auto reboot can be disabled is because the system never completed the system_start_up.ec, which means the booting flag is still on. The exec_coms take a dump and do an emergency shutdown, but do not reboot Multics. In this case, if you got to ring 4 before the system crashed, you should do a "boot stan," get into admin mode, and fix the problem. If you crashed before you got to ring 4, you should do a "boot rlvs."

BOOTLOAD FAILURE

If the system fails while it's rebooting Multics, the first thing you should do is run recovery again. If you want the system to perform automatic recovery, type "ec rtb." If you want to recover the system yourself, use the "ec dump," "esd," and "ec auto star" commands.

Clock Problems

If you get the following error message:

```
initializer: bad_dir_ condition signalled
```

you should check the calendar clock. If it's wrong, use the BCE reinitialize command with the -time control argument, fix the clock, and then do a "boot rlvs." (The BCE reinitialize command is described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.)

If you attempt to boot Multics with a clock setting which is obviously bad, the BCE boot command detects the bad value and either refuses to boot or queries you. (See the description of the BCE boot command in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.) You should fix the clock setting and reboot Multics.

If the time zone on the clok card is not one that the system accepts (i.e., it is not listed in the system's table of time zones), BCE crashes and you go to the "bce_crash" state. You should edit the clok card and reboot Multics. (For a list of acceptable time zones, see the description of the clok card in Section 7.)

Root Volume Problems

If you get the message "No space on RPV", the system will probably crash again. This time, when you reboot Multics, do a "boot rlvs." The system will perform various kinds of salvaging, then pause in ring 1. Type "star" and the system will continue rebooting Multics.

If you attempt to boot Multics with a non-root volume where an RLV should be, the system will fail with one of the following messages:

```
init_pvt: no root
init_pvt: dska 3 has no Multics label
init_pvt: no partition hc on root dska 3
```

First, check to see that the correct packs are mounted on the correct drives. (Use the BCE display_disk_label command, described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.) If they aren't, fix them. If they are, check the root card in the config deck. If it's wrong, correct it. If it's right, recover the RPV. Then reboot Multics.

If the system refuses to leave ring one, and you get a message which says the root logical volume is incomplete, the volume registration for the RLV may be damaged. It can get damaged if the system crashes without ESD, or if the system crashes while you are in the middle of changing the logical volume registration for the RLV. To recover from this, the first thing you should do is shut the system down. Then you should make sure that all of the physical volumes of the RLV are listed on the root card and mounted. Next, you should type:

```
boot stan nosc
```

to reach the ring four emergency listener level. Once there, use the volume registration commands to repair the volume registration for the RLV. Then type:

```
hphcs_$shutdown
```

to leave the emergency listener level and return to BCE. Finally, reboot Multics.

If this procedure doesn't resolve the problem, shut the system down normally, do a "boot nolv", and reregister all of your non-root physical volumes with the reregister command in ring one. Then boot to ring four.

Non-Root Volume Problems

When the system is unable to accept one or more disk volumes, the initializer process types a message and inhibits automatic startup. The system waits at ring 1 or ring 4 command level, depending on where it detected the error. The system may be unable to accept the volumes for a number of different reasons: the disk drives they're mounted on may not be ready, the disk packs may have been moved, or the volumes themselves may have been damaged. If the disk drives aren't ready, you

should ready them. If the disk packs have been moved, you should use the add_vol command to tell the system where they are. If the volumes have been damaged, you should use the del_lv command to delete them, then recover the packs by using the volume reloader or BCE restore.

Disk Table Problems

Occasionally, the system may not come to ring 1 command level because of problems with the Disk Table (the record of disk configuration) or with storage system volume registration. When this happens, include the nodt and/or the nolv parameters on the BCE boot command line. (See the BCE boot command description in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64).

FNP Load Problems

If an FNP doesn't load, use the load_mpx command to load it.

HARDWARE PROBLEMS

If hardware problems are the root of a system failure, none of the procedures above will be successful. Hardware problems can be hard to detect. Sometimes, when a module is in trouble, the TROUBLE light on its control panel will light up. But this isn't always the case. If you suspect that one of the modules is having hardware problems, you should notify CSD. Remove the module from the system and reconfigure the system before rebooting it. Refer to Section 11.

SALVAGING

This subsection describes volume and directory salvaging.

The system automatically performs certain recovery operations when a pack is used after not being demounted or shutdown properly. Other recovery operations are performed when damage to directories or physical volumes is detected. These operations are known as volume and directory salvaging, respectively, and are performed automatically. There are occasions, however, when you are required to perform these operations manually. Instructions and reasons for doing so are detailed below.

The volume salvaging program checks a physical volume for inconsistencies. It makes two kinds of checks. It checks the allocation of disk pages and VTOCEs on the volume to be sure that free pages and VTOCEs are listed as free, that used pages and VTOCEs are listed as used, and that no two VTOCEs are using the same page. It also checks the internal consistency of each VTOCE to be sure that no dates are in the future, that the checksum is reasonable, and that various other data are at least plausible. When the volume salvaging program runs offline, at volume acceptance time, it is called the volume salvager. When the volume salvaging program runs online, while the volume is in use, it is called the volume scavenger.

The directory salvaging program checks a directory for internal inconsistencies, and optionally, for inconsistencies between the directory and the VTOCEs of its entries. When the directory salvaging program runs automatically, it is called the online directory salvager. When the directory salvaging program runs automatically during initialization, in response to the "boot rlvs" command, it is called the bootload directory salvager. When the directory salvaging program runs in response to your invocation of either the x repair, salvage_dir, or salvage_dirs command, it is called the demand directory salvager.

Volume Salvaging

The purpose of volume salvaging is to recover free records and free VTOCEs which have been lost to the system. The most common cause of lost records and VTOCEs is a crash without emergency shutdown (ESD), where the volume was mounted at the time of the crash. Another possible cause is hard I/O errors on the device. Volume salvaging reconstructs the map of available addresses and the list of available VTOC entries on a physical volume, and ensures the integrity of the label and the VTOC.

As stated earlier, volume salvaging can be done in one of two ways: with the volume scavenger or with the volume salvager. Although each method has the same effect, using the volume scavenger is the recommended method. The difference between these methods is that the volume scavenger is an online program that operates while the volume is in use, while the volume salvager is an offline program that operates while the volume is being mounted and is not available for use. The volume salvager should be used only to salvage the RLV during initialization when the system will not boot without a salvage (caused, for example, by insufficient space on the RLV).

CRASHES WITHOUT ESD

Due to some types of hardware and software failure, it is not always possible to obtain a completely successful emergency shutdown. When this happens, two kinds of problems result. The first is that various inconsistencies appear in the storage system at the time of the next bootload. These inconsistencies may reflect themselves as damaged or inconsistent directories, lost or damaged segments, and inconsistent record-quota-used totals. The system guarantees that no segment's pages will be listed as free, and that no two segments will claim the same page, even when ESD fails. However, some segments may become orphaned; i.e., not in use, but not listed as free. The second problem is that some volumes do not get shut down.

For each volume, the system maintains a count of volume inconsistencies since the last scavenge. This count is incremented by one as follows:

- Whenever a volume is mounted without having been demounted properly (for example, the first time a volume which was mounted at the time of a crash without ESD is mounted following that crash).

- Whenever an inconsistency is detected in a volume control structure while the system is running. Such an inconsitency indicates that records or VTOCEs may be lost. These lost records and VTOCEs can be recovered only by a volume scavenge.

The system administrator should monitor volume inconsistency counts with either the display_pvte or the display_disk_label command.

After a crash without ESD, any volume except the RLV may be used without any salvaging. The only effect of using a volume without first salvaging it is an apparent reduction in its capacity (due to lost records and VTOCEs). Although files may be damaged as a result of a crash without ESD, this damage cannot be repaired by a volume salvage. So using a volume which was mounted at the time of a crash without ESD and which hasn't been salvaged has no effect on files residing on that volume.

During initialization, the system will offline salvage volumes of the RLV only if there is danger that initialization might not be able to complete without the salvage. The decision to salvage a volume of the RLV is based on the number of free records on the volume and the number of free VTOCEs. Other than automatic offline salvaging of RLV volumes under certain circumstances, the system does not salvage volumes automatically.

When the system is initialized, the standard system_start_up.ec logs in a daemon that automatically scavenges all volumes with inconsistencies and all volumes that weren't shut down.

Following a crash without ESD, the system administrator should monitor the number of records and the number of VTOCEs left on each volume which was mounted at the time of the crash. If volume capacity is in danger of being exhausted, a volume scavenge should be run.

When a volume is mounted or demounted, the count of inconsistencies is examined. If it is non-zero, a message is recorded in the syserr log. This message includes the count of inconsistencies, the number of free records, and the number of free VTOCEs.

REQUESTING A VOLUME SCAVENGE

Scavenging Any In-Use Volume

Any mounted physical volume can be scavenged at any time. Several physical volumes can be scavenged simultaneously, depending on the size of the scavenger's database (see the description of the tbls configuration card in Section 7). For example, to scavenge the RPV while the system is running, type:

```
x scav rpv
```

To reduce the performance degradation while the scavenger is running, type the following instead:

```
x scav rpv -nopt
```

Each of these command lines logs in a daemon process (Scavenger.SysDaemon) to do the scavenge. The start and completion of the scavenge is reported to the console and any damage found on the volume is recorded in the syserr log.

Scavenging All Volumes of a Mounted Logical Volume

To scavenge all physical volumes belonging to the logical volume libraries, type one of the following:

```
x scav -lv libraries
```

or

```
x scav -lv libraries -nopt
```

As explained earlier, the -nopt control argument reduces the performance degradation caused by the scavenger on other users. It also causes the scavenge to take longer.

Scavenging All Volumes With Inconsistencies

Either of the following command lines logs in one or more daemons to scavenge all mounted physical volumes for which records or VTOCEs may be lost:

```
x scav -all -auto
```

or

```
x scav -all -auto -nopt
```

The x scav command is documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

REQUESTING A VOLUME SALVAGE

Salvaging the Root Physical Volume (RPV)

The RPV can be salvaged only during system initialization. A salvage is requested by including the rpvs option to the BCE boot command. This request causes a volume salvage of the RPV and directory salvages of certain critical system directories. For example, to salvage the RPV and start up the system normally, type:

```
boot rpvs star
```

Salvaging All Volumes of the Root Logical Volume (RLV)

Volumes of the RLV can be salvaged only during system initialization. A salvage of all volumes of the RLV is requested by including the rlvs option to the BCE boot command. This request causes a volume salvage of all member volumes of the RLV, including the RPV, and directory salvages of certain critical system directories. For example, to salvage all volumes of the RLV and start up the system normally, type:

```
boot rlvs star
```

Salvaging Non-RLV Volumes During Initialization

Volumes which are not members of the RLV can be salvaged during initialization. First, boot the system to ring 1 command level (that is, star, stan, and mult must not be used as options to the BCE boot command). At ring 1 command level, use the salvage_vol command to request a volume salvage of each volume for which one is needed or for all volumes known to the system during initialization (except for RLV volumes, which can be salvaged only as described earlier). This procedure is shown below:

```
    bce (boot) 0850.2:
!   boot

    Multics MR11.0 - 03/27/82 0853.0 est Sat
    Command:

!   salvage_vol -all

    Volume salvage of dska_18, volume ldd1 of logical vol libraries.

    (A message like this will be sent for each volume as the volume
    is salvaged.  It may be followed by other messages from the
    volume salvager.)

    Command:
!   star
```

Salvaging Non-RLV Volumes While the System is Running

Prior to salvaging any volume, the logical volume to which it belongs must be demounted. Then, the salvage_vol command is used to salvage each physical volume for which a salvage is needed. Following the completion of all volume salvages needed, the logical volume can be mounted. For example, the following sequence requests a salvage of all physical volumes belonging to the logical volume libraries. In this example, logical volume libraries contain two physical volumes: ldd0 (on dska_15) and ldd1 (on dska_18).

```
! dlv libraries

   demount_pv:  Unload dska_15 for storage system.
   demount_pv:  Unload dska_18 for storage system.
   demounted lv libraries
   Ready (User_name)

! salvage_vol ldd0 dska_15

   Volume salvage of dska_15, volume ldd0 of logical vol libraries.
   (May be followed by other messages from the volume salvager.)
   Ready (User_name)

! salvage_vol ldd1 dska_18

   Volume salvage of dska_18, volume ldd1 of logical vol libraries.
   (May be followed by other messages from the volume salvager.)
   Ready (User_name)

! av ldd0 dska_15
   Ready (User_name)

! av ldd1 dska_18
   Ready (User_name)

! alv libraries

   lv libraries mounted
   Ready (User_name)
```

The volume salvager reports damage to segments in the syserr log. In these reports, the volume salvager cannot display the pathnames of the damaged segments; therefore, it places binary messages in the syserr log describing these segments.

You can display the pathnames of the damaged segments by using the print_sys_log command with the -expand control argument. For example:

```
psl -syserr -from <time> -expand -match salvager scavenger
```

The salvage_vol command and the print_sys_log command are both documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

VOLUME SALVAGING MESSAGES

The volume scavenger reports its findings for all volumes to the syserr log. The volume salvager reports its findings to the console for an RLV volume, and to the syserr log for other volumes.

Directory Salvaging

Directory salvaging detects and corrects errors in one or more directories. It has three functions. The first function is to recover disk space by compacting or rebuilding directories. This is a routine maintenance operation.

The second function of directory salvaging is to detect and repair problems with directories. There are three kinds of directory problems: forward connection failures, inconsistencies between an entry in a directory and the VTOCE for that entry, and directory damage. All three kinds of problems can result from disk errors, hardware failures, system crashes, especially crashes without ESD, and crawlouts.

When there is a forward connection failure, the directory entry describes a VTOCE which does not exist on the volume. Directory salvaging can detect connection failures, but it cannot correct them, since it has no way of finding out where the desired VTOCE is for a given directory entry. However, it can delete entries which have connection failures.

When there are inconsistencies between an entry in a directory and the VTOCE for that entry, the VTOCE is there, but some important fields in the VTOCE don't match the entry in the directory. Directory salvaging can correct the directory for all fields except AIM fields. In the case of AIM inconsistencies, it sets the soos (security out of service) switch.

When there is directory damage, the directory is internally inconsistent. Directory salvaging makes the directory consistent, sometimes deleting entries, ACLs, names, or all of the contents to do so.

The third function of directory salvaging is to insure that segment and directory qota values are consistent.

The storage system maintains records-used totals for segments and directories in a data structure implemented in the VTOC entries and AST entries for segments and directories and their superior directories. When the system (or a given volume) cannot be successfully shut down, or when part of the storage system is recovered from a previous time via the BCE restore command, the quota-used figures associated with segments reloaded or not shut down can become inconsistent with the figures in superior directories. This situation becomes apparent to users as inconsistencies reported by the get_quota and get_dir_quota commands. These figures can be corrected while the system is up by a procedure known as quota salvaging. Quota salvaging may be performed on the entire system or on a given subtree of the directory hierarchy. If a successful emergency shutdown cannot be achieved, quota salvaging should be performed for the entire system during the next bootload. If a given volume is restored (via a BCE restore) or not properly shut down, all master directories containing segments on that volume must undergo quota recovery.

ONLINE DIRECTORY SALVAGER

The online directory salvager is invoked automatically when certain software errors or hardware failures occur. It performs directory salvaging upon a single directory while the system is running. The salv config card (documented in Section 7) is used to set the default options for the online directory salvager. The online directory salvager does not salvage quota.

BOOTLOAD DIRECTORY SALVAGER

RPV Directory Salvaging

Whenever a volume salvage is done on the RPV, i.e., whenever you do a boot rpvs or rlvs, the bootload directory salvager is invoked automatically to salvage the root directory and every other directory used in ring 0 initialization (i.e., the >, >dumps and >lv directories). This operation is done in order to make sure that the system can complete bootload and come up to ring 1 command level.

RLV Directory Salvaging

Whenever a volume salvage is done on the RLV, i.e., whenever you do a boot rlvs, the bootload directory salvager is invoked automatically to salvage all level 2 directories, while the system is running in ring 1. This directory salvage is the same as that performed when you execute the salvage_dirs command.

An RLV directory salvage includes a quota salvage of each of the level 2 directories. After this salvage is complete, the system checks to see if there are at least 1000 free records of segments quota in the directories > (the root) and >sc1. If either of these directories has less than 1000 records, the system forces it to have 1000 free records with the equivalent of a set_quota command. When this happens, the system prints the following message:

```
system_startup_:  Forcing quota for <directory> from <old_quota>
   to <new_quota>.  Run a quota salvage.
```

If you see this message, you should insure that you have enough quota allocated in the directory, and then run a quota salvage on it, since forcing quota creates an inconsistency. To run the quota salvage, use "x repair" (described below).

DEMAND DIRECTORY SALVAGER

For routine maintenance of the hierarchy, invoke the demand directory salvager by using the x repair initializer command in ring 4. To recover disk space, which should probably be done on a weekly basis, type:

```
x repair salv > <number_of_processes> -compact
```

It's usually a good idea to find and repair quota inconsistencies when you recover disk space. To do this, type:

```
x repair salvquota > <number_of_processes> -compact
```

To completely reconstruct all system directories, which should probably be done on a monthly basis, type:

```
x repair salv > <number_of_processes> -rebuild
```

It's usually a good idea to find and repair quota inconsistencies when you reconstruct directories. To do this, type:

```
x repair salvquota > <number_of_processes> -rebuild
```

Another good idea is to add the -check_vtoce control argument to the command lines above, to clean up any old problems left over from past system crashes. This control argument is necessary if you want to detect connection failures, but makes the salvage run even slower than it would otherwise.

After a system crash, the way you invoke the directory salvager depends on whether ESD was successful or not. If ESD was successful, you are much more likely to have quota problems than other, more serious, problems. Therefore, you should run a quota salvage immediately after the crash. In other words, type:

```
x repair quota > <number_of_processes>
```

Run a normal salvage sometime later, by typing:

```
x repair salvquota > <number_of_processes> -check_vtoce
```

If ESD was not successful and you didn't do a boot rlvs, use the salvage_dirs initializer command in ring 1 to salvage critical system directories. Then type "mult". If the answering service comes up, run a normal salvage sometime later. In other words, type:

```
x repair salvquota > <number_of_processes> -check_vtoce
```

If the answering service doesn't come up, get into admin mode. Use the salvage_dir Multics command from the initializer process to fix directories with problems, and the fix_quota_used exec_com from the initializer process to fix local quota problems. To use this exec_com, type:

```
ec >t>fix_quota_used <dirname>
```

Finally, if a user complains about problems with a single directory, use the salvage_dir Multics command from a privileged process to salvage that particular directory.

The x repair, salvage_dirs, and salvage_dir commands are all documented in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

Both the online directory salvager and the bootload directory salvager print their error messages on the bootload console and in the syserr log. Where demand directory salvager messages go depends on which command you use to invoke the demand salvager. The x repair command logs in one or more daemons, which print some of their messages on user_output. All of their messages are also printed in dprinted files. Where user_output goes depends on where you route it with the message coordinator route and define commands. The salvage_dirs command prints its messages on the bootload console and in the syserr log. The salvage_dir command puts its messages in a file.

DISK FAILURES

This subsection describes disk failures, including how to recognize a disk failure, how to determine what kind of disk failure it is, how to recover from different kinds of disk failures, and how to be prepared for disk failures. It also includes detailed procedures for recovering from various kinds of disk volume failures.

*

Recognizing a Disk Failure

Often, only one disk volume or disk drive is involved in a disk failure. The symptom that you'll notice is a burst of error messages for the drive, of the form:

```
disk_control: MAJOR_STATUS SUBSTATUS for dskX_NN (iom X chn YY).
rec RRRR, sect SSSS, main AAAA.
detailed status: XX XX XX XX XX XX XX XX
```

and then a message such as:

```
disk_control: dska_04 requires intervention.
```

with an audible alarm. The error messages may repeat periodically as the system attempts to access the drive to determine if it has been made available again. If the system is waiting for the drive to become operational and a user is trying to access data on the drive, messages of the form:

```
pxss:  notify timeout, event = 144163153167,
   processid = NNNNNNNNNNNN
```

are printed on the bootload console. These messages indicate that disk input/output operations are taking longer than normal to complete. If the processid is 003000777777, the initializer process is waiting for the drive, and no initializer or admin commands can be issued until the drive becomes operational. In some cases, the system crashes when a disk drive fails; most often, the system continues operating, although without the availability of segments on the affected device.

Determining the Nature of a Disk Failure

The disk_control error messages which include IOM and device channel numbers, record addresses, and interpretations of error statuses provide many clues about the nature and extent of disk failures. Patterns in the device numbers (e.g., dskX_NN), IOM and channel numbers, and record addresses can indicate whether the failure stems from a malfunctioning IOM, MPC, channel or disk drive.

If all disk error messages refer to a single record address, a single device, and multiple IOMs/channels, there may be only a single bad record on the disk volume. If such is the case, you can prevent further accessing of the bad record by using the record_to_vtocx command to identify the segment containing the bad record. You can then prevent access by turning on the segment's damaged switch, by denying access to the segment, or by deleting the segment's VTOCE with the following command:

```
hp_delete_vtoce pvname vtocx -clear
```

Errors for multiple disk records on a single device, with multiple IOMs/channels, may indicate a disk drive failure. Any disk records written during the failure event (which is possibly ongoing) may be damaged. Another possibility is that read errors are occurring, with disk records remaining undamaged. Moving the disk volume to another drive might correct the latter problem.

Errors on multiple devices which all occur through IOMs/channels leading to a single MPC may indicate an MPC failure. Any disk volume accessible through this MPC may have been damaged during the failure event.

Errors on multiple devices which all occur through a single IOM and channel, with IOM system fault messages of the form:

```
disk_control: Unexpected IOM status SSSS for dskX_NN (iom X
   chn YY).
```

may indicate a malfunctioning IOM Peripheral System Interface Adaptor (PSIA). Any disk volume accessible through this channel may have been damaged during the failure event.

The preceding patterns are symptoms of various types of failure; they are NOT hard-and-fast rules of failure. But analyzing such patterns can be helpful in trying to determine which device or devices are failing, and in isolating such devices from the system.

Recovering from Disk Failures

DELETING THE FAILING IOM, MPC OR CHANNEL

If disk error patterns indicate that a major device such as an IOM or MPC may be malfunctioning, or that a single physical disk channel (IOM PSIA) may be malfunctioning, and if the disk volumes accessible through the failing device are crossbarred, it is often useful to try deleting the failing IOM, MPC or channel from the system. This can help avoid disk error messages, possibly restoring system operation with slightly degraded performance. It can also help to identify the failing unit, since error messages may stop after the IOM, MPC or channel is deleted. Use the reconfigure command to delete possible failing devices. For example:

```
rcf delete iom b -delete_all_attachments
rcf delete mpc mspa
rcf delete chan b23
```

REREADYING THE DISK DRIVE

When a disk intervention message is reported and the pattern of error messages indicates failure of a single disk drive, you should attempt to reready the disk drive by pressing its START/STOP button. If rereadying the drive corrects the problem, the system prints:

```
disk_control:  dska_04 now operational.
```

and attempts to use segments on the drive again. Sometimes a disk drive may reready itself automatically. The system prints out messages and sounds an alarm as long as the drive is not operational.

For MSU0500/501 disk drives, if rereadying does not make the drive operational, have your Customer Service Representative check for tripped circuit breakers inside the back of the drive. You can also try using the ONLINE/OFFLINE button to set the drive offline, then pushing the T&D RESET BUTTON (which is on top of the unit in the lowered area between the two disk spindles), and then putting the unit back online. However, repeated stopping and starting of an MSU0500/501 disk drive can damage the drive's motor and should therefore be avoided.

MOVING THE DISK VOLUME TO ANOTHER DRIVE

If a disk drive with a removable volume (an MSU0451) cannot be made ready and you have an unused disk drive on the same disk subsystem, you can move the volume to this spare drive along with the associated disk drive device number plug.

To move a disk volume from a failing drive to a spare drive, put the failing drive in standby mode by pushing the STOP button. If there is a disk volume on the spare drive, put that drive in standby mode as well. Remove the device number plugs from both the failing drive and the spare drive immediately. When the spindles stop, demount any volume from the spare drive, and move the volume from the failing drive to the spare drive. Push the START button to start the spare drive. After the spare drive becomes ready, insert the device number plug from the failing drive into the spare drive; insert the device number plug from the spare drive into the failing drive.

Note that the spare drive must already be ready before you insert the device number plug. Note also that you must wait at least 30 seconds from the time you remove the plugs until the time you reinsert the plugs. This is because the MPC only polls devices every 15 seconds for status changes.

If the procedure above fails, the following procedure may be tried as an alternate. Put the failing drive in offline mode by setting the rotary switch on the inside of the back door of the unit. Then power down the failing drive, dismount the disk volume, mount it on the spare drive, and swap the device number plugs between the failing drive and the spare drive. Set the spare drive in offline mode, ready it, and then put it in online mode.

Note that a similar procedure exists for moving fixed-head disk volumes (MSU0500/501s) to another disk drive on the same disk subsystem, but this procedure can only be carried out by your Customer Service Representative. The procedure involves dismounting the head assembly (HDA) from the failing disk drive and from a spare drive on the same subsystem, swapping the head assemblies, and swapping the "Berg block" identifiers on the two disk units. Both disk drives should be placed offline during this process. This procedure may take up to an hour, but at times, the system can continue running with acceptable performance while such a move is undertaken. Consult with your Customer Service Representative for further information.

RELOADING DISK MPC FIRMWARE

If the system still reports the drive as not operational, either after moving the disk volume to another drive, or after attempting to reready the failing drive, it may be necessary to reload the firmware in the MPC for the disk subsystem. Reloading firmware may also be necessary if the pattern of disk error messages indicates an MPC failure.

If the failing disk subsystem is serviced by two MPCs, then you will have to reload the firmware in both of the MPCs. This can be done most easily by using the Multics load_mpc command. For example, if the dska subsystem is serviced by MPCs mspa and mpsb, send the following commands to Utility.SysDaemon:

```
r ut load_mpc mspa -firmware
r ut load_mpc mspb -firmware
```

Wait for the first command to complete before sending the second command.

Note that it is best to run the load_mpc command from some other process besides the initializer process. A failure of this command could cause the initializer process to hang or terminate, crashing the system. However, some sites may have secured access to other SysDaemon processes (e.g., Utility.SysDaemon). At such sites, you can only load the firmware by going into admin mode in the initializer process:

```
admin
load_mpc mspa -firmware
load_mpc mspb -firmware
ame
```

Also, note that load_mpc can only be used to reload firmware when there are two MPCs serving a given disk subsystem. This is because reloading firmware takes the MPC out of service, and the system must have an alternate path to access the disk subsystem via the second MPC. If a noncrossbarred disk MPC fails to respond, you can return to BCE to reload its firmware. When BCE is reentered, it automatically polls each MPC to make sure it is operational. If an MPC fails to respond, BCE asks whether to reload its firmware, as shown below:

```
! bce
  load_disk_mpcs:  Disk mpcs mspa appear not to be operating.  Enter
  disk mpc names to be loaded, or "none" or "abort" or "all":
! mspa
  bce (crash) 1601:3
! go
```

Of course, if the disk string containing the RPV is serviced by only one MPC and that MPC is failing, you can't return to BCE to reload the MPC firmware. Returning to BCE accesses the bce partition on the RPV to read the BCE memory image. Failure of the MPC servicing the RPV makes this read operation impossible.

If so many error messages are being printed on the bootload console that the system will not respond to operator requests, you can reinitialize the firmware in the disk MPC rather than reloading it, by using the procedure described in the *Operators' Guide to Multics* (Order No. GB61). This procedure takes the entire system out of service for a short period of time, and should only be performed when disk failures are severe.

SHUTTING DOWN OR CRASHING THE SYSTEM

If you cannot make the failing disk drive operational, the system will continue printing error messages. The system can often continue to run with such errors for an extended period of time. If it appears to be running properly except for the disk errors, attempt to contact your Customer Service Representative for help in correcting the disk drive problem. He may be able to correct the problem which prevents making the drive ready, without having to shut down the system.

If many errors are occurring and system performance is seriously degraded, or if the initializer process is affected by the disk failure, it may be necessary to shut down or crash the system. In this case, you should attempt to shut down the system normally. If the system won't respond to the shutdown command, you should follow the procedure for dealing with failures that do not crash the system in the *Operators' Guide to Multics* (Order No. GB61). This procedure includes doing an emergency shutdown of Multics. For more information on system crashes, refer to "Crashing" earlier in this section.

During regular or emergency shutdown, the system attempts to shut down as many disk volumes as possible. If a volume cannot be shut down because of a hardware problem with its disk drive, the system prints the message:

```
disk_emergency:  dska_XX inoperative:  shutdown of dska_XX
   suspended.
```

and continues to try to shut down all other drives. When emergency shutdown completes, the system prints:

```
Shutdown complete except for drives suspended.
```

instead of the normal "shutdown complete" message. Once the system has shut down, you or your Customer Service Representative should try to make the disk drive operational. Then you can issue (or reissue) the esd command to reattempt shutdown on the suspended disk drive.

Sometimes a disk drive can fail in such a way that ESD will not complete for other, operational disk drives. If this happens, try powering off the failing disk drive and reissuing the esd command. The power off status from the drive will force the drive into an inoperative state which the esd command will recognize. The esd command will suspend shutdown of the disk volume and move on to other volumes.

You can issue the esd command as many times as necessary until the shutdown completes normally, or until you decide that you cannot make failing drives operational. Any disk volumes which remain suspended following ESD may contain segments which are damaged. This means that you must perform record quota recovery operations (via an "x repair salvquota"). The RPV will be salvaged automatically during the next Multics bootload.

If a disk drive with a removable volume cannot be made operational, the disk volume can be moved to any other spare disk drive for the next Multics bootload. However, you should be sure that the disk volume has not been damaged by a head crash. Moving a damaged disk volume to another drive can cause a head crash on the second drive. Have your Customer Service Representative check the disk volume for damage. If your CSR is not available, check for fine dust particles on the walls of the disk housing where the disk is mounted. Such dust particles usually indicate that a head crash has occurred.

When a disk volume is moved to another drive with a different device number, you should boot Multics to ring 1 command level and use the add_vol command to tell the system the new location of the disk volume. If the RPV is moved to a drive with a different device number, you must reboot BCE so it will be able to find the new location of the RPV. The part config cards referring to partitions on the RPV must be updated to reflect the RPV's new device address. If an RLV volume containing a hardcore partition is moved, you must update the root config card to reflect the new location of the disk volume.

Disk Volume Failures

A disk volume failure happens when the data written on a disk volume has become unreadable or damaged. Disk volume failures can result from hardware failure of the disk drive on which the volume is mounted, hardware failure of the IOM or MPC through which the disk drive is connected to the system, failure of a CPU, memory or SCU, or failure of the Multics software which manages disk input/output.

DEGREES OF DISK VOLUME FAILURE

One type of disk drive hardware failure is a head crash, in which the sensors which read/write magnetic signals on a disk track come in contact with the disk. This causes damage to the disk heads and to the disk volume. Another type of failure is a marginal track condition, in which the magnetic signal written on a disk track is too weak to be read, either because the signal was weak when it was being written, the recording media on the track is damaged, or the read sensors and electronics are improperly adjusted. MPC failures include failure of the hardware circuit boards in the MPC or failure of MPC firmware logic.

Disk volume failures may be transient or permanent in degree. In a transient failure, read operations on a few disk records may produce errors, but attempts to reread those records may succeed. In a permanent failure, attempted read operations always fail.

EXTENT OF DISK VOLUME FAILURE

Disk volume failures may be either partial or total in extent. In a partial failure, most records on the disk can be read or rewritten correctly, but a few records are unreadable (i.e., read errors occur). In a total failure, most or all records on the volume are unreadable.

To determine the extent of failure, you can use the BCE test_disk command to read the entire volume and report records which are unreadable:

```
! test_disk r dskc_16
  <several disk_control error messages>
  test_disk: Could not read record 47261 on dskc_16.
```

The test_disk command normally reads three records at a time, starting from the inside of the volume and working outward. When test_disk encounters an error, it reverts to single record mode and retries the I/O to locate the failing record. If an error is encountered in single record mode, an error message is displayed and the record is skipped. You can also manually attempt to reread a failing record reported in one of the error messages with the following command:

```
test_disk r dskc_16 -record 47261
```

RECOVERING FROM TRANSIENT DISK VOLUME FAILURE

It is often possible to recover from transient errors by moving a removable disk volume to another drive. Another way of recovering is by using the BCE test_disk command to read and rewrite failing records:

```
! test_disk rw dskc_16 -record 47261
```

If the read operation succeeds, then the rewrite operation usually rewrites the data with a stronger signal. No further recovery steps are needed.

RECOVERING FROM PERMANENT DISK VOLUME FAILURE

If the read operation described above fails after repeated attempts, the failure is permanent, and you must follow the procedures for recovering from partial disk volume failure (described next) or the procedures for recovering from total disk volume failure (described later), depending on the extent of the failure.

RECOVERING FROM PARTIAL DISK VOLUME FAILURE

If only a few records on a disk volume are damaged, you can copy the remaining data from the damaged volume onto another disk volume using the BCE copy_disk command. This method of recovery is much faster than the methods for recovering from total failure described below. However, using BCE copy_disk may not work, depending on what part of the volume is damaged. Errors in the disk label region of the disk (records 0. to 7.) make the disk unusable. Errors in the VTOC regions can cause the loss of up to five VTOCEs for each unreadable disk record. Errors in the paging region affect only the segment which owns the disk record.

If you decide to try to copy the damaged disk volume, the following example shows how to copy the records in the paging region (including the VTOCEs) and any partitions on the root2 volume of dskc_16 onto another disk volume mounted on dskb_03:

```
! copy_disk dskc_16 dskb_03
```

Once the disk volume has been copied, you should use the BCE test_disk command to zero any records that were not successfully copied onto the new volume:

```
! test_disk w dskb_03 -record 47261
```

Disk records that are zeroed should be traced to see what VTOCEs or segments have been damaged. If the disk record is in the VTOCE region, then up to five VTOCES (segments) have been lost. A complete hierarchy salvage operation should be performed to look for connection failures (directory entries that have no corresponding VTOCE). Refer to "Hierarchy Salvaging" below. Also, the physical volume should be volume-salvaged if VTOCEs were lost, to recover use of the VTOCEs for new segments. Refer to "Volume Salvaging" below. If a zeroed disk record is in the paging region of the disk volume, you can use the record_to_vtocx command to determine which segment contains the zeroed disk record. This segment can then be retrieved.

RECOVERING FROM TOTAL DISK VOLUME FAILURE

There are three strategies for recovering from a total disk volume failure: volume reloading; a BCE restore followed by volume reloading; a BCE restore followed by hierarchy reloading. All three strategies involve replacing physically damaged disk volume or volumes with spare volumes, and reloading the contents of the volumes from backup tapes. Only the volume reloading strategy is recommended for recovering an entire disk volume or set of volumes. Refer to Section 9 for a discussion of volume and hierarchy backup facilities.

Volume Reloading and BCE Restore/Volume Reloading

The volume reloading and BCE restore/volume reloading strategies can be used to recover the Root Physical Volume (RPV), volumes of the Root Logical Volume (RLV), and non-root volumes.

Recovery via volume reloading involves initializing a spare disk volume, and reloading complete, consolidated, and incremental dump tapes produced by the volume dumper in reverse chronological order. (The most recently dumped tapes are loaded first.) The -pvname control argument of the reload_volume command specifies which volume or volumes are to be reloaded. This is the recommended strategy.

Recovery via a BCE restore followed by volume reloading involves replacing the damaged disk volume with a spare volume, restoring the most recent BCE save tapes for the damaged volume using the BCE restore command, and then reloading the consolidated and incremental volume dumper tapes created after the BCE save operation was performed. The -save control argument of the reload_volume command indicates that the date-contents-modified field of each entry being reloaded should be compared with the date-unmounted field of the volume label. Since a volume must be unmounted before a BCE save operation can be performed, the date-unmounted value placed in the volume label by the BCE restore operation is a good indicator of the date on which the BCE save operation was performed. If the entry from the volume backup tape is newer than the date-unmounted field from the disk label, then the tape entry is reloaded.

We recommend use of the volume reloading strategy for disk volume recovery. The disk recovery procedures described later in this section employ this strategy. Circumstances almost never warrant use of the BCE restore/volume reloading strategy. Therefore, to avoid confusion, this alternate strategy is documented in Appendix H.

Different procedures must be used when doing volume reloading for RPV, RLV, and non-root volumes. The difference stems from the amount of the hierarchy which is available for use during volume reload operations. When the RPV is being reloaded, the hierarchy isn't available at all. A separate test Multics system must be used during the RPV reload operation. (Refer to "Test System" later in this section.) When other RLV volumes besides the RPV are being reloaded, you can use the part of the hierarchy on the RPV to perform the reloading. When a non-root volume is being reloaded, you can use the space on the entire RLV or on other mounted physical volumes to perform the reloading. See "Disk Volume Recovery Procedures" below for descriptions of how to reload each type of volume (RPV, RLV or non-root) using the volume reloading disk recovery strategy.

BCE Restore/Hierarchy Reloading

The BCE restore/hierarchy reloading strategy can be used to reload a volume which is not part of the Root Logical Volume (single volume reload), to reload the entire Root Logical Volume (RLV reload), or to reload the entire hierarchy (complete reload). BCE restore/hierarchy reloading cannot be used to recover only a single root volume (either the RPV or an RLV volume). A complete or RLV reload must be performed to recover single RLV volumes.

The hierarchy backup facility is useful primarily for retrieving or reloading entire subtrees of the hierarchy. It is not recommended for reloading one or more physical volumes. Therefore, procedures for performing BCE restore/hierarchy reload operations are described in Appendix H.

AFTER DISK RECOVERY SUCCEEDS

Recovery from partial or total disk failure should be followed by several additional steps. After recovering the RPV or other RLV volumes, you may want to copy information from the LOG, DUMP, CONF or FILE partitions of the damaged volume onto the reloaded disk volume. After recovery succeeds on any volume, volume salvaging, hierarchy salvaging, and reverse connection failure garbage collection should be performed to make the storage system consistent again. See "Disk Volume Post-Recovery Procedures" later in this section for descriptions of how to perform these follow up operations.

Preparing for Disk Volume Failure

Each site should take steps in advance to prepare for a disk volume failure. Should a disk failure occur, these advance steps will speed recovery and minimize system and logical volume unavailability.

DISK VOLUME LAYOUT INFORMATION

One of the most important steps to take in preparing for disk failure is to have hardcopy listings of the disk volume layout available for each physical volume. This information can be printed using the command:

```
display_disk_label pvname -long
```

The output of this command includes the physical and logical volume unique ids, the volume serial number, the location and size of each partition, and the number of VTOCEs. This information is often needed to initialize a spare disk volume during hierarchy or volume reloads. Hardcopy listings of this information should be produced for each physical volume on a regular basis. Perhaps the easiest way to accomplish this is via an absentee job that runs periodically.

BACKUP TAPE LOGS

Another vital step to take in preparing for disk failure is to have your operators maintain an accurate log of BCE save tapes, hierarchy dump tapes, and volume dump tapes written during backup procedures. Tape volume names should be recorded in a common log, along with the type of dumping operation. When disk volume recovery is needed, you will have to know which tapes contain the data to be restored or reloaded. The tape log provides that information.

For a BCE save, the tape volume name consists of two parts: the tape set name (e.g., blue, root, June) and the reel number (i.e., 1-9999). A BCE save tape set is a collection of reels numbered from 1 to N and a locator tape called the "Info" tape. The "Info" tape contains the names of all the volumes and partitions that were saved and the corresponding tape reels that contain this information. The "Info" tape is always the first tape read during a BCE restore, to allow for program control over tape mounts. The backup tape log should identify the tapes used for each set and the physical volumes saved in each set.

For a hierarchy reload, the log should identify the tapes included in each incremental, catchup and complete dump set.

The volume reloader maintains its own, online tape logs automatically. However, in order to reload these online logs, you must record the tape volume last used for volume dumping (incremental, consolidated or complete) for each volume backup group. Therefore, your operator should record the volume names of volume backup tapes in the tape log as these tapes are written.

OFFSITE COPIES OF BACKUP DATA

A prime method of guarding against a major disaster at your site (e.g., fire in the machine room) is to periodically make complete backup tapes of all data on the system, and to ship these tapes to an offsite storage location. Along with the tapes, you should send hardcopies of pack layout information, tape logs for the offsite tapes, and other information which would be required to restore the data on your system.

DISK DRIVE RECONFIGURATION PLAN

Another step to take in preparing for disk failure is to plan how you would reconfigure each of your disk subsystems to handle a disk drive failure.

One reconfiguration technique is to have a spare disk drive in each subsystem. If a disk drive with a removable volume fails, its volume can be moved to the spare disk drive without interrupting system operation. The procedure for moving a disk volume is described above.

If you don't have a spare disk drive in each disk subsystem, having one spare drive is still helpful. When a removable disk volume fails in a disk subsystem which does not have a spare drive, you can shut down the system and move the disk volume mounted on the failing drive to the spare drive in another subsystem. When booting Multics, boot to ring 1 command level and use the add_vol command to specify the new location of the moved disk volume. If the volume is the RPV or if it contains LOG or DUMP partitions, you will also have to change the part cards in the config deck. If the volume is an RLV with a hardcore partition, you will have to change the root config card.

For nonremovable disk volumes, having a spare disk drive is still a good idea. It need not be in the same disk subsystem as the failing drive. Information from a failing disk volume can be reloaded onto the spare disk drive and its disk volume. Also, in some instances, your Customer Services Representative can move a disk volume head assembly (HDA) from a failing disk unit to a spare unit by swapping HDAs, as described earlier in this section. During normal operation, the spare disk drive can be used for process directories or other temporary data files, or the spare drive can be used for a test Multics system, which has a separate hierarchy from the production Multics system.

If there are no spare disk drives in your configuration, your system should be divided into two or more logical volumes. You should place data on these logical volumes in such a way that the system could run properly without one of the logical volumes being mounted. Then if a disk drive fails for an extended period of time, you can still run the system without the failing drive by demounting the nonessential logical volume and moving data from the failing drive onto a drive used by the demounted logical volume. One technique for doing this is to define a separate logical volume to hold the information under the >ldd hierarchy, plus process directories. The >ldd directory can be created as a master directory before its contents are loaded from Multics release tapes. If a disk drive fails, you can then dismount the >ldd logical volume and use its disk drive as a spare when reloading information from the failing disk volume. Refer to the *Multics System Administration Procedures* manual (Order No. AK50) for a description of how to create master directories. If you define a separate logical volume for >ldd, remember to create a separate master directory for >ldd>include. The include files in this directory are needed during normal system operations for program development.

PREFORMATTED DISK VOLUMES

One of the most time consuming steps in recovering from a disk volume failure is formatting and testing of a spare disk volume prior to reloading data from the failing disk volume. This step can be avoided during failure recovery by formatting and testing spare disk volumes in advance.

Both removable disk volumes (MSU0402 and MSU0451) and nonremovable disk volumes (MSU0500 and MSU0501) should be formatted and tested using the online T&D tool called MTR, available under TOLTS/MOLTS. Procedures for formatting and testing disk volumes are described in the *Multics Online Test and Diagnostics Reference Manual,* Order No. AU77.

Note: the recovery procedures described later in this section assume that you have spare disk volumes already formatted and tested.

TEST SYSTEM

It is sometimes necessary to use a small, one-volume Multics system during disk recovery operations. This "test" system has its own Multics hierarchy, usually on a single disk volume. This test hierarchy is totally separate from the Multics hierarchy on your "production" Multics system.

If your site has removable disk volumes, then the test system can be placed on a disk volume which is not normally mounted on a disk drive during production operation. If your site has only nonremovable disk volumes, then your one-volume system can be stored on BCE save tapes during normal system operation, and restored onto a spare disk volume (one used for process directories during normal system operation) when you need to run the test Multics system.

A test Multics system is a tool which serves a variety of functions. Besides being needed for recovery of a damaged RPV, the test Multics system can be used to check out software changes to the operating system, hardware reconfiguration changes, and correctness of repaired hardware after the Customer Service Representative declares it to be operational.

In order to serve all these functions, the test system may need to operate under a variety of system configurations. It should be able to operate in the hardware configuration used for the production system. But you may want it to run on smaller hardware configurations as well, especially if your hardware is fully redundant and can be reconfigured into two separate systems. Be sure to maintain accurate configuration decks on the test system for all of its possible operating configurations.

To initially create the one-volume test system, follow the Installation Instructions for a new site which are shipped with each Multics release. Be sure that the disk volume has been formatted and tested using MTR. Once the test system has been created, it can be upgraded for each Multics release in parallel with (or prior to) upgrading your production system.

Because the test system will be run during periods of hardware failure, it is important to have an adequate backup for the one-volume system. Use of BCE save/restore is recommended, because that is the fastest method of backup and reload. You should make two sets of BCE save tapes for the test system, to offset the effects of possible tape errors. And you should have offsite copies of your test system's BCE save tapes, disk volume layout information, etc.

Disk Volume Recovery Procedures

The following step-by-step procedures describe how to recover from disk volume failures on each type of volume (RPV, RLV, and non-root) using the volume reloading disk recovery strategy. The procedures include determining the extent of damage and recovering from transient or partial failures, as well as recovering from total disk volume failure. If you need to use one of the other disk recovery strategies (BCE restore/volume reloading, hierarchy reloading, or BCE restore/hierarchy reloading), refer to Appendix H.

As stated earlier, the procedures which follow assume that you have spare disk volumes already formatted and tested. If you do not have any formatted, tested disk volumes available when a disk failure occurs, there are several procedures you can use to obtain a formatted disk volume. One of the simplest is to dump the contents of a physical volume not needed for system operation onto tape, using the BCE save command. Then you can initialize this volume and reload data from the failing disk volume onto it.

If you have a small (one- or two-volume) test system available, you can boot that test system, and use online T&D MTR to properly format and test a new volume. Or, if the failing disk volume is not part of the RLV, you can boot the system using only the RLV, and use MTR to format and test a volume.

If you do not have a test system available, and one of the volumes of the RLV has been damaged, then you should follow the procedure described earlier for creating a one-volume Multics system on which you can invoke online T&D MTR to format and test a new volume.

RECOVERY OF THE RPV WITH VOLUME RELOADING

If a disk volume failure occurs for the RPV, the following procedure can be used to recover the contents of the RPV from volume backup tapes. See Section 9 for general information and more details on volume backup and volume reloading. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described above under "Recovering from Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering from Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the original RPV volume, or to recover its data onto a spare disk volume, you * will need to boot BCE and Multics on a temporary RPV. This temporary RPV may be obtained in any of the following ways:

- If your site has prepared a one- or two-volume "test system" for hardware and software checkout purposes, you can boot this test system for use in testing and reloading the original RPV.

- If you have BCE save tapes for the original RPV, and a spare disk volume, you can restore these save tapes onto the spare disk volume for use as the temporary RPV. The actual data on the temporary RPV is not important since it will not become part of the production hierarchy; an older set of save tapes can be used, as long as the saved RPV is for the Multics release you are currently running.

  You will have to boot BCE on the temporary RPV, and specify "cold" to the "Enter rpv data:" prompt to allow the temporary RPV to be properly initialized. After restoring the RPV, remember to update the root and part configuration cards to describe only the temporary RPV.

- If you have neither a "test system" nor save tapes for an RPV, you can perform a cold boot of Multics on a spare disk volume to create the temporary RPV. To perform the cold boot, follow the procedures in the *Installation Instructions* for the release you are running.

Spare disk volumes should be properly formatted and tested as described above under "Preformatted Disk Volumes."

3. Boot BCE on the temporary RPV, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original RPV disk volume, attempt to read it using the BCE test_disk command, as described above under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original RPV, follow the procedures above under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original RPV is only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures above under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload RPV information from volume backup tapes onto a spare disk volume. These steps assume that the original RPV volume is totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that the original RPV is physically damaged (i.e., scratched or warped), then replace the RPV with a spare volume which has already been formatted and tested, as described above under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original RPV.

* 7. Boot Multics on the temporary RPV, coming up to Multics ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61.

8. Mount the disk volume to be reloaded on any available drive. If necessary, convert the drive to a storage system drive, using the set_drive_usage command. For example:

```
sdu dska_04 ss
```

9. Issue an init_vol command with the -copy control argument. Issue directions to init_vol to define the number of VTOC entries and the partition names and sizes as they were on the destroyed disk volume. Your site should have hardcopy printouts of this disk label information available at all times, as described above under "Disk Volume Layout Information."

   Note that you may request more VTOC entries on the volume being reloaded than were on the destroyed RPV, but you cannot decrease this number. You may increase or decrease the sizes of partitions on the new RPV, or add or delete partitions. However, if you do change the partition layout, then you will not be able to copy the contents of partitions (such as the LOG and DUMP partitions) from the damaged RPV onto the reloaded RPV. Remember to include an alternate track partition for a removable disk volume, if the disk volume being reloaded has been formatted with alternate track assignments.

10. Convert the disk drive on which the new RPV is mounted to an I/O drive, using the set_drive_usage command. For example:

```
sdu dska_04 io
```

11. Recover the volume log for the RPV using the recover_volume_log command with the -wd control argument. For example:

```
recover_volume_log rpv -wd
```

    Mount the last volume backup tape for the volume backup group which includes the RPV. The volume name of the last tape should be recorded in the tape log, as described above under "Backup Tape Logs." If volume backup operations were ongoing at the time of disk failure, you should mount the tape which was being written at the time of failure.

12. Reload the new RPV using the volume reloader, by issuing the reload_volume command with the -pvname, -operator, and -wd control arguments. For example:

```
reload_volume -pvname rpv -operator Jones -wd
```

    Mount tapes as requested by the reload_volume command. When all tapes have been reloaded, continue with the next step.

13. Shutdown Multics on the temporary RPV. *

14. If the RPV was reloaded onto a spare volume and the original RPV is partially readable, you may want to try to copy the contents of the CONF, FILE, | DUMP and LOG partitions onto the new RPV, as described below under "Recovery of Partitions after RLV Volume Recovery". *

15. If the newly reloaded RPV is not mounted on the proper disk drive for normal operation, move the new RPV to the proper disk drive.

16. Boot BCE on the newly reloaded RPV, according to normal site procedures. If reloading was performed on a spare disk volume rather than on the original and the contents of the CONF and FILE partitions was not copied in step 14, then the contents of these partitions must be recreated from files on the Multics system tape or retyped at the operator's console. The CONF partition can be reloaded from a BCE file by typing the command:

```
config bce_file_name
w
q
```

17. Boot Multics according to normal site procedures.

18. Perform the procedures for salvaging, quota adjustment, and connection failure detection described below under "Disk Volume Post-Recovery Procedures." This completes recovery of the RPV.

RECOVERY OF A NON-RPV ROOT VOLUME WITH VOLUME RELOADING

If a disk volume failure occurs on a volume which is part of the Root Logical Volume (RLV) but is not the RPV, the following procedure can be used to recover the contents of that volume from volume backup tapes. See Section 9 for general information and more details on volume backup and volume reloading. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described above under "Recovering from Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering from Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the original root volume, or to recover its data onto a spare disk volume, you will need to boot BCE and Multics on the RPV.

3. Boot BCE on the RPV, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original root disk volume, attempt to read it using the BCE test_disk command, as described above under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original root volume, follow the procedures described above under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original root volume is only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures above under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload root volume information from volume backup tapes onto a spare disk volume. These steps assume that the original root volume is totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that the original root volume is physically damaged (i.e., scratched or warped), then replace it with a spare volume which has already been formatted and tested, as described above under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original root volume.

7. Remove all disk volumes from the root config card, except for the RPV. If any part config cards identify the damaged disk volume, remove those part cards from the config deck. *

8. Boot Multics on the RPV, coming up to Multics ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61.

9. Mount the disk volume to be reloaded on any available drive. If necessary, convert the drive to a storage system drive, using the set_drive_usage command. For example:

```
sdu dska_05 ss
```

10. Issue an init_vol command with the -special control argument. Issue directions to init_vol to define the number of VTOC entries and the partition names and sizes as they were on the destroyed disk volume. Your site should have hardcopy printouts of this disk label information available at all times, as described above under "Disk Volume Layout Information."

    Note that you may request more VTOC entries on the volume being reloaded than were on the damaged root volume, but you cannot decrease this number. You may increase or decrease the sizes of partitions on the new root volume, or add or delete partitions. Remember to include an alternate track partition * for a removable disk volume, if the disk volume being reloaded has been formatted with alternate track assignments.

11. Convert the disk drive on which the new root volume is mounted to an I/O drive, using the set_drive_usage command. For example:

```
sdu dska_05 io
```

12. Recover the volume log for the root volume using the recover_volume_log command with the -wd control argument. For example:

```
recover_volume_log root2 -wd
```

    Mount the last volume backup tape for the volume backup group which includes the RLV. The volume name of the last tape should be recorded in the tape log, as described above under "Backup Tape Logs." If volume backup operations were ongoing at the time of disk failure, you should mount the tape which was being written at the time of failure.

13. Reload the new root volume using the volume reloader, by issuing the reload_volume command with the -pvname, -operator, and -wd control arguments. For example:

```
reload_volume -pvname root2 -operator Jones -wd
```

Mount tapes as requested by the reload_volume command. When all tapes have been reloaded, continue with the next step.

14. Shutdown the Multics running on the RPV.

15. Restore the root and part config cards to their normal values, either by retyping the changed cards or by issuing the BCE "config <deckname>" command to load a new copy of the config deck from a BCE file.

* 16. If the root volume was reloaded onto a spare volume and the original volume is partially readable, you may want to try to copy the contents of the DUMP and LOG partitions onto the new RPV, if these partitions were on the damaged root volume. Follow the procedure described below under "Recovery
* of Partitions after RLV Volume Recovery."

17. If the newly reloaded root volume is not mounted on the proper disk drive for normal operation, move the volume to the proper disk drive.

* 18. Boot BCE on the RPV, according to normal site procedures. Make adjustments to the configuration file as necessary, to reflect the current hardware configuration and disk volume locations.

19. Boot Multics according to normal site procedures.

20. Perform the procedures for salvaging, quota adjustment, and connection failure detection described below under "Disk Volume Post-Recovery Procedures." This completes recovery of the root volume.

RECOVERY OF A NON-ROOT VOLUME WITH VOLUME RELOADING

If a disk volume failure occurs on a volume which is not part of the Root Logical Volume (RLV), the following procedure can be used to recover the contents of that volume from volume backup tapes. See Section 9 for general information and more details on volume backup and volume reloading. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described above under "Recovering from Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering from Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the original volume, or to recover its data onto a spare disk volume, you will need to boot BCE and Multics on the RLV. *

3. Boot BCE as described in the *Operators' Guide to Multics,* Order No. GB61. |

4. If your Customer Service Representative believes there has been no physical damage to the original disk volume, attempt to read it using the BCE test_disk | command, as described above under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original volume, follow the procedures above under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original volume is only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described above under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload information from volume backup tapes onto a spare disk volume. These steps assume that the original volume is totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that the original volume is physically damaged (i.e., scratched or warped), then replace it with a spare volume which has already been formatted and tested, as described above under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original disk volume.

7. Boot Multics on the RLV, coming up to Multics ring 1 command level, as * described in the *Operators' Guide to Multics,* Order No. GB61.

8. To complete the boot, delete the logical volume which contains the damaged physical volume, using the del_lv command. For example:

```
del_lv Xpublic
```

Note: each spindle of an MSU0500/501 disk drive holds two physical volumes, accessed through a single disk arm. It is not possible to have one of these physical volumes accessed as part of the storage system (via disk_control) while the other is used for user I/O operations (via rdisk_ or exercise_disk, which use ioi_). This restriction exists because disk_control and ioi_ do not communicate with one another about sharing the single disk arm which accesses the two physical volumes.

To satisfy the requirements of this restriction, the pair of physical volumes of a spindle are usually configured in the same manner, with both being part of the same logical volume or both being treated as I/O disks, etc.. Because reload_volume writes the volume being reloaded using user I/O (via rdisk_), reloading one physical volume of an MSU0500/501 spindle requires that the other physical volume be deleted from the storage system as well. If both physical volumes are part of the same logical volume, this is not a problem, since deleting the entire logical volume deletes both physical volumes. However, if each physical volume is a member of a separate logical volume, then both affected logical volumes must be deleted from the system during the reload_volume operation.

Note that there is no problem with volume reloading both physical volumes of a spindle at the same time, even if they are members of separate logical volumes. The system knows how to perform user I/O operations on both physical volumes at once, or storage system operations on both at once. Problems arise only when attempting to perform user I/O operations on one physical volume and storage system operations on the other.

9. Issue the standard command to move to ring 4:

```
standard
```

10. If the system can run reasonably without the deleted logical volume, warn users (via a message_of_the_day, or with a login warning set by the word command) that the logical volume has been deleted for repair operations. For example:

```
word login Xpublic volume is offline for repairs.
```

If the system cannot run reasonably without the deleted logical volume, put the system into a special session, using the multics and go commands. This will prevent users from logging in:

```
multics
go
```

11. Mount the disk volume to be reloaded on any available drive. If necessary, convert the drive to a storage system drive, using the set_drive_usage command. For example:

```
sdu dska_06 ss
```

12. Issue an init_vol command with the -special control argument. Issue directions to init_vol to define the number of VTOC entries and the partition names and sizes as they were on the destroyed disk volume. Your site should have hardcopy printouts of this disk label information available at all times, as described above under "Disk Volume Layout Information."

Note that you may request more VTOC entries on the volume being reloaded than were on the damaged volume, but you cannot decrease this number. Remember to include an alternate track partition for a removable disk volume, if the disk volume being reloaded has been formatted with alternate track assignments.

13. Convert the disk drive on which the new volume is mounted to an I/O drive, using the set_drive_usage command. For example:

```
sdu dska_06 io
```

14. Login the volume reloader and issue a display_volume_log command to insure that the volume log for the disk to be reloaded is undamaged and up-to-date. For example:

```
login Volume_Reloader.Daemon vrld
r vrld display_volume_log xpub02
```

If it appears undamaged (no errors occur while displaying it), and if it includes the last tape mounted during the most recent volume dump operation, then reload the volume using the reload_volume command with the -operator and -pvname control arguments. For example:

```
r vrld reload_volume -pvname xpub02 -operator Jones
```

Mount tapes as the reloader asks for them; it will indicate when all necessary tapes have been reloaded.

If the volume log is unavailable or damaged, reload the volume log for the disk using the recover_volume_log command. For example:

```
r vrld recover_volume_log xpub02
```

Mount the last volume backup tape for the volume backup group which includes the failing volume. The volume name of the last tape should be recorded in the tape log, as described above under "Backup Tape Logs." If volume backup operations were ongoing at the time of disk failure, you should mount the tape which was being written at the time of failure. After the volume log has been recovered, then reissue the reload_volume command, as shown above.

15. After volume reloading is complete, issue a set_drive_usage command to convert the drive back into storage system usage. For example:

```
sdu dska_06 ss
```

16. Issue the add_vol command to inform the system of the new location for the reloaded disk volume. For example:

```
add_vol xpub02 dska_06
```

17. Issue the add_lv command to add the logical volume containing the reloaded disk volume. For example:

```
add_lv Xpublic
```

18. If the system is in special session, return it to normal session:

```
word login
maxu auto
abs start
abs maxu auto
```

19. Perform the procedures for salvaging, quota adjustment, and connection failure detection described below under "Disk Volume Post-Recovery Procedures." This completes recovery of the volume.

Disk Volume Post-Recovery Procedures

Recovery from partial or total disk failure should be followed by several additional steps. These steps are described in the subsections which follow.

RECOVERY OF PARTITIONS AFTER RLV VOLUME RECOVERY

After recovering the RPV or other RLV volumes, you may want to copy information from the LOG, DUMP, CONF or FILE partitions. This can be done using the BCE copy_disk facility. For example, to copy the LOG partition onto a new RPV volume mounted on dskb_03, type:

```
copy_disk dskc_16 dskb_03 -partition log
```

The location of the partition on disk is determined according to the partition information in the disk volume labels.

VOLUME SALVAGING

When the system detects that a disk volume may be in an inconsistent state, it schedules volume salvaging for that volume. This salvaging occurs automatically during Multics bootload. It insures that all records on the disk volume appear in only one VTOCE or in the list of free records. It also deletes all perprocess VTOCEs (e.g., VTOCEs for segments in process directories remaining from an earlier Multics session).

You can manually force volume salvaging of all volumes of the RLV by booting Multics with the Root Logical Volume Salvage (RLVS) option:

```
boot rlvs
```

This command also performs a hierarchy salvage with quota adjustment for the root directory (>) and all directories immediately below the root (e.g., >sc1). These directories include those required to boot Multics.

Volume scavenging is an operation similar to volume salvaging, but it can be performed after the disk volume has been added to the system (when it is in use). The following command will create a Scavenger.SysDaemon process to scavenge all volumes that have inconsistencies:

```
x scav -all -auto -nopt
```

Many sites place this command in the start_up.ec of Utility.SysDaemon to insure that volumes are automatically scavenged as needed.

Refer to "Salvaging" earlier in this section for further details.

HIERARCHY SALVAGING

After a disk volume failure in which the disk has been reloaded, you must perform hierarchy salvaging of Multics directories to insure that they are consistent, and to detect connection failures (a situation in which a directory entry exists for a segment, but its corresponding VTOCE does not exist). After the system is operational, the following command should be issued to salvage all directories:

```
x repair salvquota > 4 -dcf -compact
```

This command creates four Salvager.SysDaemon processes to begin salvaging from the root directory (>). Directory salvaging repairs any damage, repairs improper quota used values, compacts directories which are using more space than necessary, and deletes branches which have connection failures.

To get information about file system damage, check the salvager output.

Refer to "Salvaging" earlier in this section for further details.

REVERSE CONNECTION FAILURE DETECTION

Neither volume salvaging nor volume scavenging detect reverse connection failures (a situation in which a VTOCE exists but its corresponding directory entry does not exist). After a volume has been reloaded, reverse connection failures may exist for a segment which was deleted after the most recent volume reload tape was written, or for a segment whose directory could not be successfully reloaded.

Segments associated with a reverse connection failure can be recovered by adoption, or the inaccessible VTOCEs can be deleted by garbage collection to recover the VTOCE and its disk records. Refer to Section 12, "Segment Adoption" and "How to Perform VTOCE Garbage Collection on a Pack," for instructions on these operations. A useful command for garbage collection on reloaded volumes is:

```
sweep_pv pvname -gc -delete
```

RECOVERING FROM A BAD CLOCK SETTING

If you discover that you have a bad clock setting, your response will depend on how long you've been running with the bad setting, whether the clock was set to a time in the future or a time in the past, and how much damage has been done to the system. In every case, you must weigh the amount of damage against the effort required to fix the clock setting. If you decide to fix the setting, you should follow the procedure below. Step 1 offers some suggestions on what to do first, depending on your situation.

1. a. If you've just brought the system up, and users haven't had a chance to do much work, you should crash the system. Do not perform an emergency shutdown. If the clock was set to a time in the future, you should keep the system down for as long as the clock was off by. Then you should continue with step 2.

   b. If the system has been running long enough for users to have done a significant amount of work, you should perform a normal shutdown. If the clock was set to a time in the future, you should keep the system down for as long as the clock was off by, if possible. Then you should continue with step 2.

   c. If the system has been running for some time with the clock set to a time far in the future, the damage to the system may be minimal. You should perform a normal shutdown, then continue with step 2.

   d. If the system has been running for a long time with the clock set to a time in the past, the damage to the system may be extensive. If it is, you should perform a normal shutdown. Then you should *RELOAD* the entire system with hierarchy backup tapes. Do *NOT* continue with this procedure.

2. Set the clock correctly. (Refer to "Calendar Clock" in Section 3.)

3. Do a "boot rlvs" with no other arguments.

4. At ring one command level, do a "salvage_vol -all" to manually salvage all of the volumes that haven't been mounted yet. These will be all volumes except those RLV volumes with hardcore partitions. Most of the dates in the VTOCEs will be corrected by this action, with the exception of date-time-volume-dumped dates.

5. Do an "add_lv" for all of the volumes that you just salvaged.

6. At ring four command level, bring the system up in a special session, without daemons. It's acceptable to let the daemons log in, but you should log them out immediately before they have a chance to get confused.

7. During the special session, run a complete hierarchy and quota salvage, using "x repair salvquota > 2 -rebuild -check_vtoce". This will correct all dates in the directories.

8. At this point, it's safe to log the daemons in and let users on the system.

9. If you use the volume dumper, you should run a complete volume dump as soon as possible. The hierarchy dumper isn't affected by a bad clock setting.

10. Multics unique-id values are based on the time, so there may be some unique-ids left in the system after this recovery which have already been duplicated or will be duplicated in the future. If such duplication occurs, unpredictable damage may result. For example, the wrong segment may be deleted when two segments have the same unique-id. The only way to correct this problem is to: (a) restore the file system to a point before the bad clock setting occured; (b) recover file system changes made since that point by using the hierarchy reloader. (DO NOT use the volume recovery subsystem.) In practice, the danger of damage is quite small. Sites should balance the likelihood of damage against the cost of recovery.

11. Some date-time-contents-modified and date-time-used dates will be incorrect after this recovery. You don't need to be concerned about these - they will fix themselves.

RECOVERING FROM BOOTLOAD CONSOLE FAILURE

If an unrecoverable bootload console error occurs, the system will take the following actions:

1. Search the configuration for an alternate console. The alternate chosen will be the next alternate encountered in the configuration deck. If the system finds a usable alternate, it will:

   a. Unassign the current bootload console and change its state to inop.

   b. Assign the alternate console as the new bootload console and change its state to on.

   c. Write a message in the syserr log indicating that automatic console recovery has occurred.

2. If no usable alternate consoles exist and there is an active message coordinator, the system will:

   a. Change the state of the current bootload console to inop.

   b. Send all syserr and normal traffic to the message coordinator.

   c. Write a message in the syserr log indicating that automatic console recovery has occurred.

3. If no usable alternate consoles exist, there is no active message coordinator, and the ccrf parameter was specified on the parm configuration card, the system will:

   a. Change the state of the current bootload console to inop.

b. Crash with the following message:

```
ocdcm_ (console_recovery): console recovery
   failure
```

If the console is truly inoperative, this message will not be seen but will appear in the flagbox and the syserr log.

4. If no usable alternate consoles exist, there is no active message coordinator, and the ccrf parameter was not specified on the parm configuration card, the system will:

a. Change the state of the current bootload console to inop.

b. Write a message in the syserr log indicating that the console is inoperative.

c. Continue running. Subsequent console traffic will be sent to the syserr log. You'll know that this has happened because no syserr messages will be printed. To confirm it, you may read the syserr log. You should either dial up and accept a message coordinator terminal, or crash the system directly into ESD. On a Level 68 system, this is done by executing switches with the DATA switches set to 024002717200. On a DPS 8 system, this is done by using the BCE 24002 command.

SECTION 11

DYNAMIC RECONFIGURATION PROCEDURES

OPERATIONAL PROCEDURES FOR RECONFIGURATION

Step-by-step procedures for adding and deleting processors, memory, IOMs, FNPs, logical channels, tape drives and disk drives, as well as procedures for adding an alternate bootload console, deleting the bootload console, and changing the bootload console, are available in the *Operator's Guide to Multics*, Order No. GB61. You can also add and delete MPCs, link adapters, and pages of memory. *

There are five commands you can use for dynamic reconfiguration: the reconfigure (rcf) initializer command, the adddev (addd) and the deldev (deld) initializer commands, the reconfigure privileged Multics command, and the set_system_console privileged Multics command. The reconfigure initializer command can only be executed in the initializer process in ring 4. It can be used to add and delete tape and disk drives. The adddev and deldev initializer commands can only be executed in the initializer process in ring 1. They can be used to add and delete tape and disk drives during system recovery, when system recovery must be done in ring 1 (for example, a volume reload of the RLV disk volumes or a hierarchy reload of a damaged critical system database, such as >sc1). These two commands allow you to add and delete tape and disk devices as necessary to set up the system for the recovery process. For example, they allow you to delete all 800/1600 bpi tape handlers and to add only 6250 bpi tape handlers, which is necessary when all system dump tapes are written at 6250 bpi. And of course, these commands allow you to delete known bad devices that shouldn't be used. The reconfigure and set_system_console privileged Multics commands require access to the highly privileged gates hphcs_ and rcp_sys_, and can only be executed in a privileged user process, in admin mode in the initializer process, or in SysAdmin and SysDaemon processes (which also have the necessary access). Detailed descriptions of all three of these commands are available in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

Only entities that are defined by configuration cards at bootload time can be added. These cards have fields to indicate if the unit is online or offline at bootload time. All the memories actually being used at the time of the bootload must be indicated as on in the state field of the mem configuration cards. These memories should be the first in the config deck. Memories that might be added later should be indicated as off in this field. Normally, there should be a mem card (either on or off) in the config deck for every memory in the installation. The cpu card for the bootload CPU must be indicated as on in the state field of the cpu card. Any other CPUs indicated as on are automatically added at the end of system initialization. Those indicated as off can be added later.

The PORT CONTROL switches for all 6000 SCUs running on the system should be set in the PROG CONT position for all ports at all times. Notice that after adding a memory, unused ports are left in the OFF position. After the adding of the memory is completed, all PORT CONTROL switches should be put in the PROG CONT position.

Notes on Adding and Deleting Processors

Before you add a CPU to the configuration, you must initialize it. The procedure for doing this is included in the procedure for adding a processor in the *Operator's Guide to Multics*, Order No. GB61.

If you attempt to add a CPU and it fails to start running for some reason, the software informs you as to the reason for the failure. If a problem exists, you can correct it. Then you can make another attempt to add the processor.

After your command to delete a processor has finished, the processor switches can be changed as desired; the system software ensures that the processor cannot access any of the memories. (It is not necessary to change the PORT CONTROL switches on all memories for the processor being deleted.)

Notes on Adding Memory

Before you add a SCU to the configuration, you must clear its store units. The procedure for doing this is included in the procedure for adding memory in the *Operator's Guide to Multics*, Order No. GB61.

Notes on Adding IOMs

IOMs and IMUs are both defined by an iom card in the configuration deck (the model field is set to either iom or imu). Therefore, the device type to use with the reconfigure command is "iom".

Before you add an IOM to the configuration, you must initialize it. The procedure for doing this is included in the procedure for adding an IOM in the *Operator's Guide to Multics*, Order No. GB61.

During IOM dynamic reconfiguration, the IOM's configuration settings are checked for consistency before the IOM is added to the system. Since the configuration settings cannot be read directly, they must be validated experimentally, by accessing the IOM using a series of test instructions.

If the configuration settings are incorrect, the experiment can damage arbitrary sections of main memory. Therefore, all modified memory pages are written out to disk before performing the experiment. The system will stop responding to users while these memory pages are being written and while the experiment is being conducted. This pause can last up to 15 seconds, depending upon the number of modified pages in memory.

If experimental results indicate that the configuration settings are incorrect, an error message is printed and the bce (crash) environment is entered. Emergency shutdown (ESD) is disabled after such crashes, since there are no modified pages in memory and since memory contents may have been corrupted by the experiment.

You can bypass the IOM configuration settings consistency check by adding the dris (don't read IOM switches) parameter to the parm card in the config deck. See Section 7 for details.

Converting Disk Drives from User I/O to Storage System Use

At any time, any disk drive is available for either storage system or user input/output use, but not both. The list_disks initializer command can be used to determine the usage status of an RCP disk_drive resource. All user input/output drives are listed as "i/o drive"; all others are storage system. The set_drive_usage initializer command can be used to change the usage status of a given drive. Step-by-step procedures for converting disk drives from user I/O to storage system use and vice versa are available in the *Operator's Guide to Multics*, Order No. GB61.

Action after a Failure in Reconfiguration

The reconfiguration commands are all controlled by a reconfiguration lock. If a reconfiguration attempt aborts unexpectedly for some reason, this lock may remain set. When the lock is set, any further attempts to reconfigure result in a message stating that reconfiguration is currently in progress. If this happens, the Multics command:

```
reconfigure$force_unlock
```

unlocks the lock so that further reconfiguration attempts may be made. However, it is quite possible that the reconfiguration databases will still be in an inconsistent state (e.g., indicating a CPU was added when it really was not). If such is the case, all reconfiguration requests should be delayed until after the next bootload.

SECTION 12

STORAGE SYSTEM MAINTENANCE OPERATIONS

HOW TO MOVE A PACK

Certain procedural steps must be followed when a disk pack is to be moved. The procedure varies according to whether or not the system is running when the pack is moved. The following discussion lists the procedural steps to follow when Multics is not running and when it is running. The commands used in these procedures are all described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

While Multics Is Not Running

1. Stop the drives involved, unload the packs, move them, and reload and restart them. *

2. If the root physical volume (RPV) is moved, change the root card.

3. If any pack that is moved is pointed to by a part card, change the part card to name the new location.

4. If any BCE exec_coms name specific drives, they must be changed. Modify them and reload them. *

5. If the config deck has changed, edit it and reload it.

6. Create a new BCE/Multics tape. *

7. Load BCE and Multics using the boot command or the auto exec_com.

8. If the RPV is moved, a new disk table is created. If this occurs, issue an add_vol initializer command for all other volumes of the RLV, and then for all other volumes.

9. If the RPV is not moved, the disk table is retained but contains some incorrect assumptions. Issue the list_disks initializer command to see what the disk table contains. For each incorrect assumption, issue a del_vol initializer command to delete the wrong assumption, and an add_vol initializer command to insert the actual situation. Do the RLV first.

10. Proceed with startup as usual.

While Multics Is Running

1. If any of the packs to be moved are part of the RLV, they cannot be moved while the system is running.

2. If any packs contain partitions, they cannot be moved while the system is running.

3. If any packs contain process directory segments (either the RLV or the volumes named in the set_pdir_volumes command), use the vacate_pdir_volume initializer command to force these segments off of the logical volumes.

4. If the packs can be moved, issue a list_disks initializer command to determine the logical volumes that contain them. Issue a del_lv initializer command to demount each such logical volume.

5. The del_lv initializer command stops all the drives in each of the logical volumes deleted. Cycle up the drives that were not to be moved.

6. Move the packs that are to be moved, and cycle up the new drives.

7. Issue a del_vol initializer command for each drive that had a pack taken from it.

8. Issue an add_vol initializer command for each drive that had a pack moved onto it.

9. Perform an add_lv initializer command for each logical volume containing a pack that is moved.

10. If any BCE exec_coms name specific drives, they must be modified and reloaded at the next shutdown.

When MSU0451 disk units are in use, it is possible to switch drives by changing address plugs as well as packs, so that disk drive changes are not seen by the software. Packs can be swapped only between two drives attached to the same MPC. This is done as follows:

1. Put the drive you are switching from (the old drive) in STANDBY mode by pushing the STOP button. If there is a disk volume on the drive you are switching to (the new drive), put that drive in STANDBY mode as well.

2. Remove the address plugs from both the old drive and the new drive immediately.

3. When the spindles stop, demount any volume from the new drive, and move the volume from the old drive to the new drive.

4. Push the START button to start the new drive.

5. After the new drive becomes ready, insert the address plug from the old drive into the new drive; insert the address plug from the new drive into the old drive.

Note that the new drive must already be ready before you insert the address plug. Note also that you must wait at least 30 seconds from the time you remove the plugs until the time you reinsert the plugs. This is because the MPC only polls devices every 15 seconds for status changes.

If the procedure above fails, the following procedure may be tried as an alternate:

1. Put the drive you are switching from (the old drive) into offline mode, by setting the rotary switch on the inside of the back door of the unit.

2. Power down the old drive.

3. Dismount the disk pack from the old drive and mount it on the drive you are switching to (the new drive).

4. Swap the address plugs between the two drives.

5. Put the new drive into offline mode, ready it, and then put it into online mode.

HOW TO EXPAND A LOGICAL VOLUME

To add a physical volume to a logical volume:

1. If necessary, format the disk pack using the online T&D tool, MTR, under TOLTS and MOLTS. Procedures for formatting both MSU0500/MSU0501 and MSU0451 disk packs with MTR are described in the *Multics Online Test and Diagnostics Reference Manual*, Order No. AU77.

2. If a drive must be added to the configuration, and the config deck is changed, the disk table may have to be recreated at the next bootload. If so, it will be empty, and you will have to issue the add_vol command for each old volume.

3. Issue an add_volume_registration command with the -lv and -pv control arguments to add the new physical volume to the specified logical volume.

4. Issue an init_vol command to write the label and VTOC of the new physical volume. Use the -special control argument if there are partitions, defective track space, or special space requirements. Note that every RLV volume must have a hardcore (HC) partition.

5. Issue an add_vol command to begin using the new physical volume mounted on the specified disk drive. If the root volume is being expanded, schedule a shutdown so you can change the root and part config cards as necessary.

6. Add the volume name to the volume dumper control files, if your site dumps by physical volume name.

The segment creation algorithm somewhat favors physical volumes with a high percentage of unused space. This is especially true in cases where the other physical volumes in the same logical volume are close to full. This fact might lead to excessive arm motion on the drive of the new volume and degraded performance. The sweep_pv command and the inhibit_pv command can be used to move segments between packs or inhibit segment creation on any pack. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

HOW TO COMPRESS A LOGICAL VOLUME

To delete one or more physical volumes from a logical volume:

1. Make sure that enough free space is available so that the logical volume can be compressed.

2. Make sure that adequate backup dumps and output from the BCE save command are available as a hedge against any problems that may arise.

3. The logical volume to be compressed must be mounted and in use.

4. Log in a SysDaemon process, preferably in ring 1, and issue the sweep_pv command from that process. The command should specify the physical volume to be removed and the -move and -force control arguments. Refer to the description of the sweep_pv command in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

   This operation (vacating the pack) may take 30 minutes to an hour. It should be done on a lightly loaded system, due to its impact on I/O performance.

   If several volumes are being removed, they may be vacated in parallel. Log in one SysDaemon process for each volume to be vacated in parallel. Issue the sweep_pv command from each of the processes, specifying the respective physical volume and the -move and -force control arguments.

5. Delete the physical volume from the volume dumper control files, if your site dumps by physical volume name.

6. Delete the volume log segment using the delete_volume_log command.

7. When the sweep_pv commands have terminated, inspect all error files, if any are produced. If segments cannot be moved, delete them, acquire access, or take whatever remedial action is appropriate. Use the sweep_pv command with the -move, -force, and -only control arguments to move such segments. It is not necessary to take any further action for segments in process directories.

8. If you're compressing the RLV, bump all users at some scheduled time. Then issue the delete_volume_registration command and shut the system down. Before rebooting, change the root and part config cards as necessary. Then reboot the system.

If you're compressing a non-RLV volume, issue the delete_volume_registration command during system operation. Then issue the del_lv command to demount the entire logical volume. Finally, issue the add_lv command to remount the entire logical volume. If will be necessary to issue an add_vol command for each physical volume remaining in the logical volume. (It is advisable to enter a special session to compress heavily used non-RLV volumes. This will avoid interactive user confusion and absentee user terminations.)

9. If segments could not be moved by step 4 above, quota recovery must be performed at some time to account for the quota used by those segments.

10. The RPV may not be deleted. Physical volumes on which process directory segments reside produce errors of the form:

```
"There was an illegal attempt to deactivate an AST entry"
```

in the error file and such segments are not moved. This is not a problem, since these segments are deleted anyway during the next bootload.

All of the commands used in this procedure are described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

HOW TO PERFORM VTOC GARBAGE COLLECTION ON A PACK

VTOC garbage collection consists of freeing the VTOC entries and pages of segments on a given pack that no longer correspond to branches in the storage system hierarchy. Since such segments have no branches, they cannot be used by users or system processes and waste valuable space. Such segments result when physical volumes are recovered via a BCE restore, or in certain crash situations. VTOC garbage collection should be performed periodically on all packs as time permits.

At sites where it is critical to recover the latest versions of segments in case of root pack failure, branches may be constructed automatically for segments lacking them. This process is known as adoption. (See "Segment Adoption," below.)

To perform a VTOC garbage collection on a pack, the pack must be part of a mounted, in-use logical volume. The garbage collection must be performed from a privileged process (SysDaemon). It may take 20 to 45 minutes.

The sweep_pv command with the -gc, -dl, and -force control arguments is issued for each physical volume to be garbage-collected. The -gc control argument specifies garbage collection; the -dl control argument causes deletion of these segments in addition to locating them; the -force control argument specifies that access is to be forced.

A report of the garbage collection is produced as well as an error file, if errors are encountered. The report is given a three-component name with "pvgc" as the first component. The error file (if it is produced) is given a three-component name with "pvef" as the first component. Both have the physical volume name as the second component of their name and the time the segment was created as the third component.

Multiple processes may be used in parallel to speed up garbage collection of multiple volumes. Only one process should perform garbage collection on a single volume.

VTOC garbage collection should be performed when only system users are logged in; users renaming directories while it runs can reduce its efficiency.

The hp_delete_vtoce command can be used to manually delete VTOCES known to be damaged or orphans. This command is documented in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

SEGMENT ADOPTION

In cases of root pack failure, a BCE restore (or reload via the volume reloader) of a root pack can cause valid and useful segments not on that pack to lose their branches. Usually, such segments must be deleted via VTOC garbage collection, and reloaded (in the case where volume backup is not used), or retrieved in either case. This situation can also come about when a directory is damaged by some other mishap than pack failure.

Branches for segments left "orphaned" in this way may be reconstructed automatically by a procedure known as segment adoption. Segment adoption constructs a branch for an orphan segment, but no name, ACL, and other branch information is recovered. A partially unique name is constructed for the segment. At sites where having the latest copy of segments is more important than having all of their names and access information correct, segment adoption should be used in cases of root pack or directory failure.

Lists of orphan segments may be obtained per physical volume basis by running sweep_pv over those volumes with the -gc (without the -dl) control argument. The list of segments that can be adopted are those marked as having "no entry" on the garbage collection report (pvgc) file.

All segments that can be adopted on a physical volume can be adopted by running the sweep_pv command over the volume with the -gc and -adopt control arguments. Segments may be adopted individually with the adopt_seg command. The pathnames of the constructed branches are reported to the garbage collection report file.

Access to the phcs_ and hc_backup_ gates is required to perform segment adoption.

BCE SAVE AND RESTORE

What Constitutes a Physical Volume Set

The BCE save and restore commands allow you to save or restore up to four sets of physical volumes at one time. A volume set is defined as all the physical volumes and partitions described in a control file or several control files that are to be saved in one tape set. The syntax of the commands allow you to define multiple control files for a set. These sets are defined by the parameters following the "-set" and "-restart" control arguments. See the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64, for a description of the syntax of the save and restore commands.

What Constitutes a Tape Set

A tape set is defined as the collection of tapes required to save a set of physical volumes. The tape reels are numbered 1 to N+1, and include one reel called the "Info" reel. Each tape label contains the name of the set as defined by the "tape_set" control file request. The "Info" tape is the last tape written during a save, and the first tape read during a restore. This tape contains information that relates the numbered tape reels and the saved physical volumes. This information aids in tape mount requests and allows for partial restores.

How to Create a Control File

The first step you must perform in preparation for a save or a restore is to create the necessary control file(s). This file defines the tape set name, tape devices, physical volumes, and partitions in the volume set. The control file requests are described under the descriptions of the save and restore commands in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

What follows is a sample control file being created at BCE command level.

```
qx
a
" Save/Restore Tape devices.
tape_device tapa_02 -density 6250
tape_device tapa_05 -density 6250
tape_device tapb_03 -density 6250
\f
w save_tapes
bl
a
" Save/Restore control file for the ROOT logical volume.
tape_set ROOT
physical_volume rpv dska_01
partition rpv dska_01 conf file log dump
physical_volume root2 dska_02
physical_volume root3 dska_03
physical_volume root4 dska_04
\f
w root_lv
q
```

The tape devices were defined in a separate control file so they could be used with several physical volume control files, during separate saves or restores.

How to Execute a Save and What Messages Are Displayed

Once the control files have been properly set up, you can begin the save process by typing the following:

```
save -set save_tapes root_lv
```

The tape devices are polled, and verified to be accessible and capable of the requested density. If problems are detected, a message is displayed and the device is removed from the list of available devices. This list is displayed in the order that the drives will be used.

```
save(ROOT): The following tape devices will be used:

     tapa_02  tapa_05  tapb_03
```

A check is made to insure that the physical volume requests match the corresponding disk packs. For each physical volume, a message is displayed. Errors are noted by (***) in columns 76–78 of the screen (not shown here).

```
save(ROOT): Multics Storage System Volume rpv on dska_01
                    Last updated: 05/08/86  1209.2 mst Fri

          Partition conf:    3908  for     4 records
          Partition file:   33836  for   255 records
          Partition dump:   34091  for  3500 records
          Partition log:    37591  for   256 records

save(ROOT): Multics Storage System Volume root2 on dska_02
                    Last updated: 05/08/86  1209.2 mst Fri

save(ROOT): Multics Storage System Volume root3 on dska_03
                    Last updated: 05/08/86  1209.2 mst Fri

save(ROOT): Multics Storage System Volume root4 on dska_04
                    Last updated: 05/08/86  1209.2 mst Fri
```

If multiple physical volume sets are requested, the above sequence is repeated for each one. After all the sets are examined, the following query is displayed:

```
save: Would you like to continue?
```

At this point, the output messages can be examined. If all is correct and acceptable, a "yes" response causes the save to begin. If any problems need to be corrected, a "no" response aborts the save and returns to BCE command level. After corrections are made the save request can be re-entered.

CAUTION: Any tape that is mounted with a write ring present will be considered a premounted save tape and will be written on when the tape device is selected.

If a tape is not mounted, the following message is displayed:

```
save(ROOT): Please mount tape# 1 on tapa_01.
```

If after two minutes no tape has been mounted, the following query is displayed:

```
save(ROOT): Would you like to skip to the next tape device?
```

One of the following responses must be entered:

yes, y
: This device is skipped and the next device is selected. The tape mount is then checked in the same manner. The skipped device remains in the list of available tape devices.

no, n
: This device is not skipped. The mount for this device is checked again in the same manner.

remove
: This device is removed from the list and the next device is selected. The tape mount is then checked in the same manner.

help, ?
: The possible responses are displayed.

Once a tape is mounted, the save process can continue. Shown below are the messages that will be displayed as the save progresses. This example assumes that tapes have been premounted on devices tapa_05 and tapb_03.

```
save(ROOT): Volume rpv, record 0, on tape# 1 (tapa_02)
save(ROOT):   Partition conf on rpv, record 3908, on tape# 1 (tapa_02)
save(ROOT):   Partition file on rpv, record 33836, on tape# 1 (tapa_02)
save(ROOT):   Partition dump on rpv, record 34091, on tape# 1 (tapa_02)
save(ROOT):   Partition log  on rpv, record 37591, on tape# 1 (tapa_02)
save(ROOT): Volume root2, record 0, on tape# 1 (tapa_02)
save(ROOT): Volume root3, record 0, on tape# 1 (tapa_02)
save(ROOT): Unloading tape# 1 from tapa_02, 23537 records (12 errors)
save(ROOT): Volume root3, record 4356, on tape# 2 (tapa_05)
save(ROOT): Volume root4, record 0, on tape# 2 (tapa_05)
save(ROOT): Unloading tape# 2 from tapa_05, 5477 records
save(ROOT): OK to write "Info" tape on tapb_03?
```

The query above allows for preassigned "Info" tapes. If you answer "yes", the current tape is used; if you answer "no", the tape is dismounted and the following occurs.

```
save(ROOT): Unloading tapb_03
save(ROOT): Please mount the "Info" tape on tapb_03.
```

After the correct "Info" tape has been mounted and written, the following is displayed, indicating that the save request is complete.

```
save(ROOT): Unloading "Info" tape from tapb_03, 3 records
save(ROOT): save complete...
```

HOW TO ABORT A SAVE

A save can be interrupted by use of the console "request" key. If you hit the "request" key while a save is in progress, the following prompt will appear:

```
save: Abort request:
```

You will then be required to input one of the following responses:

no, n
: This causes the request to be ignored and the save to continue.

abort
: This aborts all save sets and returns to BCE command level.

restart tape_set
This allows you to restart the specified tape_set, using its current tape device. You are then required to mount the "restart" tape on the device and to follow the procedure described below under "How to Restart a Save". Once the tape_set has been restarted, the remaining sets will continue operation.

stop tape_set
This aborts the specified tape_set, and continues the save for the other sets.

help, ?
This displays the possible responses, with a small description of each.

HOW TO RESTART A SAVE

Due to various problems that can arise while performing a save, it may be necessary to restart a set. The restart operation can be invoked in one of three ways:

- By using the "-restart_set" argument in the command line
- By giving the "restart tape_set" response to the "Abort request:" prompt described above. (See "How to Abort a Save" earlier in this section.)
- By giving the "restart_set" or "remove_device_from_set" response during error recovery. (See "How to Recover from Unrecoverable Tape Errors" later in this section.)

Restarting consists of skipping all volumes and/or partitions that have been successfully saved, restarting the save of one volume somewhere in the middle, and then continuing normally with the remaining volumes.

A restart must always start at the beginning of a tape, called the restart tape. This is usually the tape being written at the time of failure. The tape label successfully written at the beginning of this tape holds all the information about where to restart. If the tape label on that tape is unreadable, the previous successfully written tape in the set can be used as the restart tape and the information in its label can be used.

The tape label is read from the save tape from which you want to restart. If the tape is not already mounted, the following message is displayed and the normal mount procedure is executed.

```
save(ROOT): Please mount the "restart" tape on tapa_02.
save(ROOT): Tape# 2 on tapa_02, created 05/08/86  1535.3 mst Thu
```

After the tape label has been read, the tape creation time is checked. If the time is older than one week, the tape is rejected. This involves unloading the current tape and asking that another be mounted.

The tape label information is used to locate all the volumes that can be skipped and to find out what record number to start at when rewriting the tape. The following messages are displayed:

```
save(ROOT): Skipping volume rpv on dska_01.
save(ROOT): Skipping volume root2 on dska_02.
save(ROOT): Starting from record 4356 of volume root3 on dska_03.
```

You are then queried with the following:

```
save(ROOT): Do you want to replace or rewrite tape# 2 on tapa_02?
```

This query gives you the chance to select a different tape reel, in case the previous save was aborted because this tape contained too many errors. Below are the possible responses.

replace, rep
: The current tape will be unloaded and a new tape requested in its place.

rewrite, rew
: The tape will be rewound and used when the save begins again.

From this point on, the save resumes normal operation.

How to Execute a Restore and What Messages Are Displayed

Once the control files have been properly set up, you can begin the restore process by typing the following:

```
restore -set save_tapes root_lv
```

The tape devices are polled, and verified to be accessible. If problems are detected, a message is displayed and the device is removed from the list of available devices. This list is displayed in the order that the drives will be used.

```
restore(ROOT): The following tape devices will be used:

    tapa_02  tapa_05  tapb_03
```

At this point, the program needs to read in the contents of the "Info" save tape. This tape contains the list of volumes and partitions that were saved and the starting and ending tape number for each. The "Info" tape is the last tape written as part of a save. It allows program control over what tapes are mounted, which saves a lot of time in searching for tapes.

The program now attempts to read the tape on the first device in the list. If a tape is not mounted, the following message appears:

```
restore(ROOT): Please mount the "Info" tape on tapa_02.
```

If the tape which is read does not contain a label of "Info", the program queries you to find out if the "Info" tape is available. If you answer "no", the program uses the label information from the current tape in place of the "Info" data. It is in the same format but is not as complete. If you answer "yes", the current tape is unloaded and the mount/label read process is restarted.

If the "Info" tape is not available, the save tape closest to the end of the save should be read in its place. This will give the program the greatest amount of information.

The volumes to be restored are sorted so that they are in the same order as they were saved. Each of the disk labels is read and a display/check of the information is done. If a problem is detected, the volume is removed from the "to-be-processed" list. This procedure is duplicated for each restore set. Below is an example of the information that is displayed during this procedure. Messages that indicate a possible problem will have (***) in columns 76-78 (not shown here).

```
restore(ROOT): Multics Storage System Volume rpv on dska_01
                       Last updated: 05/08/86  1209.2 mst Fri

restore(ROOT): Multics Storage System Volume root2 on dska_02
                       Last updated: 05/08/86  1209.2 mst Fri

restore(ROOT): Multics Storage System Volume root3 on dska_03
                       Last updated: 05/08/86  1209.2 mst Fri

restore(ROOT): Multics Storage System Volume root4 on dska_04
                       Last updated: 05/08/86  1209.2 mst Fri
```

If multiple physical volume sets are requested, the above sequence is repeated for each one. After all the sets are examined, the following query is displayed:

```
restore: Would you like to continue?
```

At this point, the output messages can be examined. If all is correct and acceptable, a "yes" response causes the restore to begin. If any problems need to be corrected, a "no" response aborts the restore and returns to BCE command level. After corrections are made the restore request can be reentered.

The program now knows the first tape to be read from the label information or at least has a "best guess" if the first tape read was not the "Info" tape. It attempts to read this tape on the next tape device in the list. If the tape read is not the correct tape or if no tape is mounted, the following message is displayed:

```
restore(ROOT): Please mount tape# 1 on tapa_02.
```

If after two minutes no tape has been mounted, the following query is displayed:

```
restore(ROOT): Would you like to skip to the next tape device?
```

One of the following responses must be entered:

yes, y
: This device is skipped and the next device is selected. The tape mount is then checked in the same manner. The skipped device remains in the list of available tape devices.

no, n
: This device is not skipped. The mount for this device is checked again in the same manner.

remove
: This device is removed from the list and the next device is selected. The tape mount is then checked in the same manner.

help, ?
: The possible responses are displayed.

After a successful read of the current tape label, the program checks to see if another tape in the set is needed. If another tape is needed, a premount message is displayed. Shown below is a sample sequence of events during a restore process.

```
restore(ROOT): Tape# 1 on tapa_02, created 05/08/86  1525.0 mst Thu
restore(ROOT): Please premount tape# 2 on tapa_05.
restore(ROOT): Volume rpv, record 0, on tape# 1 (tapa_01)
restore(ROOT): Partition conf on rpv, record 3908, on tape# 1 (tapa_02)
restore(ROOT): Partition file on rpv, record 33836, on tape# 1 (tapa_02)
restore(ROOT): Partition dump on rpv, record 34091, on tape# 1 (tapa_02)
restore(ROOT): Partition log  on rpv, record 37591, on tape# 1 (tapa_02)
restore(ROOT): Volume root2, record 0, on tape# 1 (tapa_02)
restore(ROOT): Volume root3, record 0, on tape# 1 (tapa_02)
restore(ROOT): Unloading tape# 1 from tapa_02, 23537 records
restore(ROOT): Tape# 2 on tapa_05, created 05/08/86  1535.3  mst Thu
restore(ROOT): Volume root3, record 4356, on tape# 2 (tapa_05)
restore(ROOT): Volume root4, record 0, on tape# 2 (tapa_05)
restore(ROOT): Unloading tape# 2 from tapa_05, 5477 records
restore(ROOT): restore complete...
```

HOW TO ABORT A RESTORE

A restore set can be interrupted by use of the console "request" key. If you hit the "request" key while a restore is in progress, the following prompt will appear:

```
restore:  Abort request:
```

You will then be required to input one of the following responses:

no, n
: This causes the request to be ignored and the restore to continue.

abort
: This aborts all restore sets and returns to BCE command level.

restart tape_set
This allows you to restart the specified tape_set, using its current tape device. You are then required to mount the "restart" tape on the device and follow the procedure described below under "How to Restart a Restore." Once the tape_set has been restarted, the remaining sets will continue operation.

stop tape_set
This aborts the specified tape_set, and continues the restore for the other sets.

help, ?
This displays the possible responses, with a small description of each.

HOW TO RESTART A RESTORE

Due to various problems that can arise while performing a restore, it may be necessary to restart a set. The restart operation can be invoked in one of three ways:

- By using the "-restart_set" argument in the command line
- By giving the "restart tape_set" response to the "Abort request:" prompt described above. (See "How to Abort a Restore" earlier in this section.)
- By giving the "restart_set" or "remove_device_from_set" response during error recovery. (See "How to Recover from Unrecoverable Tape Errors" later in this section.)

Restarting consists of skipping all volumes and/or partitions that have been successfully restored, restarting the restore of one volume somewhere in the middle, and then continuing normally with the remaining volumes.

If you are restarting from the command line, then the "Info" tape must still be read before the "restart" tape.

The tape label is read from the save tape from which you want to restart. If the tape is not already mounted, the following message is displayed and the normal mount procedure is executed.

```
restore(ROOT): Please mount the "restart" tape on tapa_02.
restore(ROOT): Tape# 2 on tapa_02, created 05/08/86 1535.3 mst Thu
```

From the tape label, the program determines which volumes were completed on previous tapes and skips them. It then restarts the restore of the first volume on the tape that has been requested to be restored. The following messages are displayed:

```
restore(ROOT): Skipping volume rpv on dska_01.
restore(ROOT): Skipping volume root2 on dska_02.
restore(ROOT): Starting from record 4356 of volume root3 on dska_03.
```

From this point on the program reverts back into a normal operational mode.

How to Recover from Unrecoverable Tape Errors

During a save or restore, there are times when errors occur that require special handling. These are errors that are either nonretryable or that cause the retry process to fail. When an unrecoverable error occurs, a message is displayed that shows the error interpreted in English, plus the detailed status in hex (if required). After this message is displayed, you are queried as to the course of action that should be taken. Shown below is some sample error output.

```
save(ROOT): Device Attention, Handler check on tapb_03.
detailed status: 20 8C 2B 6D 0A 01 16 00 00 16 48 87 24
                 18 06 00 00 0C 00 00 08 08 80 00 00 00
save: Action:
```

One of the following responses must be entered:

abort
: This causes the program to abort the entire save/restore and return to BCE command level.

retry, r
: For errors that are retryable, this forces the retry process again. It is invalid for nonretryable errors.

skip, s
: This is only valid for data alert errors detected while doing a restore. The unreadable record is skipped and the restore continues by attempting to read the next record.

stop_set, stop
: This causes this set to be aborted, but all other sets to continue operation.

restart_set, restart, rt
: This allows you to restart this set, using the current tape device. You are then required to mount the "restart" tape on the device and follow the restart procedures. Once the set has been restarted, the remaining sets continue operation.

remove_device_from_set, remove
: Works like the "restart_set" request above, but removes the current tape device from the set and sequences to the next device before going through the restart process. This is not a valid response if the next device is the only tape device left in the set.

help, ?
: This displays the above possible responses.

OPERATIONS ON PHYSICAL VOLUMES

There are several commands that may be invoked to perform operations on physical volumes: adopt_seg reconstructs a lost directory branch; inhibit_pv and sweep_pv perform various operations on physical volumes for storage system maintenance purposes; and format_disk_pack formats disk packs. Complete descriptions of these commands are available in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

SECTION 13

SYSTEM MESSAGES AND LOGS

SYSTEM MESSAGES

This subsection describes the different types of messages produced by the Multics system during operation, including their forms, where they appear, which subsystems produce them, and what they mean.

The Form of a System Message

A system message usually begins with the time it was sent. After the time, the message usually gives its source. After the time and the source comes the body of the message.

There are four major kinds of messages. A *BCE message* is not indented and does not include a time or a source. (The source is always BCE.)

A *syserr message* is also not indented. It includes a time with a decimal point. Its source is either the name of a subsystem (for example, RCP) or the name of a program (for example, disk_control). Some syserr messages do not include their source.

A *message coordinator message* is indented one space. It includes a time without a decimal point. Its source is either a daemon (indicated by the daemon's label) or the answering service (indicated by "as").

An *initializer command response* is a response to an initializer command. It is not indented, and does not include either a time or a source. (The source is always the initializer process.)

Where Messages Appear

The following kinds of messages will usually appear on the bootload console:

- BCE messages
- Syserr messages:
    - RCP messages

- Disk error messages
- Salvager messages

The following kinds of messages will usually appear on the initializer terminal(s):

- Message coordinator messages:
  - Backup daemon messages
  - I/O daemon messages
  - Login, logout, and other answering service messages
- Initializer command responses

* BCE Messages

BCE messages are produced by BCE. An example of a BCE message is:

```
Booting t610 on IOM a chn 14 with m610 rev.11 firmware.
```

BCE messages are only produced when the system is at BCE level -- during startup and after certain system failures.

Syserr Messages

Syserr messages are produced by supervisor programs, and by user ring programs with access to privileged gates. If the bootload console ceases operation, syserr messages are automatically directed to the initializer process, which attempts to handle them.

RCP MESSAGES

RCP messages are produced by RCP. An example of an RCP message is:

```
1420.1 RCP: Authenticate tapa_05.  It has no label.
```

The most common RCP messages provide instructions for performing tape and disk mounts.

RCP Mount Messages

When a process requests the mounting of disk packs or tape reels, the system prints a message. There are three different kinds of messages, corresponding to the three types of demountable media: tape reels, storage system disk volumes, and user I/O disk packs. In each case, a message from RCP is printed on the bootload console and the audible alarm sounded. Step-by-step procedures for mounting tapes, user I/O disks, and storage system disks are available in the *Operator's Guide to Multics*, Order No. GB61.

RCP Access Messages

Access to all devices is controlled by the access control lists on special segments, one per device, called access control segments (ACSs). These segments are kept in the directory >sc1>rcp. If a device is added to the config deck or if one of these segments is lost in a crash, the system may type a message like the following at the next bootload:

```
RCP: Created >sc1>rcp>dska_16.acs with default access.
```

No users are able to access the device until the system administrator corrects the ACL on that ACS.

DISK ERROR MESSAGES

Disk error messages are produced by the supervisor. An example of a disk error message is:

```
0822.5 disk_control: dska_04 requires intervention.
```

Disk error messages tell the operator when a disk drive is in trouble, and may provide information that will help him solve the problem.

SALVAGER MESSAGES

Salvager messages are produced by the salvagers. Examples of salvager messages are:

```
0643.8 scavenger:  Begin scavenge of dska_01 by
   Scavenger.SysDaemon.z

0643.8 scavenge_volume:  Freed 72 VTOCEs on dska_01.

0643.9 scavenge_volume:  9 VTOCEs on dska_01 damaged.

0643.9 scavenger:  Scavenge of dska_01 by Scavenger.SysDaemon.z
   completed.
```

Whenever a salvager is invoked to correct a physical volume, the operator will receive a whole set of messages. These messages provide information. They don't require a response.

Message Coordinator Messages

BACKUP DAEMON MESSAGES

Backup daemon messages are produced by the volume and hierarchy backup systems. An example of a backup daemon message is:

```
1955 cd2 Input tape label
->cd2
```

A message from a daemon that begins with a "->" is called a *sentinel*, and indicates that the daemon wants input.

I/O DAEMON MESSAGES

I/O daemon messages are produced by the I/O daemon processes. An example of an I/O daemon message is:

```
1956 prtb prtb driver ready at 06/01/83 1956.1 est Wed
->prtb
```

Again, the line beginning with a "->" indicates that the daemon wants input.

LOGIN AND LOGOUT MESSAGES

Login and logout messages are produced by user and system processes when they login and logout. An example of an interactive user login message is:

```
1719 as LOGIN User1.ProjectA int a.h026.001 (create)
```

An example of an absentee user login message is:

```
1719 as LOGIN User4.ProjectB Q 3 abs2 (create) [my_absentee]
```

An example of a system process login message is:

```
1719 as LOGIN Backup.SysDaemon dmn bk (create)
```

An example of an interactive user logout message is:

```
1805 as LOGOUT User1.ProjectA int a.h026.001 0:17
   $6.92 (logout)
```

An example of an absentee user logout message is:

```
1805 as LOGOUT User4.ProjectB Q 3 abs2 0:09 $3.41 (logout)
```

An example of a system process logout message is:

```
1805 as LOGOUT Backup.SysDaemon dmn bk 5:28 $23.45 (logout)
```

A great number of login and logout messages are produced. When they're concerned with interactive and absentee users, they simply provide information, and don't require a response. When they're concerned with system processes, the operator should take note of them. They may mean that a daemon is logging in to perform a task that may require operator assistance, or possibly that a daemon is logging out because it's in trouble.

OTHER ANSWERING SERVICE MESSAGES

Most other answering service messages simply provide information. An example of another answering service message is:

```
0345 as act_ctl_:  bumping User4.ProjectB for inactivity.
```

Initializer Command Responses

Initializer command responses are produced by the initializer process. An example of an initializer command response is:

```
reconfigure:  CPU a is now running.
```

An initializer command response is always printed on the terminal that was used to issue the initializer command. Initializer command responses are logged for later analysis in the admin log.

Error Message Documentation

Every release of Multics includes an online error messages segment. This segment documents all of the error messages that can be generated by the system. It also documents quite a few messages which are not error messages, but just regular messages. The messages are listed in alphabetical order. Each message description tells you where the message gets printed, when you are likely to receive it, what it means, and what action, if any, you should take to respond to it.

If you receive an error message that you don't recognize or don't know how to respond to, you can look it up in this segment. The name of the segment is:

```
>doc>MR12.0>error_messages.doc
```

where "MR12.0" is the number of the Multics release under which you are running. We strongly recommend that you dprint a copy of this segment and keep it in the machine room.

SYSTEM LOGS

This subsection provides general information about Multics system logs, then describes each individual log, including what kind of messages they contain, how to get information from logs, how to store and discard old logs, and how to deal with common problems.

Multics System Logs

Multics keeps records of important events in *logs*. A log is a set of segments which contain log messages. Each log message is a record of a system event. System programs put log messages into logs using standard subroutines. Maintainers and administrators use logs to keep track of system errors, user problems, I/O devices and volumes, and operator activity.

All logs are families of segments. The newest member of the family has a name of the form log_name. Older log segments have names of the form log_name.YYYYMMDD.HHMMSS. A log segment consists of a header and a series of log messages. Each log segment records in its header the directory in which the next oldest log segment is located. You can use the display_log_segment command to see the information in the log segment header.

There are three logs that are always kept by the system: the syserr log, the answering service log, and the admin log. There are two other classes of logs that your site may or may not use. If your site runs Data Management, there will be a Data Management system log for each Data Management system. If your site defines message coordinator "log" destinations, there will be a log for each destination.

THE SYSERR LOG

The messages in the syserr log record many different events which occur in ring 0 and ring 1. These include:

- All I/O errors
- All RCP activity
- Software and hardware errors detected in rings 0 and 1
- Reconfiguration activity
- Activity relevant to system security

A syserr message which can't be logged is marked with "*lost".

Users see the syserr log as a normal log family, which consists of one or two log segments in >sl1, older log segments in >sc1>syserr_log, and even older log segments in >udd>sa>a>history. (The underlying relationships between these directories are explained in more detail in the paragraphs below.) The supervisor syserr mechanism puts messages into segments in >sl1. The answering service copies them to >sc1>syserr_log every few minutes. The crank moves segments more than a day old from >sc1>syserr_log to >udd>sa>a>history.

The system implements this view with a three part structure. This structure is required to accomodate the fact that syserr messages are created in different environments. The first part of the structure is called the wired log. This is a wired buffer reserved for syserr messages generated by wired software such as page control.

The second part of the structure is called the LOG partition. This is a special area of disk storage located on the RPV. It is reserved for syserr messages generated by paged software such as RCP. The LOG partition exists in the file system as one or two standard log segments in the directory >sl1. The most recent information in the LOG partition is contained in the segment named >sl1>syserr_log. A supervisor process called Syserr_Logger.SysDaemon copies messages from the wired log to the LOG partition.

The third part of the structure is called the permanent log. It consists of a set of standard log segments. The most recent information in the permanent log is contained in segments in the directory >sc1>syserr_log. The oldest information in the permanent log is contained in segments in the directory >udd>sa>a>history. The answering service copies messages from the LOG partition to the permanent log (i.e., it copies the entire contents of the older of the two segments in >sl1 to a new segment in >sc1>syserr_log).

The monitor_sys_log, print_sys_log, and summarize_sys_log commands (described later) all use the "-syserr" control argument to refer to the syserr log. Note that the LOG partition segments and the permanent log segments are treated together as one standard log family by the Multics logging commands.

For detailed information about syserr log messages, refer to "Syserr Log Messages" later in this section.

THE ANSWERING SERVICE LOG

The messages in the answering service log record important events in the answering service. These include:

- Logins and logouts (including accounting information)
- Use of the dial facility and dial manager
- System table installations
- Activity relevant to system security

The most recent log segment of the answering service log is always >sc1>as_logs>log. There will generally be some older log segments in the same directory. The crank moves segments more than a day old to >udd>sa>a>history.

The print_sys_log, monitor_sys_log, and summarize_sys_log commands (described later) all use the "-answering_service (-as)" control argument to refer to the answering service log.

THE ADMIN LOG

The messages in the admin log record initializer command executions. When an operator enters a command at the bootload console or an initializer terminal, or a privileged user uses send_admin_command to send a command, the command and all of its output are copied into the admin log. The admin log includes special messages that allow you to see to what I/O switch output was written. Normally, output is written to the switch user_output. However, it can also be written to other switches, e.g., error_output. Before writing a message in the admin log, the system checks to see if it came from the same I/O switch as the previous message. If not, one of these special messages is written first. A special message looks like this:

```
error_output:
```

The most recent log segment of the admin log is always >sc1>as_logs>admin_log. Older admin log segments are handled just like older answering service log segments.

The print_sys_log, monitor_sys_log, and summarize_sys_log commands (described later) all use the "-admin" control argument to refer to the admin log.

MESSAGE COORDINATOR LOGS

You can use the message coordinator (described in Section 8) to route message coordinator output to logs. All message cordinator logs have their most recent segment in >sc1>as_logs. The crank copies older segments to >udd>sa>a>history.

Some output from the initializer process shouldn't or can't be routed to message coordinator logs. The answering service has three switches which are connected to the message coordinator: severity1, severity2, and severity3. All messages which are written to these switches are also logged in the answering service log. Thus, they shouldn't be routed to a message coordinator log. Output written on the normal process I/O switches (user_output, error_output, and user_i/o) is *NOT* processed via the message coordinator. It is logged in the admin log. Thus, it can't be routed to a message coordinator log.

To define a log as a destination of a virtual console, you must use the initializer define command. A typical define command, found in the system_start_up.ec, would be:

```
sc_command define iolog log iolog
```

This adds a destination to the virtual console named "iolog" of a log named "iolog." The most recent segment of this log would be >sc1>as_logs>iolog.

To route messages to a message coordinator log, you must use the initializer route command. A typical system_start_up.ec fragment would be:

```
sc_command define ut_log log ut_log
sc_command route ut user_i/o ut_log
```

This adds a destination to the virtual console named "ut_log" of a log named "ut_log." The most recent segment of this log would be >sc1>as_logs>ut_log. It then routes all output from the switch user_i/o on the source "ut" (where Utility.SysDaemon is usually logged in) to >sc1>as_logs>ut_log.

For all standard daemons, the switch user_i/o will catch all of their output. For I/O daemons, there are three interesting switches: user_i/o, error_i/o, and log_i/o. The daemon writes all output in response to daemon commands on the user_i/o switch. The daemon writes all errors intended for the system operator on the error_i/o switch. The daemon writes records of all processed daemon requests on the log_i/o switch. Thus, it is often useful to route log_i/o to an additional destination. A typical system_start_up.ec fragment would be:

```
sc_command define ioc tty a.h004
sc_command define iolog log iolog
sc_command route prta user_i/o ioc
sc_command route prta error_i/o *ioc
sc_command route prta log_i/o ioc
sc_command route prta log_i/o iolog
```

This sends all output from the daemon to the terminal a.h004, and also sends the record of all processed requests to the log iolog. ("*ioc" turns on the beeper when errors are printed on the terminal.)

The print_sys_log, monitor_sys_log, and summarize_sys_log commands (described later) all use the "-mc_log (-mcl)" control argument to refer to message coordinator logs. For example:

```
print_sys_log -mcl iolog
```

DATA MANAGEMENT SYSTEM LOGS

The messages in a Data Management (DM) system log record important events in that Data Management system. These include:

- Bootloads (recovery and initializations)
- Data Management daemon activity
- Exception conditions
- Idle time-outs
- Shutdowns

There is one DM system log for each Data Management system. The most recent log segment of a DM system log is always >site>Data_Management><authorization>>dm_system_log, where <authorization> is the AIM authorization of the Data Management system (or system_low if your site doesn't use AIM). There can't be more than one Data Management system running at the same authorization, so there can never be more than one DM system log with the same name. There will generally be some older log segments in the same directory as the most recent log segment. The crank does *not* move the oldest log segments to >udd>sa>a>history. DM system log segments can only be moved by a Data Management system administrator.

The print_sys_log, monitor_sys_log, and summarize_sys_log commands (described later) all use the "-dm_system (-dms)" control argument to refer to DM system logs. Note that you must have access to the dm_admin_gate_ gate to manipulate a DM system log with one of these commands.

Getting Information from Logs

There are three basic commands that retrieve information from logs: print_sys_log, monitor_sys_log, and summarize_sys_log. These three commands are described below. In addition, there are five specialized commands that extract information from the syserr log: display_cpu_error, fnp_data_summary, io_error_summary, mos_edac_summary, and mpc_data_summary. These five commands are not described here.

Use the print_sys_log (psl) command to print some or all of the messages from any log on your terminal. Here are some examples of the use of this command:

```
psl -syserr -last 5
```

will print the last five messages from the syserr log.

```
psl -syserr -from -1day -to -2hours
```

will print all the messages entered in the syserr log after one day ago and before two hours ago.

```
psl -as -match LOGIN -from -1hour
```

will print all the messages containing the string "LOGIN" entered in the answering service log in the last hour.

```
psl -admin -match /^word/ -last 5
```

will print the last 5 messages in the admin log that *begin* with the string "word". If necessary, this command will continue to search until it has scanned the entire log.

Use the monitor_sys_log (msl) command to have your process print some or all new messages in one or more logs on your terminal as they are added to the log(s). Here are some examples of the use of this command:

```
msl -as -match LOGIN
```

will print all LOGIN messages on your terminal as they happen. If you are using the video system, you may want to give these messages their own window.

```
msl -as -match LOGIN -osw log_window
```

will send the message to the window associated with the switch log_window. Note that you must have previously used the window_call command to define the window.

Use the summarize_sys_log (ssl) command to prepare reports which sort, select, and count log messages. See the system-supplied master.ec for examples of the use of this command by the crank.

Complete descriptions of the print_sys_log, monitor_sys_log, and summarize_sys_log commands, as well as the display_cpu_error, fnp_data_summary, io_error_summary, mos_edac_summary, and mpc_data_summary commands, are available in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

Storing and Discarding Old Logs

The crank uses the move_log_segments command to move old log segments from their inital directories to history directories. When the move_log_segments command moves a log segment to a new directory, it records the name of the directory in the header of the next newest log segment (in that family). You must use this command when you move log segments, to preserve the records of where old log segments live. The crank also deletes very old log segments.

Dealing With Common Problems

CRASHES WITHOUT ESD

When there is a crash without ESD, the log segments can be damaged. This will cause answering service initialization or the define command to fail. Usually, the problem is that the most recent segment is damaged. You can rename or delete it and retry initialization. If older segments are destroyed, you may have to use the set_log_history_dir command to correct the trail of old log segments.

SYSERR LOG COPY FAILURES

When an answering service log copy of the syserr log fails, you will receive the following messages:

```
syserr_log_man_:  <description of error>
syserr_log_man_:  Automatic syserr log copying disabled.
```

You should respond by fixing the problem, then typing the following command in the initializer process (using admin mode or the sac command):

```
syserr_log_man_$restart_copying
```

DAMAGED LOG FAMILIES

When one or more log segments of a log family have been damaged, the log commands can't find old log segments in the history directory (>udd>sa>a>history).

The first thing you should do is use the display_log_segment command to display the information in the header of each log segment in the log family. Start with the most recent log segment and continue displaying headers until you find one which contains an incorrect history directory. (Remember that each log segment's header records the directory in which the next oldest log segment is located.)

When you find a log segment whose header contains a bad history directory, use the set_log_history_dir command to correct it. Type:

```
set_log_history_dir <log segment> <history dir>
```

It's a good idea to continue displaying headers until you've checked all the log segments in the log family, in case more than one history directory is recorded incorrectly.

The set_log_history_dir command is described in the *Multics Commands and Active Functions* manual, Order No. AG92.

SYSERR LOG MESSAGES

This subsection provides more detailed information about syserr log messages, including their format, and the meaning of those messages which contain binary data.

Syserr Log Contents

The following is an example of the contents of the syserr log as they are printed by the print_sys_log command (described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64).

```
1985-03-11 Mon est
13:10:01    1034548  0 added memory e
13:13:02    1034549  0 added cpu b
13:13:03    1034550  4 RCP: Assigned tape_06 to Dumper.SysDaemon.z
13:13:04    1034551  0 RCP: Attached tape_06 for Dumper.SysDaemon.z
13:13:05    1034552  3 RCP: Mount Reel 50202 without ring on tape_06
                        for Dumper.SysDaemon.z
13:14:06    1034553  0 RCP: Detached tape_06 from Dumper.SysDaemon.z
14:21:07    1034554  0 disk_control: EDAC performed for dska_07
                      (channel A20). rec 21745, sect 442150,
                      main 13262000.
14:21:08    1034555  0 disk_control: Auto retries for dska_07
                      (channel A20). rec 21666, sect 440560,
                      main 11550000.
14:30:09    1034556  4 RCP: Assigned tape_02 to Fred.AvgUser.m
14:30:10    1034557  0 RCP: Attached tape_02 for Fred.AvgUser.m
14:30:11    1034558  0 RCP: Note (tape_02) - 50202,den=1600
14:30:12    1034559  3 RCP: Mount Reel 50202 with ring on tape_02 for
                        Fred.AvgUser.m
14:31:13    1034560  0 RCP: Detached tape_02 from Fred.AvgUser.m
14:32:14    1034561  0 hardware fault: parity fault on CPU C by
                                       Herbie.SysMaint
14:32:15    1034562  4 mos_memory_check: EDAC error on mem c store a
^           ^        ^ ^
|           |        | |
|           |        | +-->  text of message.
|           |        |
|           |        +-->  severity.
|           |
|           +->  sequence number.
|
+--> time message logged.
```

Format of Syserr Log Messages

Each syserr log message consists of the following components:

sequence number
: A number that is increased by one for each message. This generally serves to number the messages, but sequence numbers may be missing from some groups of messages.

time
: The date and time when the message was logged.

severity
: A combined action code and sorting class. The low-order decimal digit is the syserr action code. The high-order digits are the sorting class. Both are explained in more detail below.

text
: The text of the message.

binary data class
(optional) A character string that identifies the structure/type of the binary data, if any, included in the message. It is explained in more detail below.

binary data
(optional) A binary data structure containing more detailed information about the event than that contained in the text. This information is in a machine-readable form. It is explained in more detail below.

SEVERITY CODES

A severity code is associated with each message in the syserr log. It consists of two parts: an action code and a sorting class.

Action Codes

The action code is the least significant digit of the severity code. Constants for the action codes are defined in syserr_constants.incl.pl1. They are interpreted (at message creation time) according to the following table:

SYSERR_PRINT_ON_CONSOLE	0	(print message on console, log message)
SYSERR_CRASH_SYSTEM	1	(print message on console, activate console alarm, log message, return to BCE)
SYSERR_TERMINATE_PROCESS	2	(print message on console, activate console alarm, log message, terminate affected process)
SYSERR_PRINT_WITH_ALARM	3	(print message on console, activate console alarm, log message)
SYSERR_LOG_OR_PRINT	4	(try to log message, print it on console if wired log is full)
SYSERR_LOG_OR_DISCARD	5	(try to log message, discard it if wired log full)
UNUSED	6-9	

Messages with the action code SYSERR_CRASH_SYSTEM typically will not appear in the permanent log, since the system immediately crashes. In all other cases, an attempt is made to log the message. If the wired log buffer is full, the message is printed on the console (unless the action code is SYSERR_LOG_OR_DISCARD).

Sorting Classes

The sorting class is the next (and most) significant digits of the severity code. The current values for sorting classes (as defined in syserr_constants.incl.pl1) are:

SYSERR_SYSTEM_ERROR	00	(general system error or status; leading zero will not appear in printed log)
UNUSED	10	
SYSERR_COVERT_CHANNEL	20	(security audit message regarding possible covert channel activity)
SYSERR_UNSUCCESSFUL_ACCESS	30	(security audit message regarding denial of access to a resource)
SYSERR_SUCCESSFUL_ACCESS	40	(security audit message regarding granted access to a system resource)
UNUSED	50-90	

Sorting classes 20, 30, and 40 are for security audit messages. For more information about security auditing, refer to the *Multics System Administration Procedures* manual, Order No. AK50.

BINARY DATA CLASSES AND BINARY DATA

Some syserr messages contain structured binary information in addition to the text of the message. The binary data is tagged with a 1 to 16 character binary data class. (Before MR11.0, it was tagged with a fixed binary data code.) Binary data class names and old binary data code constants are defined in syserr_binary_def.incl.pl1.

The log perusal command print_sys_log does not display binary data included in syserr messages by default. To display binary data as a dump of octal words, use the -octal control argument. To display binary data in interpreted form, use the -interpret {CLASSES} control argument, where CLASSES is an optional list of those types of binary data to be interpreted. If CLASSES is omitted, all classes of binary data contained in the messages are interpreted. For a complete description of the print_sys_log command, see the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64. In addition to the print_sys_log command, there are also commands which operate on groups of syserr messages with the same binary data class. For example, the mos_edac_summary command digests information contained in messages with the data class "mos".

The following paragraphs describe each of the syserr binary data classes which can be found in the syserr log, and provide examples of expanded binary data. In each example, the text of the message is included for completeness. As a convention, the text is preceded by "*text*" to distinguish it from that portion of the message which is expanded binary data. "*text*" is not printed by the system. The meaning of the texts of the messages is not addressed here. For a description of any particular text, see the message documentation supplied with the release (>doc> [release] >error_messages.doc) or the system module indicated in the message. Note that binary data of a particular class may be associated with any number of completely different texts.

io_status

The system logs a syserr message with binary data of this class when an interrupt whose status indicates an error or an unusual condition occurs on an I/O device. This includes disk errors, tape errors, and console errors. The interpretation of the data performed by print_sys_log produces the device, channel, status, and (if present) detailed status . The detailed status will be suppressed if it is not useful, as determined by heuristics. Although the print_sys_log command will interpret the binary data for this class of message, use of the io_error_summary command is recommended to isolate frequently occurring problems. (For more information, see io_syserr_msg.incl.pl1, which describes the format of the binary data. It is five words when just the basic information is present and eleven words when the detailed status is included. Also see expand_io_status_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
DEVICE (chnl IOM_TAG_AND_CHANNEL_NUMBER). MAJOR_STATUS.
   MINOR STATUS. {DETAILED_STATUS}.
```

The following are examples of syserr messages which contain binary data with the class io_status, including text and interpreted binary data:

```
*text* ioi_masked$interrupt: I/O error.
tapf_06 (chnl A18). Device Data Alert. Lateral parity alert.

*text* ioi_masked$interrupt: I/O error.
tapf_06 (chnl A18). MPC Data Alert. Multi-track error.

*text* disk_control: EDAC performed for dska_07 (channel B22).
*text* rec 21666, sect 440560, main 536000.
dska_07 (chnl B22). Channel Ready. EDAC correction performed.

*text* disk_control: Auto retries for dska_07 (channel A20).
*text* rec 22614, sect 457440, main 10440000.
dska_07 (chnl A20). Channel Ready. Retried 1 time.
```

The modules which log syserr messages which contain binary data with the class io_status are: disk_control, ioi_masked, and ocdcm_.

The system logs a syserr message with binary data of this class when a hardware error fault is signalled unexpectedly in ring 0. The binary data includes the associated machine conditions. The interpretation of the data performed by print_sys_log produces an octal dump of the machine conditions and history registers. Although the print_sys_log command will interpret the binary data for this class of message, use of the display_cpu_error command is recommended, as it supplies a more useful interpretation. (For more information, see syserr_fault_msg.incl.pl1, which describes the format of the binary data. It consists of machine conditions followed by history registers. Also see expand_hwfault_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Pointer Registers:
   SEG_NO|WORD_OFFSET(BIT_OFFSET
      [RING_NO] <there are 8 of these>
x0-x7: OCTAL_HALFWORDS <there are 8 of these>
a: A_REG_OCTAL_WORD q: Q_REG_OCTAL_WORD e: E_REG_OCTAL_WORD
   t: TIMER_REG_IN_OCTAL ralr: RING_ALARM_REG
Fault Register: FAULT_REG_IN_OCTAL
SCU Data:
OCTAL_WORDS <there are 8 of these>
EIS Info:
OCTAL_WORDS <there are 8 of these>

<The following 4 lines appear for Level 68 processors>

OU History Reg Data:
OCTAL_WORDS <there are 32 of these>
CU History Reg Data:
OCTAL_WORDS <there are 32 of these>
DU History Reg Data:
OCTAL_WORDS <there are 32 of these>
APU History Reg Data:
OCTAL_WORDS <there are 32 of these>

<The following four lines appear for DPS 8 processors>

DU/OU History Reg Data:
OCTAL_WORDS <there are 32 of these>
CU History Reg Data:
OCTAL_WORDS <there are 32 of these>
APU #2 History Reg Data:
OCTAL_WORDS <there are 32 of these>
APU #1 History Reg Data:
OCTAL_WORDS <there are 32 of these>
```

The following is an example of a syserr message which contains binary data with the class hwfault, including text and interpreted binary data:

```
*text* verify_lock: null_pointer condition by EJUser.Multics.a
Pointer Registers:
   42|017360(0) [0]      230|003540(0) [0]
  102|000021(0) [0]      230|002500(0) [0]
   16|013372(0) [0]    77777|000001(0) [4]
  230|000220(0) [0]      325|002004(0) [0]
x0-x7:  011300 003004 000002 000731 002520 000000 000000 001040
a: 134417065731 q: 000000000001 e: 000 t: 000043155 ralr: 1
Fault Register: 000000000000
SCU Data:
000127170041 000004000051 477777776020 000000000000
076773500200 000001000400 500000100400 000100100400
EIS Info:
000400000000 000400000000 000000000010 004077777770
001074000115 000000000000 002142000105 000077777671
DU/OU History Reg Data:
760204177036 076764235400 760204177036 076764235000
760204177036 076765755000 760204177036 076766236000
760204177036 076766236000 760204177036 076767236000
760204177036 076770236000 760204177036 076770236000
760204177036 076771116500 760204177036 076772116500
760204177036 076772116500 760204177036 076772116500
760300377036 076773116500 760102377636 076773116500
760301377436 076773116500 720301375436 076773116500
CU History Reg Data:
600021755100 054567660442 200011755100 052147372002
200121236100 052147360402 600021600004 054567700442
200021600004 000770040442 200121236100 052147340402
600011116007 054567720442 200011116007 000000010402
200111601004 000767770442 700021371520 054567740442
300021371520 052150760402 200011371400 000000010402
200111100400 000001000400 000111100400 000000000402
400111100500 054567760442 000101100500 175100530000
APU #2 History Reg Data:
002006235100 000000000000 002006235100 000000000000
002006235100 000000000000 002006235100 000000000000
000737755100 000000000000 000736236100 000000000000
077004600004 000000000000 000734236100 000000000000
000001116007 000000000000 076777601004 000000000000
001076371520 000000000000 000001371400 000000000000
000100100400 000000000000 000000100400 000000000000
000001100500 000000000000 000001100500 000000000000
APU #1 History Reg Data:
003251000402 175077540040 003250200402 024666520000
003250102402 175057750000 003250006442 114620060304
002300007540 052147370040 002300007540 052147360000
001270002076 000770040000 002300007540 052147340000
001270002000 000000010040 001270002076 000767770040
002300007540 052150760000 777770000200 000000014040
001270002000 000001000040 001270002000 000000000040
777771000400 175100534040 777771000401 175100534002
```

The modules which log syserr messages which contain binary data with the class hwfault are: hardware_fault, scavenger, system_startup_, and verify_lock.

mos

If your site is doing mos_memory_check polling (which is invoked by the poll_mos_memory command), the system logs a syserr message with binary data of this class when a MOS EDAC error occurs. The interpretation of the data performed by print_sys_log produces the chip type, the board identifier, and the location on the board. Although an interpretation for this class of message is supplied by print_sys_log, the mos_edac_summary command is recommended for use in isolating chips which exhibit high error rates. (For more information, see scr.incl.pl1, which describes the format of the binary data. It consists of the System Controller mode register. Also see expand_mos_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
MEMORY_TYPE, NNk chip, Error: board X, chip YYY.
```

The following are examples of syserr messages which contain binary data with the class mos, including text and interpreted binary data:

```
*text* mos_memory_check: EDAC error on mem d store b.
MOS-M128, 16k chip, Error: board A, chip 14D

*text* mos_memory_check: EDAC error on mem d store a.
MOS-M128, 16k chip, Error: board H, chip 02C
```

The module which logs syserr messages which contain binary data with the class mos is mos_memory_check.

voldamage

The system logs a syserr message with binary data of this class when it detects possible damage to a physical disk storage volume. The only existing case of this is the "device read not complete" condition. The interpretation of the data performed by print_sys_log produces only the physical volume name. (For more information, see the first two words of segdamage_msg.incl.pl1, which describe the format of the binary data. They hold the physical and logical volume unique IDs. Also see expand_voldamage_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Volume: PHYSICAL_VOLUME_NAME
```

The following is an example of a syserr message which contains binary data with the class voldamage, including text and interpreted binary data:

```
*text* page_fault: device read not complete dskg_42 075321
Volume: pub053
```

The module which logs syserr messages which contain binary data with the class voldamage is page_error.

segdamage

The system logs a syserr message with binary data of this class when it detects new or existing damage to a segment in the hierarchy. Damage to a segment may be caused by paging problems, file map checksum problems, or volume inconsistency problems. The binary data includes the UID path of the subject segment. The interpretation of the data performed by print_sys_log produces the pathname of the damaged segment (as derived from the UID path). (For more information, see segdamage_msg.incl.pl1, which describes the format of the binary data. Also see expand_segdamage_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Segment: PATHNAME
```

The following are examples of syserr messages which contain binary data with the class segdamage, including text and interpreted binary data:

```
*text* evict_page: fatal parity error moving page, frame at
15400000, SCU a Segment: >udd>Multics>EJUser>EJUser.profile.

*text* scavenge_volume: damaged switch found on for report.compout
at 1134 Segment: >udd>SysLib>AvgUser>finance>report.compout
```

The modules which log syserr messages which contain binary data with the class segdamage are: activate, page_error, salvage_pv, and scavenge_volume.

mdc_del_uidpath

The system logs a syserr message with binary data of this class when Master Directory Control discovers that a nonexistent directory has been deregistered. The binary data includes the UID pathname of the subject directory. The interpretation of the data performed by print_sys_log produces the pathname of the missing directory (not including its entryname), as derived from the UID path. (The format of the binary data is 16 words representing the UID path of the directory in question. There is no include file defining this simple array: "dcl uid_path (0:15) bit 36 aligned;". For more information, see expand_mdc_del_uidpath_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Directory: PATH
UID path: OCTAL_WORDS
```

The following is an example of a syserr message which contains binary data with the class mdc_del_uidpath, including text and interpreted binary data:

```
*text* mdc_repair_$validate_uidpaths: Master directory entry with
*text* bad uidpath deleted from Pubvol_2. >udd>??
Directory: >udd>-UNLISTED-
UID path: 777777777777 435712736432 435734234434 000000000000
000000000000 000000000000 000000000000 000000000000
000000000000 000000000000 000000000000 000000000000
000000000000 000000000000 000000000000 000000000000
```

The module which logs syserr messages which contain binary data with the class mdc_del_uidpath is mdc_repair_.

mmdam

The system logs a syserr message with binary data of this class when it detects main memory damage and removes a main memory frame due to parity errors. The interpretation of the data performed by print_sys_log produces the address of the main memory frame and the controller in which it resides. (For more information, see syserr_mmdam_msg.incl.pl1, which describes the format of the binary data. It is two words: the first contains the 24 bit main memory address of the page in question; the second contains the four character controller tag. Also see expand_mmdam_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Page at addr: MAIN_MEM_PAGE_ADDR, controller: SCU_TAG.
```

The following is an example of a syserr message which contains binary data with the class mmdam, including text and interpreted binary data:

```
*text* page_fault: Deleting main memory at 1540000, SCU a,
*text* due to parity errors.
Page at addr: 1540000o, controller: a.
```

The modules which log syserr messages which contain binary data with the class mmdam are: hardware_fault and page_error.

mpc_poll

If your site is doing MPC polling (invoked by the poll_mpc command), the system logs a syserr message with binary data of this class when MPC polling encounters an error. No interpretation of the data is provided by print_sys_log. Analysis of this class of message is performed only by mpc_data_summary. (For more information, see poll_mpc_data.incl.pl1, which describes the format of the binary data.)

The module which logs syserr messages which contain binary data with the class mpc_poll is poll_mpc.

fnp_poll

If your site is doing FNP polling (invoked by the poll_fnp command), the system logs a syserr message with binary data of this class when FNP polling encounters an error. No interpretation of the data is provided by print_sys_log. Analysis of this class of message is performed only by fnp_data_summary. (For more information, see poll_fnp_data.incl.pl1, which describes the format of the binary data.)

The module which logs syserr messages which contain binary data with the class fnp_poll is poll_fnp.

config_deck

The system logs a syserr message with binary data of this class during system initialization, to provide a record of the bootload config deck. The binary data simply contain configuration cards from the config deck BCE used to boot the system. The interpretation of the data performed by print_sys_log produces the printed representation of the cards in that portion of the config deck (the whole config deck may be split over more than one syserr message). (For more information, see config_deck.incl.pl1, which describes the format of the binary data. Also see expand_config_deck_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
CONFIG_CARD_IMAGE <there are N of these, depending on the size
   of the binary>
```

The following are examples of syserr messages which contain binary data with the class config_deck, including text and interpreted binary data:

```
*text* Config deck, part 1 of 2
note  ****  MAIN  FRAM  E     ****
cpu   a  7  on    dps8  70.  8.
iom   a  1  iom   on
mem   a  4096.  on
note  ****  CONT  ROLL  ERS   ****
mpc   mspa  609.  a  20.  2  b  20.  2
mpc   mtpa  502.  a  12.  1  b  12.  1
mpc   urpa  600.  a  24.  4
note  ****  PERI  PHER  ALS   /     CHAN  NELS  ****
prph  dska  a  20.  2  451.  8.  500.  2.  501.  10.  0  40.
   501.  2.
prph  fnpa  a  17.  6670.  on
prph  opca  a  28.  6601.  80.  on
prph  prta  a  24.  1600.  600.  136.
prph  prtc  a  35.  1600.  600.  136.
prph  tapf  a  18.  2  0  2  630.  2  610.  2  0.  8.  630.  2.

*text* Config deck, part 2 of 2
note  ****  PART  ITIO  NS    ****
part  dump  dska  7
note  ****  PARA  METE  RS    ****
parm  wlim  500.  vtb   66.  ttyb  65536.  chwm  hcpt  freq  400.
note  ****  TUNI  NG    PARA  METE  RS    ****
schd  1000000  2  2  100  2  12.
sst   1000.  500.  300.  100.
tcd   88.  400.
note  ****  MISC  ELLA  NEOU  S     ****
clok  5  est  96.
dbmj  64.  700.  400.  150.  60.  25.
```

The module which logs syserr messages which contain binary data with the class config_deck is system_startup_.

vtoce

The system logs a syserr message with binary data of this class when it detects damage to the storage system. The binary data includes a copy of the VTOCE of a damaged segment or directory, which is saved for later analysis. A vtoce class message immediately follows a segdamage class message. The interpretation of the data performed by print_sys_log simply produces an octal dump of the VTOCE, broken into its main components. (For more information, see vtoce.incl.pl1, which describes the format of the binary data. It is a VTOCE image. Also see expand_vtoce_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Seg VTOCE UID: OCTAL_WORD, original name: ENTRY_NAME
VTOCE Header:
OCTAL_WORDS <there are 12 of these>
File map:
OCTAL_HALFWORDS <there are 256 of these>
VTOCE Permanent info:
OCTAL_WORDS <there are 40 of these>
```

The following is an example of a syserr message which contains binary data with the class vtoce, including text and interpreted binary data:

```
*text* scavenge_volume: Damaged vtoce 6274 (dskf_42).
Seg VTOCE UID: 134417066243, original name: "vtoce.msgs"
VTOCE Header:
000000000000 134417066243 377001001000 456211060024
456211050210 001000000000 075321777020 000000000002
000000000000 000000000000 000000000000 000000000000
File map:
075321 777020 777020 777020 777776 777776 777776 777776
777776 777776 777776 777776 777776 777776 777776 777776
777776 777776 777776 777776 777776 777776 777776 777776
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
777777 777777 777777 777777 777777 777777 777777 777777
```

```
VTOCE Permanent info:
000000000000 000000000000 000000000000 000000000000
000000000000 000000000000 000000000000 456211050210
166055000001 000000000000 000000000000 000000000000
777777777777 124175365342 124176054376 127506006747
133346065051 000000000000 000000000000 000000000000
000000000000 000000000000 000000000000 000000000000
166164157143 145056155163 147163040040 040040040040
040040040040 040040040040 040040040040 040040040040
456211026126 516404202602 003775002350 000000000000
000000000000 000000000000 000000000000 000000000000
```

The module which logs syserr messages which contain binary data with the class vtoce is scavenge_volume.

access_audit

The system logs a syserr message with binary data of this class when security auditing is done in rings 0 and 1. (Note that other messages using this binary format are placed in the answering service log.) The interpretation of the data performed by print_sys_log produces:

1. Information about the audited process ("Subject") and its security related attributes ("ring", "Auth", etc.).

2. Information about the object being accessed (File_System_Object, File_System_Attribute, RCP_Object, Admin_Object, Special_Object, or Other_Object). The details for each of these objects will differ.

These messages are typically important only to the system security administrator. They are described in detail in the *Multics System Administration Procedures* manual, Order No. AK50. (For more information, see access_audit_bin_header.incl.pl1, which describes the format of the first part of the binary data (information about the operation and process), and access_audit_ssobj_info.incl.pl1, access_audit_rcp_info.incl.pl1, and access_audit_pnt_info.incl.pl1, which describe the second part of the binary data (detailed information about the object). Also see expand_access_audit_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log which is common to all of these messages is:

```
Subject: GROUP_ID (ring N), PID=OCTAL_WORD,
Auth: L:NNNNNN, Min: L:NNNNNN, Max: L:NNNNNN
OBJECT_TYPE, operation type: OPERATION_TYPE
{,operation detail: OCTALo}
```

The general format of the interpreted data for file system objects is:

```
Object: {branch | link} OCTAL_UID in DIR_PATH,
dtem is DATE
Raw mode: MODE  Ring brackets: N,N,N  Class: L:NNNNNN.
{(Ex mode: MODE  Ex ring brackets: N,N,N).}
Switches: ENTRY_SWITCH_VALUES.
```

The general format of the interpreted data for RCP objects is:

```
Type: RCP_OBJECT_TYPE, Name: NAME, Owner: USER_ID, Access
class: L:NNNNNN-L:NNNNNN, Raw mode = MODE, Ring brackets = N,N.
Attributes: ATTRIBUTES_STR
Flags: REGISTRY_ENTRY_SWITCH_VALUES
```

The general format of the interpreted data for PNT entries is:

```
User id = GROUP_ID, Operations = OPERATION_STR
{, Changed password} {, Changed network password}
[Old | New] PNT info:
Alias = STR, Authorization range = L:NNNNNN-L:NNNNNN,
Audit flags = AUDIT_FLAGS_STR, Flags = PNT_FLAGS_STR
{, Password timelock = DATE_TIME_STR}

<the presence of the second occurrence depends on the operation>

Old PNT info:
Alias = STR, Authorization range = L:NNNNNN-L:NNNNNN,
Audit flags = AUDIT_FLAGS_STR, Flags = PNT_FLAGS_STR
{, Password timelock = DATE_TIME_STR}
```

The following are examples of syserr messages which contain binary data with the class access_audit, including text and interpreted binary data:

```
*text* Audit (dc_find): GRANTED modification of fs_obj access
*text* for AvgUser.SysLib.a (0:000000) Level=0 to segment
*text* >process_dir_dir> BZBBcBCbBBBBBB> BBBJPdGfcpgCFk.ioi
   (0:000000).
Subject: AvgUser.SysLib.a (ring 0), PID=007400104003,
Auth: 0:000000, Min: 0:000000, Max: 0:000000
File_System_Attribute, operation type: Modify_Access,
operation detail: 5o
Object: branch 134417067151 in >process_dir_dir>-NO-ACCESS-,
DTEM is 04/02/85  1620.1 est Tue
Raw mode: rw  Ring brackets: 1,1,1  Class: 0:000000.
Switches: ^dirsw,per_process,^safety,^multiple_class,^audit,
^security_oos,^entrypt,^master_dir.
```

```
*text* Audit (mseg_check_access_): GRANTED acceptance of a wakeup
*text* for EJUser.Multics.a (0:000000) Level=4 to segment
*text* >udd>Multics>EJUser>EJUser.mbx (7:777777).
Subject: EJUser.Multics.a (ring 4), PID=006300104004,
Auth: 0:000000, Min: 0:000000, Max: 7:777777
Special_Object, operation type: Modify_Access
Object: branch 130355157266 in >user_dir_dir>Multics>EJUser,
DTEM is 11/30/84  1556.2 est Fri
Raw mode: rw  Ring brackets: 1,1,1  Class: 7:777777.
(Ex mode: rew  Ex Ring brackets: 0,0,0).
Switches: ^dirsw,^per_process,^safety,multiple_class,^audit,
^security_oos,^entrypt,^master_dir.

*text* Audit (rcp_access_kernel_): GRANTED assign device for
*text* writing for AvgUser.SysLib.a (0:000000) Level=4 to
*text* tape_drive tapf_15 (0:000000-7:777777) <raw_mode=RW
*text* rcp_ring_brackets=5,5>.
Subject: AvgUser.SysLib.a (ring 4), PID=007400104003,
Auth: 0:000000, Min: 0:000000, Max: 0:000000
RCP_Object, operation type: Modify
Type: tape_drive, Name: tapf_15, Owner: system, Access
class: 0:000000-7:777777, Raw mode = RW, Ring brackets = 5,5.
Attributes: track=9,model=630,den=1600,speed=125
Flags: device,^volume,^usage_locked,^release_locked,
^awaiting_clear,has_acs_path

*text* Audit (rcp_access_kernel_): GRANTED status of rcp object
*text* for AvgUser.SysLib.a (0:000000) Level=1 to tape_vol m-ls
*text* (0:000000-0:000000) <raw_mode=RW rcp_ring_brackets=4,4>.
Subject: AvgUser.SysLib.a (ring 1), PID=007400104003,
Auth: 0:000000, Min: 0:000000, Max: 0:000000
RCP_Object, operation type: Read
Type: tape_vol, Name: m-ls, Owner: system, Access
class: 0:000000-0:000000, Raw mode = RW, Ring brackets = 4,4.
Attributes: track=9,den=1600,length=2400
Flags: ^device,volume,^usage_locked,^release_locked,
^awaiting_clear,has_acs_path
```

The modules which log syserr messages which contain binary data with the class audit_access are: access_audit_ and page_error.

ibm3270_mde

The system logs a syserr message with binary data of this class when the IBM3270 multiplexer detects a "queue write" operation being attempted on a channel for which a write is already queued. The interpretation of the data performed by print_sys_log produces the state of the device on which the error occurred. (For more information, see ibm3270_mpx_data.incl.pl1, which describes the format of the binary data. Also see expand_ibm3270_mde_msg_.pl1, which performs the interpretation for print_sys_log.)

The general format of the interpreted data produced by print_sys_log is:

```
Device index=N, name="STR", addr="STR",
screen size=N, line size=N, position=N,
next_write_chan=N, next_poll_chan=N, next_control_chan=N,
Flags=MDE_FLAGS_STR.
```

The following is an example of a syserr message which contains binary data with the class ibm3270_mde, including text and interpreted binary data:

```
*text* ibm3270_mpx: Attempt to queue write while write queued
   a.h205.d03
Device index=3, name="d03", addr="x",
screen size=480, line size=80, position=3,
next_write_chan=45, next_poll_chan=46, next_control_chan=47,
Flags=listen,^dialed,^printer,hndlquit,^waiting_for_ready,
^erase_req,^sound_alarm,^control_queued,^end_of_page,
^keyboard_restore,^rawo,^rawi,raw3270,raw3270_in_effect,
write_queued.
```

The module which logs syserr messages which contain binary data with the class ibm3270_mde is ibm3270_mpx.

SECTION 14

METERING AND TUNING

This section describes the techniques and tools used to measure many of the Multics supervisor functions and to improve system response by controlling priorities and other system functions.

The Multics system is equipped with facilities for collection of data about system operations (metering) and provides adjustable parameters to control priorities and other system functions (tuning). This section explains the Multics data collection capabilities and discusses corrective actions to be applied to achieve desired system performance.

Note: detailed information about operating, metering, and tuning the Multics disk DIM is provided in Appendix J.

The various metering tools provide information on how the Multics system works. As diagnostic aids, they can show how the users are actually using the system, what particular actions are being performed most frequently, and whether or not and in what area of the system bottlenecks exist. Specific tools determine reasons for degradation of system performance. This information can then be used to tune the system for maximum performance.

Metering data is first accumulated by the supervisor code. This code:

- Records the number of times an event occurs or a particular piece of code is executed
- Records the time required to perform a task
- Stores the data in metering cells

The metering commands can then extract this data by reading and storing the current values of relevant metering cells. Finally, the metering commands report this data by:

- Comparing current metering cell values with previously read values
- Compiling the desired statistics
- Arranging the data in a useful format (report or diagram) and displaying it

The information reported by the metering commands determines important resource usage and can be used to diagnose specific areas that can be causing performance degradation. The system can then be tuned to change its operating characteristics (including parameters, configuration, and/or workload) based on the data and insights gained from the system's meters to best suit the operating environment and/or site policies. Specifically, tuning consists of adjusting the system's hardware, hardware configuration, static table sizes, dynamic tuning parameters, and workload.

When interpreting meters, it is necessary to know the current configuration and tuning parameter values, and to understand their implications. When changing configuration and tuning parameter values, it is necessary to know the current values of the relevant meters, and to understand the expected effect on the meters of the changes being made. The guidelines discussed under "Suggested Values and Guidelines" later in this section apply to both metering and tuning. Explanations of underlying system mechanisms, which must be understood in order to correctly interpret metering results and adjust tuning parameters, occur throughout this section.

The subject of configuring, tuning, and metering a Multics system is very complex, and requires a clear understanding of various aspects of the system's operation. It is very easy to misinterpret meters and mis-tune a system if one has only a superficial understanding of this subject. Thorough, repeated study of this section is advised.

Complete descriptions of all of the metering commands discussed in this section are available in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64. That manual also includes some guidelines on appropriate meter values, in the context of the command descriptions.

The metering_gate_, metering_util_, spg_ring_0_info_, spg_util_, and system_performance_graph$line subroutines, which may be useful to writers of new metering commands, are described in the *Multics Subroutines and I/O Modules* manual, Order No. AG93.

METERING

Metering allows you to monitor the usage of system resources. Metering tools determine whether or not the system is fully utilized, whether the usage is useful work or overhead, and how the availability of resources is reflected to users as response time. This information can then be used to tune the system, as well as to plan for future needs.

This subsection describes the detection and diagnostic techniques of metering, explains the metering databases and the activities they keep track of, briefly describes the categories and functions of the metering commands, and discusses metering design.

Overview of Metering

Metering serves two purposes. It allows you to detect the existence of actual or potential performance problems, and it allows you to diagnose the causes of performance problems.

DETECTING PERFORMANCE PROBLEMS

Two useful measures of system performance are response time and throughput. It is desirable to minimize response time and maximize throughput. Unfortunately those objectives are incompatible. A simple and intuitively obvious result of queueing theory is that if a system is loaded to its maximum capacity (maximizing throughput), the people that it serves will experience very long waiting times. To avoid this, it is necessary to provide excess capacity to serve peak loads, and allow the system to go idle at times of light load.

The decision as to how much excess capacity to provide must be made by each Multics site, based on the cost of excess capacity versus the cost of having people wait to be served. This decision can be expressed in terms of response time and throughput, as follows: maximize throughput, subject to the constraint that response times will be at least as good as a specified set of values, for at least a given percent of the time.

Metering commands can be used to determine how well these objectives are being met. Failure to meet the objectives can be considered a performance problem. The response_meters command can be used to determine if response time objectives are being met. The concepts involved in measuring response time are explained in the description of that command, in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64. The total_time_meters (ttm) command provides the best measure of throughput. The concepts involved in time metering are explained under "CPU Time Metering," later in this section. Additional information on time metering is given in the description of the ttm command, and throughout this section.

Even when response time and throughput are within acceptable limits, it is possible that parts of the system are at or near saturation, and that adjustments could be made that would improve performance beyond that which is minimally acceptable. To detect situations like this, it is a good idea to periodically (daily or weekly) run a complete set of metering commands, and learn to recognize output values that are typical for your site. Sudden changes in values, or values that are significantly different from the guidelines discussed later in this section, are cause for further investigation.

These periodic measurements should always include separately recorded values for a time interval (of half an hour to several hours) at a time when the system is heavily loaded. The system's behavior under heavy load is quite different from that under light load, and proper configuration and tuning are obviously more critical under heavy load.

DIAGNOSING PERFORMANCE PROBLEMS

The system resources whose saturation can degrade system performance fall into three categories:

1. CPUs

2. Paging hardware
   - memory
   - disk I/O capacity (channels and disk arms)

3. Shared system tables (such as the AST, described below)

The objective of system configuration and tuning is to achieve a good balance among these resources, with enough excess capacity in each category to handle peak loads. If the usage of any one of these resources reaches saturation, system performance will be degraded, and any excess capacity in the other categories will go idle and be wasted. This subject is discussed further under "Tuning."

The metering commands that give the best picture of utilization of scarce system resources are the following:

- CPUs
  total_time_meters, traffic_control_meters, traffic_control_queue

- Disk I/O
  disk_meters

- Memory and system tables
  file_system_meters, post_purge_meters

The system_performance_graph command produces a graphical summary of overall system performance and resource usage. Other metering commands are useful in a detailed problem analysis, once the general area of difficulty has been determined.

There are a number of system options that are intended to be used only in debugging and problem analysis, or in unusual operational situations. Their use during normal operation can be the cause of otherwise unexplainable performance problems. These options are enabled by various means, including hardware switches, the config deck, and tuning parameters. Thus, the site's periodic performance measurements should include the print_configuration_deck and print_tuning_parameters commands, and a visual inspection of hardware switch settings. The normal values and settings for the site should be known and documented, and any departure from the normal should be investigated.

Configuring and tuning a Multics system is a complex operation. It requires a good understanding of the system's underlying mechanisms, and a clear idea of what you expect to accomplish by each change that you make. Changing configuration and tuning parameters based on superficial knowledge or trial and error is always a mistake. If you do not know what you are doing, you will probably make things worse.

Metering Databases

This subsection contains a description of several of the major databases associated with metering. These databases include:

- The system segment table (sst) database
- The traffic control data (tc_data) database
- The configuration deck (config_deck) database
- The disk segment (disk_seg) database

SYSTEM SEGMENT TABLE (SST) DATABASE

The SST is a wired database (i.e., it cannot be removed from main memory). It contains the data used to manage the contents of main memory and the AST that describes active segments. The subsystems of the supervisor known as page control and segment control also maintain data in the SST.

The SST consists of two parts:

- SST header
- Active segment table (AST)

The SST header contains meters, counters, list pointers, and pointers to other databases in the SST. The AST consists of entries, known as AST entries or ASTEs, each of which contains per-segment data and a page table. A page table is an array of page table words (PTWs). Every page table in the system maintained by page control is part of an AST entry; thus, the AST is that place where all page tables in a system reside.

The SST is closely associated with another wired database, the core map. The core map is an array of table entries (CMEs) describing each frame of main memory in the system and its contents and state.

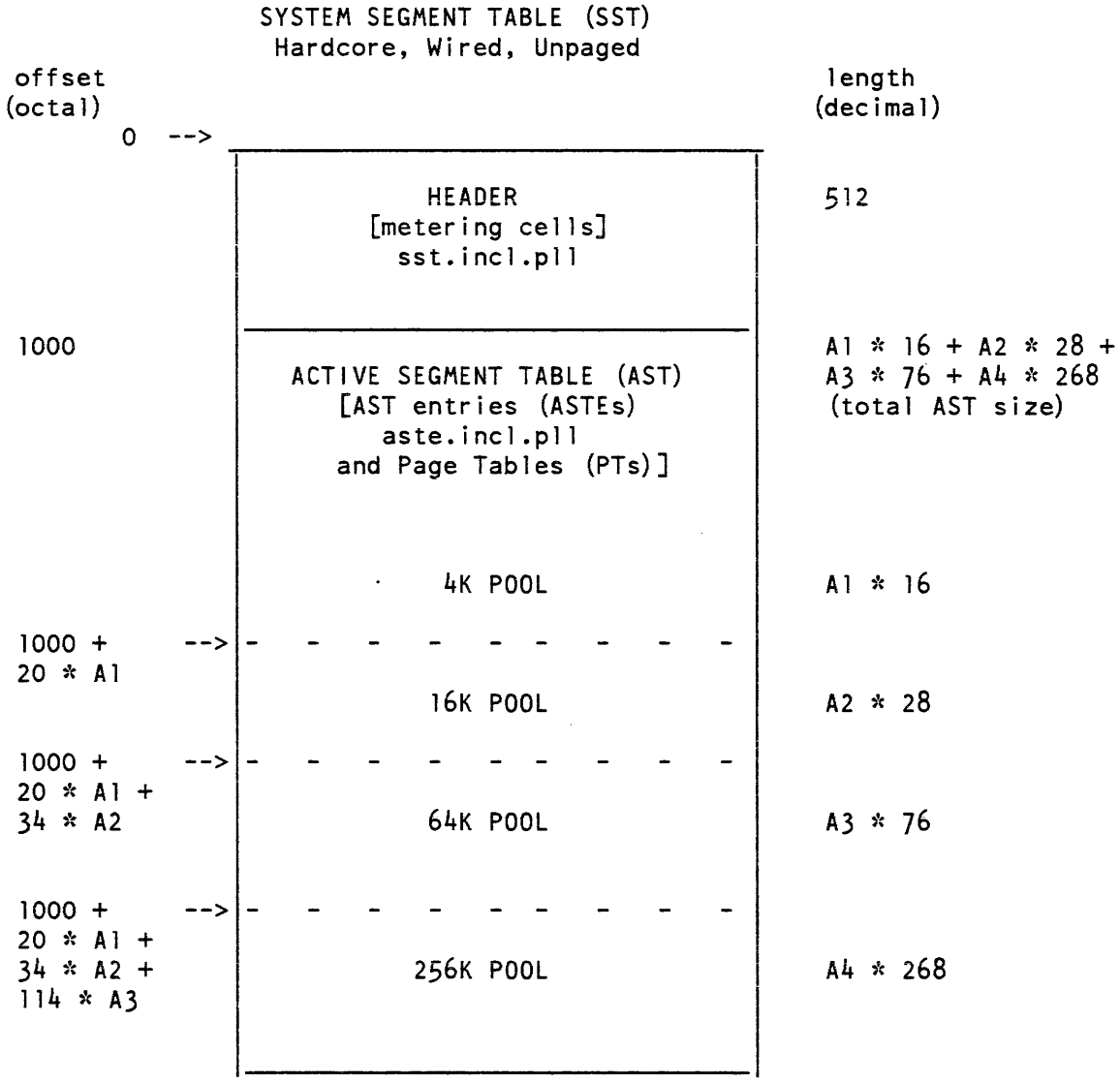
The core map entries, AST entries, and PTWs contain numerous implicit and explicit threads and pointers linking data about a given page of a given segment, defining the structure of the storage system hierarchy among AST entries, and maintaining recency-of-use information for the algorithms that multiplex main memory and the AST.

In any given Multics configuration, a fixed number of frames of main memory are available: this is the amount of main memory configured. The task of page control is to satisfy the demand for main memory by *multiplexing*; i.e., controlling the sharing in an orderly fashion, of the main memory frames in response to page faults. A page fault is the hardware condition that occurs when an attempt is made to use a page that is not in main memory. The meters reported by the file_system_meters command when the -page control argument is specified (see the command description in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64) indicate the activity of, and load on, page control. These meters (and others) can be analyzed to determine if main memory is being used effectively, if a paging bottleneck exists, or to interpret other visible manifestations of paging behavior.

During any given bootload of Multics, a fixed number of AST entries are available. This number is determined at bootload time by the sst card in the configuration deck. This number is thus the upper limit on the number of segments that can have page tables in main memory at any given time. Since all non-supervisor segments must have a page table in main memory in order to be used, the available page tables are multiplexed by segment control in response to segment faults. A segment fault is the hardware condition that occurs when an attempt is made to use a segment that does not have a page table in main memory.

There are four sizes of AST entries, capable of describing segments having up to 4, 16, 64, or 256 pages. AST entries of a given size are grouped into a pool. The number of entries in each pool is specified at bootload time by the sst configuration card (see Section 7 for more information on configuration cards). The meters reported by the file_system_meters command indicate segment fault activity in the four AST pools. These meters (and others) can be analyzed to determine if the AST is being utilized effectively, if an AST bottleneck exists, or to interpret any other performance manifestations of segment fault activity. An AST is being utilized effectively if it is large enough to prevent excessive segment fault activity, but not so large as to needlessly use memory that could be available for paging. See the *Multics Storage System* manual, Order No. AN61, for more information on the format of these databases and the meaning and interpretation of all the data contained in them.

A small side issue in tuning an AST is to specify the number of ASTEs for the pools to waste as little space as possible in the last page of the AST. This is generally done by choosing the desired pool sizes and then adding enough 4K ASTEs to fill up the last page. Table 14-1 shows the formula for computing the total size of the AST. See the segment sst_seg.cds for complete details on the layout and content of sst_seg.



AST size is specified in configuration deck:

```
sst A1 A2 A3 A4

A1 = # ASTEs in 4K pool
A2 = # ASTEs in 16K pool
A3 = # ASTEs in 64K pool
A4 = # ASTEs in 256K pool
```

Table 14-1. Computing Size of AST

TRAFFIC CONTROL DATA (TC_DATA) DATABASE

The tc_data segment is the wired database of the Multics traffic controller. The Multics traffic controller is responsible for managing the assignment of physical processors to Multics processes, and all issues relating to the relative priorities of processes. The functions assumed by the traffic controller are often known as multiprogramming, multiprocessing, scheduling, dispatching, and processor management.

Four databases reside in tc_data:

- tc_data header
- Active process table (APT)
- Work class table (WCT)
- Interprocess transmission table (ITT)

The header of tc_data contains static meters, counters, list heads, and other information used by the traffic controller. Much of this data can be displayed via the traffic_control_meters command, described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

By far the most important component of tc_data is the APT. The APT consists of APT entries (APTEs). One APTE is required for each process in the system. Thus, the number of APTEs is the true maximum number of processes that can exist at any time. The number of APTEs is fixed for the duration of the Multics bootload and is specified at bootload time by the tcd configuration card. Some APTEs, e.g., those of the initializer and the syserr logger daemon, exist for the duration of the bootload; most, however, are used and freed in response to login, logout, and new_proc commands. The APT entry of a process contains all the information required by the supervisor when that process is not running. It is used in conjunction with other databases when the process is running. Typical of such information is the process identifier of the process, the CPU time and memory usage charged to that process, the descriptor segment base register (DSBR) value to be loaded to physically switch into that process, etc. The APT entry also contains scheduling parameters associated with the process, including dynamically computed statistics accumulated during recent running of that process. This information is used by the traffic controller to assign priorities and processor resources to the process.

For each processor configured on the system there is a special process known as the *idle process* of that processor. The idle process of a processor is a full-fledged process with an APT entry. Idle processes are run when no other work, i.e., ready process, can be found for a processor to run, or the traffic controller determines that no more processes should be granted eligibility, based on main memory or administrative sharing constraints. Time spent running idle processes is known as *idle time*. The total_time_meters command displays the accumulated amounts of the various types of idle time (see the command description in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64, and the discussion of CPU time metering at the end of this subsection). The idle processes are always at the end of the eligible queue.

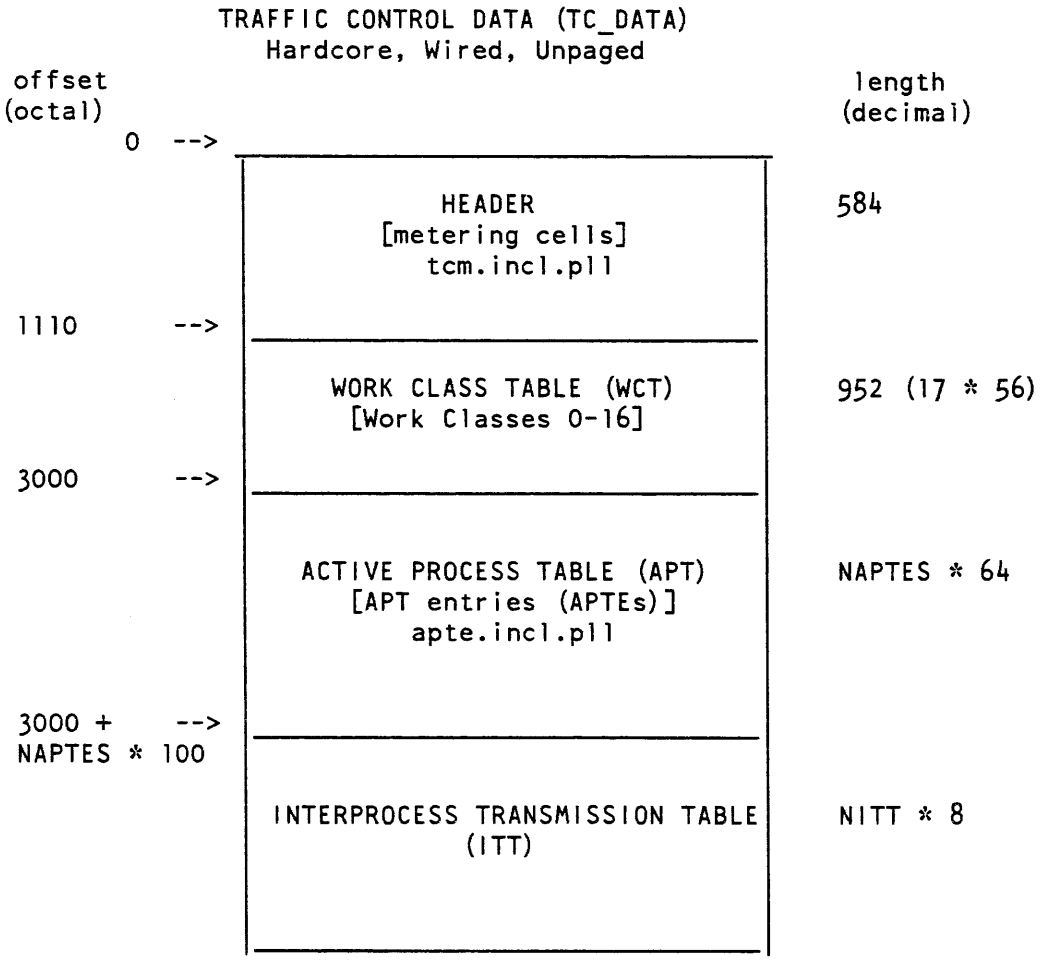
The work class table (WCT) consists of WCT entries (WCTEs) defining the work classes known to the system. A WCT entry contains the parameters of the work class, recent statistics accumulated during running of processes in the work class, and the head of the queue of ready, non-eligible processes in the work class. Work class parameters and statistics can be displayed via the work_class_meters command, described in the *Multics Administration Maintenance and Operations Commands* manual, Order No. GB64. The status of the eligible queue and work class queues can be displayed via the traffic_control_queue command, also described in that manual.

The interprocess transmission table (ITT) consists of all interprocess messages associated with wakeups. There are two kinds of wakeups corresponding to the "fast" and "regular" event channels (see the ipc_ subroutine described in the *Multics Subroutines and I/O Modules* manual, Order No. AG93). Wakeups for "fast" event channels do not involve transmission of data between processes; rather, a bit is turned on in the APT entry of the target process, and the execution state of that process goes from blocked to ready. Wakeups for "regular" event channels involve copying an eight-word message from the sending process to the target process. This message includes both the sending and target process identifiers, the target event channel identifier, the validation level of the sender, and an arbitrary two-word datum supplied by the sender for receipt by the target process. This eight-word message is stored in the ITT between the time that the sending process calls the supervisor to send a wakeup, and the time that the target process, having been woken up, calls the supervisor to retrieve it. A list of these ITT messages queued to a given target process is maintained for each process; the head of this list is in the process APT entry.

Only a fixed number of ITT entries are available. This number is specified at bootload time by the tcd configuration card. The supervisor checks and returns an error indication to a caller if a call to wakeup might overflow the ITT. When a user process causes an ITT overflow, an error message is printed on the bootload console *only* if the previous ITT overflow was not caused by the same user process. Any attempted IPC wakeups which result in an ITT overflow may cripple the system, since user terminal operations, daemon operations, and the message coordinator depend on wakeups.

The tuning of tc_data consists of choosing the sizes of the APT and ITT so that each is as large as needed, but not so large as to needlessly use memory that could be available for paging. For the APT, the choice of the number of APTEs is simple, since it is simply a reflection of the maximum number of users. For the ITT, the choice is more difficult, since there may be spurts of usage. A general rule of thumb is to have *at least* twice as many entries in the ITT as in the APT. As with the AST, you should try not to waste room in the last page of tc_data. This can be accomplished by adding enough ITT entries to fill up the last page. Table 14-2 shows how to figure the length of tc_data. See the segment tc_data.cds for complete details on the layout and content of tc_data.

Table 14-2. Figuring Length of tc_data



TC_DATA size is specified in configuration deck:

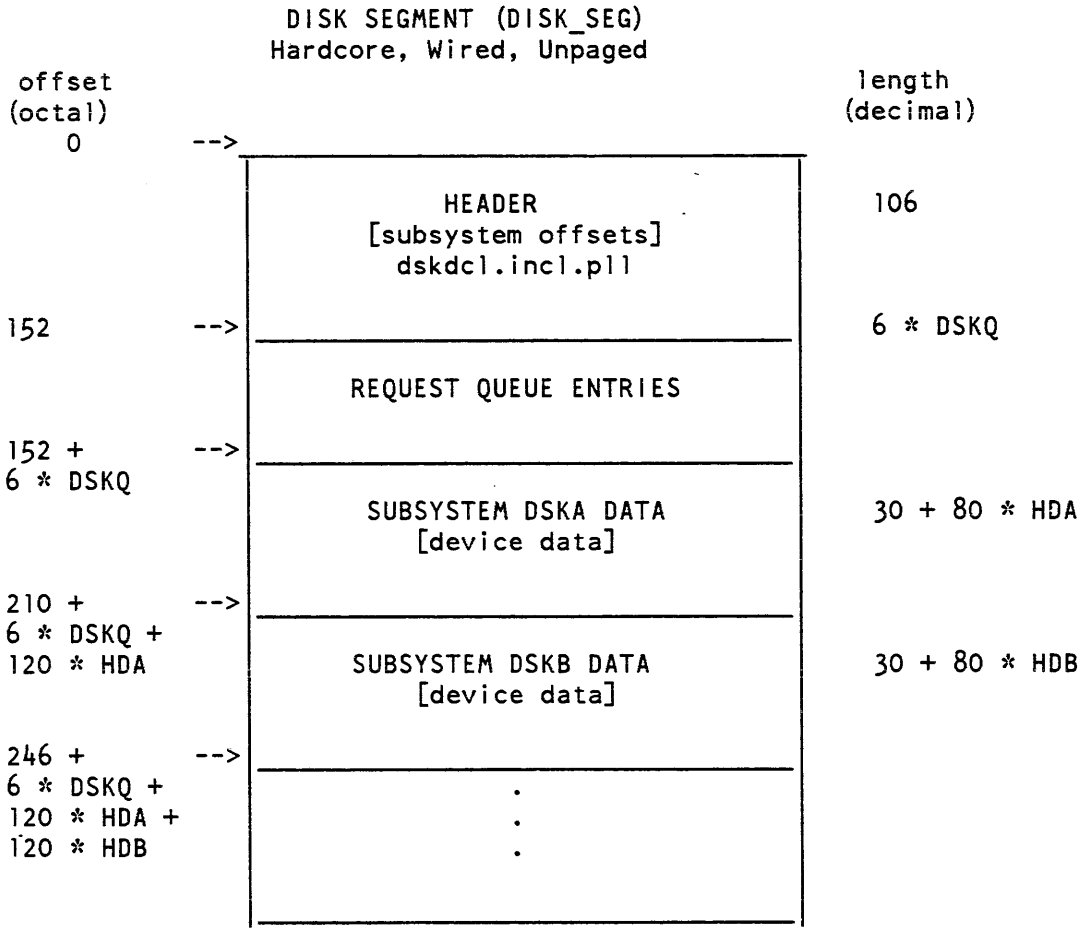
TCD NAPTES NITT

NAPTES = Number of APTEs in APT
NITT = Number of entries in ITT

DISK SEGMENT (DISK_SEG) DATABASE

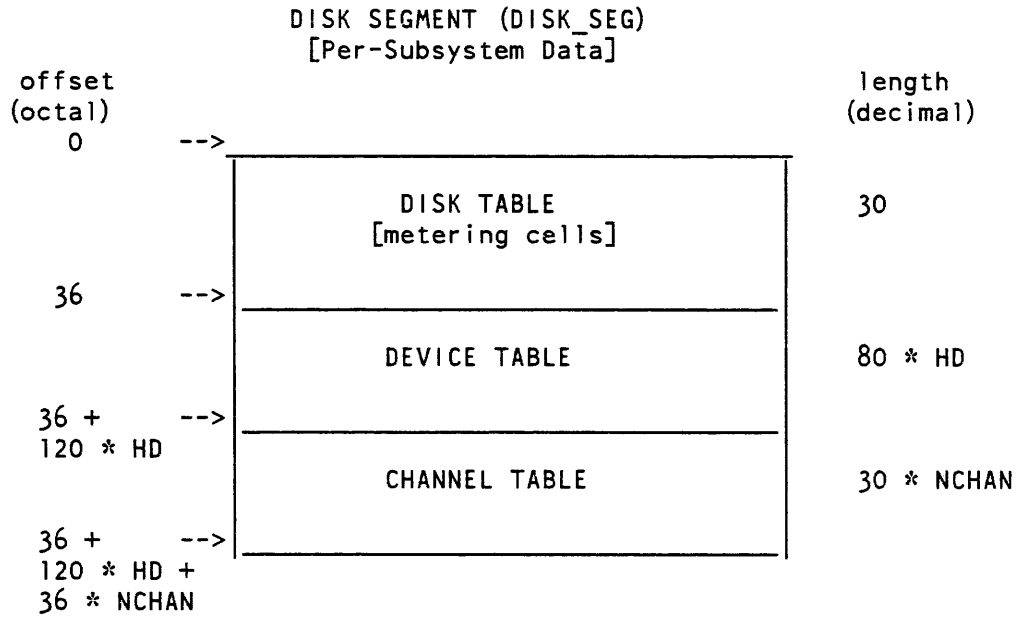
The disk_seg segment is a wired, unpaged database used by the disk control subsystem to access the various disk subsystems of Multics. It contains a 106-word header defining the number of subsystems (among other things), the queues for holding requests for the subsystem, and information for up to 32 disk subsystems. The per-subsystem information includes the various metering cells used by the disk meter commands, the number of logical channels, the number of actual devices, and a per-subsystem lock word. Tables 14-3 and 14-4 show how to figure the length of disk_seg.

Table 14-3. Figuring Length of disk_seg



- DSKQ refers to dskq parameter on parm config card, which specifies maximum number of disk queue entries that can ever be pending
- HDA/HDB refer to highest device number

Table 14-4. Figuring Length of disk_seg



CONFIGURATION DECK (CONFIG_DECK) DATABASE

The config_deck segment is a wired segment that describes the current configuration. As dynamic reconfiguration occurs (i.e., hardware modules are deconfigured and reconfigured), the state tags for these modules in the config_deck segment will change.

Each entry in the config_deck consists of 16 machine words (36 bits per word). The first word (4 characters) contains the ASCII name of the card. The following 14 words contain various fields, one word per field. The last word of each entry contains 14 2-bit type fields corresponding to the 14 field specifiers. These type fields specify the storage type of each field (i.e., ASCII, octal number, decimal number). The remaining 8 bits define the number of valid fields for this card.

The config_deck may be displayed using the print_configuration_deck command, described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

While dynamic reconfiguration of certain hardware modules is possible in Multics, the addition of modules not specified in the config_deck at bootload time is not possible. Thus, setting up the configuration before starting the system requires special attention.

In addition to specifying the hardware configuration, the config_deck also contains cards for setting up various tables (e.g., SST, tc_data, disk_seg, vtoc_buffer_seg, tty_buf, etc.). The sizes of these tables are static once the system is booted, so particular care must be taken to provide optimal values for these entries.

Metering Commands

The groups of commands listed below divide the various metering commands according to the metering database segments from which they extract information. The first group is associated with the SST, the second with the tc_data segment, the third with the disk_seg segment, and the last with the config_deck and all other hardcore databases. A brief description of the function of each command is also shown.

Commands that collect information from the sst_seg segment concerning the management of memory, page control, and segment control:

file_system_meters
: performance information and usage of the page control and segment control subsystems.

flush
: information on page control and system timing.

post_purge_meters
: information collected at post purge time, if post purge is enabled.

Metering commands that collect information from the tc_data segment concerning the management of processes and processors:

alarm_clock_meters
: information about use of simulated timers.

link_meters
: per-process information regarding linker use.

response_meters
: information regarding estimated response time (work classes, scheduling).

system_link_meters
: statistics regarding system-wide linker use.

total_time_meters
: CPU time percentages and average CPU time spent doing various tasks.

traffic_control_meters
: information regarding processes (scheduling).

traffic_control_queue
current state of the eligible queue and work class queues.

work_class_meters
work class parameters and statistics.

Metering commands that collect information from the disk_seg segment concerning the various disk subsystems:

disk_meters
statistics for each disk subsystem (and each device on that subsystem) and information on I/O channel saturation.

disk_queue
subsystem channel activity and pending I/O requests for each disk subsystem.

Metering commands that collect information from the config_deck and all other hardcore segments:

alarm_clock_meters
information on the behavior of the Multics simulated alarm clock.

cache_meters
information on central processor, hardware recoverable, cache memory errors.

command_usage_count
information on the number of times commands are used and the User_id for each invocation.

fim_meters
information on fault processing.

hc_pf_meters
system-wide statistics regarding page faults taken on hardcore segments.

instr_speed
information on CPU speed.

interrupt_meters
information on input/output multiplexer (IOM) channel interrupts.

list_vols
information on currently mounted physical or logical volumes.

meter_gate
information for entries in specified hardcore gates.

meter_rcp
information on devices controlled by the resource control package (RCP).

meter_signal
performance measurements of the signalling mechanism.

print_configuration_deck
displays the current system configuration deck.

print_tuning_parameters
displays the current values of the settable tuning parameters.

system_performance_graph
graphical representation of system-wide statistics.

vtoc_buffer_meters
information regarding the utilization of volume table of contents (VTOC) buffers.

Two metering commands not mentioned above are the channel_comm_meters and system_comm_meters commands, which give statistics on the behavior of the Multics Communication System and the various communications channels. These commands are described in the *Multics Administrator's Manual--Communications*, Order No. CC75.

Metering Design

This discussion gives a few recommendations for writing metering commands and subroutines. Also described here are some of the conventions used in the standard metering commands.

In general, a metering command must serve three functions: extracting the data, arranging it in a useful format, and printing it out. Each of these functions must be considered in writing metering commands.

EXTRACTING METERING INFORMATION

The metering_util_ subroutine, described in the *Multics Subroutines and I/O Modules* manual, Order No. AG93, can be used to extract the metering information from the system segment table (SST) and tc_data hardcore databases. These databases contain global metering data concerning the time the system has been up, the amount of memory configured, the number of processes (users) currently on the system, and the detailed metering data generated by the page control and traffic control programs of the supervisor. The metering_util_ subroutine allows the caller to get this data easily and to reset the data being sampled. In particular, the metering_util_$time entry prints the time the system has been up (or the time since the last reset call) in the standard format used by most of the metering commands (in the format HH:MM:SS, where HH is hours, MM is minutes, and SS is seconds).

Other hardcore databases referenced by the various metering commands (other than the SST and tc_data) are extracted by using the ring0_get_ subroutine.

The following example illustrates just how the metering of the system takes place and how the various metering commands extract the data and display it in a

useful format. This example displays an extract of the page fault handling procedure, page_fault.alm, and a portion of the total_time_meters procedure source code.

```
FROM page_fault.alm:

          .
          .
          .
   page_fault

          .
          .
          .
  include tc_meters
          .
          .
          .
  rccl sys_info$clock_,*  start metering
  staq pds$time_1
          .
          .
          .
  eppbp  tc_data$     restore tcd ptr
  rccl sys_info$clock_,*  meter page fault time
  sbaq pds$time_1     get cpu time for this fault
  adaq bp|cpu_pf_time keep sum of times
  staq bp|cpu_pf_time
  aos bp|cpu_pf_count and count of faults
          .
          .
          .
FROM total_time_meters.pl1:

ttm: total_time_meters: proc;

dcl (tcdp1, tcdp2, sstp1, sstp2) ptr static;
dcl unique fixed bin static init (0);
          .
          .
          .
%include tcm;
%include sst;
          .
          .
          .
  if unique = 0 then call metering_util_$get_buffers
 (unique, sstp1, sstp2, tcdp1, tcdp2, code);
          .
          .
          .
  call metering_util_$fill_buffers (unique);
  call metering_util_$time (unique, meter_time);
          .
          .
```

```
/* Now calculate the page fault information */

  time = tcdp2 -> tcm.cpu_pf_time - tcdp1 -> tcm.cpu_pf_time;
  count = tcdp2 -> tcm.cpu_pf_count - tcdp1 -> tcm.cpu_pf_count;
  if count = 0e0 then ave_time = 0e0;
  else ave_time = time/count;
  pc = time/meter_time;

  call ioa_ ("Page Faults      ^6.2f^12.3f", pc, ave_time);
          .
          .
          .
  if reset_switch then
 call metering_util_$reset (unique);
          .
          .
          .
```

VARIOUS TYPES OF METERING TIME

Several different types of metering time are of interest to users of metering tools. They are:

real time
: the actual time period as measured by a normal clock.

virtual CPU time
: the actual CPU time spent in a process or in the system minus all CPU time spent in page fault, segment fault, bounds fault, process switching, and interrupt processing.

total CPU time
: the actual CPU time spent in a process or in the system including all CPU time spent in page fault, segment fault, bounds fault, process switching, and interrupt processing.

idle time
: the amount of time the system was not running a program on behalf of a normal process. This time can be partitioned into well-defined subsets. This partitioning is of interest to anyone measuring the efficiency of the system.

processor time
: the amount of CPU-seconds the system has accumulated since it was bootloaded. This time is the same as real time for a system that has only one processor, but is guaranteed to be different if the system ever had more than one processor configured.

It is often interesting to print out the percentage of time a given program or programs spend in the system. To do this an appropriate base must be established against which to compare the given quantity. The standard base used by most metering commands is the processor time accumulated by the system. Some commands, however,

base their comparisons against nonidle processor time. Either of these methods is acceptable as long as it is clearly stated in some way what the meters represent.

RESET MECHANISM

Several metering commands allow you to begin metering again at a specified time, usually the time of the call to the command with the -reset control argument specified. This type of metering is quite useful and is generally done with the use of internal static copies of the databases containing the raw metering data. This resetting mechanism never changes the actual meter values kept by the system. Therefore one user's use of the reset mechanism has no effect on any other user who might be using the metering commands at the same time.

Although the internal static technique has proved to be very useful and easy to program, it does not allow you to specify an arbitrary time interval during the day for which metering is desired. To do this, a periodic sampling of the metering data must be done. Such sampling of a few values in the SST and tc_data segments is in fact done by the answering service program as_meter_ at accounting update time (although this data is currently unavailable to general users). as_meter_ stores this information in >sc1>stat_seg. It is normally processed and printed each day by the crank.

STANDARD CONTROL ARGUMENTS

Some control arguments are standard for metering commands. They function as described below. If any new metering command is written that performs one of the features controlled by these arguments, it should be designed so that it can use these same control arguments.

-reset, -rs
: resets the metering interval for the invoking process so that the interval begins at the last call with -reset specified. If the -reset control argument has never been given in a process, it is equivalent to having been specified at system initialization time (i.e., the metering interval begins at system initialization time).

-report_reset, -rr
: generates a full report and then performs the reset operation.

-brief, -bf
: generates an abbreviated report (i.e., some metered data is not printed on the report produced by invoking the command).

The default report, which is generated when any metering command is given with no control arguments, prints out all of the metering data normally available. The metering interval, by default, begins at system initialization time. Those metering commands that accept the -reset argument display the length of the total metering interval as their first line of output, in the format HH:MM:SS, where HH is hours, MM is minutes, and SS is seconds.

CPU Time Metering

There are several different viewpoints from which system performance can be examined. Each of these viewpoints tells you something different about system performance. One of these viewpoints is the allocation of CPU time: the proportion of CPU time being spent on useful work rather than on overhead or idling. This subsection discusses that viewpoint, and tells you what you can and cannot learn by examining the allocation of CPU time.

Briefly, CPU time can be divided into three categories: idle, overhead, and applied.

Idle time is that which is unused because the load, over some time interval, is much lighter than the system's capacity.

Overhead time is that which is spent performing system functions connected with the sharing of the system among a number of users.

Applied time is that which is spent running user's application programs, or performing system functions on behalf of those application programs.

One possible objective of configuration and tuning is to maximize applied time and minimize overhead time. This should be done under conditions of heavy load - that is, when idle time is approaching zero. (When the load is light, tuning is not critical, and adjustments made then might not produce optimal performance under heavy load.)

Maximizing applied time tends to maximize throughput. It does not necessarily optimize response time. At first, cutting down overhead will have a beneficial effect on response time. But in the end, eliminating the last few percentage points of overhead can only be done at the expense of response time: favoring long-running computations over short interactive ones, thereby reducing the overhead associated with frequent process switching and page replacement.

The total_time_meters (ttm) command displays the total CPU time used by the system, broken down into over a dozen different categories. A detailed description of the output of the ttm command is given in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64. The discussion below gives some general guidelines for understanding and interpreting the output.

For the sake of efficiency, the set of time meters maintained, and their accuracy, are both determined by the implementation. Thus, they do not map simply into the three major categories: idle, overhead, and applied. For example, there are five meters with the word "idle" in their names. Three of these measure time wasted in idling because of three different kinds of system bottlenecks, (MP, Loading, and Work Class Idle); these belong in the overhead category. The other two (NMP and Zero Idle) are true idle time caused by light load. They are kept separate because one of them (NMP Idle) indicates that users on a lightly loaded system were delayed by system bottlenecks.

Some of the overhead meters displayed by ttm include applied time spent performing system functions on behalf of users' application programs. It is not possible to separate this time from true overhead time. For example, all time spent on handling page faults is measured by a single meter. But some page faults are due to the sharing of main memory by all the users, while others are due to the automatic memory management provided to users by Multics. The latter would occur even if the user were alone on the system, and would, on other operating systems, be replaced by explicitly programmed file I/O, charged to the application program.

Thus, applied time consists of the virtual CPU time reported by ttm, plus some fraction (that can only be estimated) of the overhead times. On very well balanced and well tuned configurations, virtual CPU time has been seen to approach (or even exceed) 80%. On such systems, it is estimated that applied time is 85% to 90% and true overhead is only 10% to 15%. As configuration and tuning values depart from the optimal, all of the increases in reported overhead times should be attributed to true overhead.

The ttm command displays the time allocations both as a percent of total system time and as a percent of non-idle time. The purpose of the latter is to allow meters taken during periods of light load to be compared to standard values without requiring a lot of mental arithmetic. (For example, if the normal virtual CPU time percent is 70, then 70% non-idle virtual CPU time during a period of light load is a sign of good performance, even though virtual CPU time as a percent of total system time is much lower due to the high percent of idle time.)

However, it cannot be emphasized too strongly that meter readings from periods of light load must be interpreted cautiously. Abnormal readings during periods of light load are certainly cause for further investigation. However, decisions about tuning adjustments or configuration changes should not be based on measurments taken under light load. The system's behavior is different under light and heavy loads, and adjustments made under light load are unlikely to be optimal under heavy load.

TUNING

This subsection gives an overview of scheduling and how it relates to performance, summarizes the available tuning commands, and describes each of the tuning parameters. It also describes the effects of selected changes to certain tuning parameters under various circumstances.

Scheduling

The following discussion defines some terms and gives a generalized overview of scheduling and an idea of how the number of eligible processes affects thrashing and thus optimum performance of the system.

A *blocked process* is one that is awaiting some event, such as a timer to go off or a user to type a line of input.

A *ready process* is one that is not blocked. It has computation to do and is awaiting its turn to run on a CPU.

An *eligible process* is a ready process that the scheduler has selected for running in the near future. The scheduler tries to keep the number of eligible processes at a value that optimizes system performance.

A *loaded process* is one that is eligible and has the first page of its descriptor segment and the first page of its PDS loaded into memory and wired there. These two pages must be in memory before a process can run; they cannot be brought in by page faults because they are used when a process takes a page fault.

A *waiting process* is a process that is eligible and loaded but cannot use a CPU because it requires some other resource as well. Waiting is distinct from the blocked state. Wait events occur in a very short time (typically measured in milliseconds). Simple wait events are: page read, VTOC read, and page table lock.

A *running process* is one that is eligible, loaded, and is actually running on a CPU.

When a running process takes a page fault, the system first takes steps to have the page read into memory. Then it attempts to find another process that can be run immediately. If it fails to find such a process, the CPU goes idle and some amount of CPU time is wasted.

A process that can run immediately is one that is eligible, loaded, and not waiting. Thus, one criterion affecting the optimal number of eligible processes is that there should be enough of them to keep to a minimum the times they will all be waiting simultaneously (which forces the CPU to go idle).

If the scheduler fails to find a runnable process and it determines that there should be more eligible processes, it makes another one eligible, but this is of no immediate benefit. It probably will be two page wait times before the newly-eligible process is loaded and runnable, so the CPU still has to go idle.

Making a process eligible implies a short-term commitment of CPU resources. The process is allocated a certain amount of CPU time (an eligibility quantum) and it is given this amount of CPU time before it loses eligibility.

There are algorithms for choosing which processes to make eligible, for deciding what size quanta to give them (quanta vary between processes and vary over time for one process), and for determining priority among eligible processes. The significant aspect of eligible processes is that they are given frequent, short slices of CPU time, interrupted by their own page faults and by interrupts signalling the completion of input/output for some other process.

If an eligible process uses its quantum of CPU time without completing its computation and going blocked, it is rescheduled. It loses eligibility, it is returned to the ready state and some other process is made eligible, loaded, and given a chance to run. The real time interval until the rescheduled process again becomes eligible is generally much larger then the interval between the getwork schedulings it receives while eligible.

The term "working set" can be used loosely to refer to all of the pages that a process is currently using. Early in a process's eligibility quantum, it takes page faults frequently since much of its working set is not in memory. Later in the quantum, it is hoped that it takes page faults less frequently, because most of its working set has been brought into memory and will remain there for the rest of the quantum.

Eligible processes tend to compete with each other for pages of memory. Thus, a second criterion affecting the optimal number of eligible processes is that there should be few enough of them so that there is enough memory to hold all of their working sets. If there are too many eligible processes (or not enough memory), they fight each other for memory, continually forcing out each other's pages, taking page faults at a high rate, and accomplishing very little useful work. This condition is known as *thrashing*.

There are two tuning parameters, max_eligible and min_eligible, used by the scheduler to control the number of eligible processes. As their names suggest, they specify the maximum and minimum values.

To summarize the discussion so far: the number of eligible processes must be high enough to keep the CPU from going idle too frequently, but there must be enough memory to hold the working sets of the eligible processes and avoid thrashing.

This assumes a mode of operation in which most of the pages in the working sets of ready processes get forced out of memory before the processes become eligible again, requiring that those pages be brought back in at the beginning of the next eligibility. If there is enough memory so that at least some of those pages remain in memory during the intervals between eligibility, that cuts down on the total number of page faults taken, reduces the rate at which eligible processes take page faults, and thus reduces the number of eligible processes that are needed in order to ensure that when one of them takes a page fault there is usually another one that is able to run immediately.

Thus, there are three regions on the curve of performance versus memory size: in the low memory region, there is thrashing; in the intermediate memory region, the system is not thrashing but there are still a lot of page faults and the number of eligible processes must be kept fairly high to avoid time wasted in idling; in the high memory region, the page fault rate is lower, the number of eligible processes need not be so high, and all of the CPU time that would be consumed by the overhead and idling associated with the higher paging rate can be turned into useful computation.

A system with too little memory continues to function if max_eligible and the maximum number of users are set low, but much CPU time is wasted in idling and paging overhead.

Figure 14-1 shows the shape of the performance curve. It relates throughput to the ratio of memory to CPU power. It shows the three regions discussed above: thrashing, high paging, and low paging. The regions are delimited by points A and B. Memory/CPU ratios at or below point A are very likely to cause thrashing. Ratios between A and B are usually acceptable but fall short of optimal performance. As ratios proceed beyond B into the low paging region, the cost effectiveness of adding memory becomes increasingly questionable.

The values on the performance axis are percents of applied CPU time, as defined earlier under "CPU Time Metering."

The values on the MEM/MCPU axis are megabytes of memory per MCPU. (MCPU is a unit of CPU power roughly equivalent to one million instructions per second). MCPU is defined later under "Configuration Guidelines."

As noted under "Suggested Values and Guidelines," the optimal memory/CPU ration depends on the memory intensiveness of the application mix at the site. Variations of over 25% in the values for points A and B have been observed between some sites.

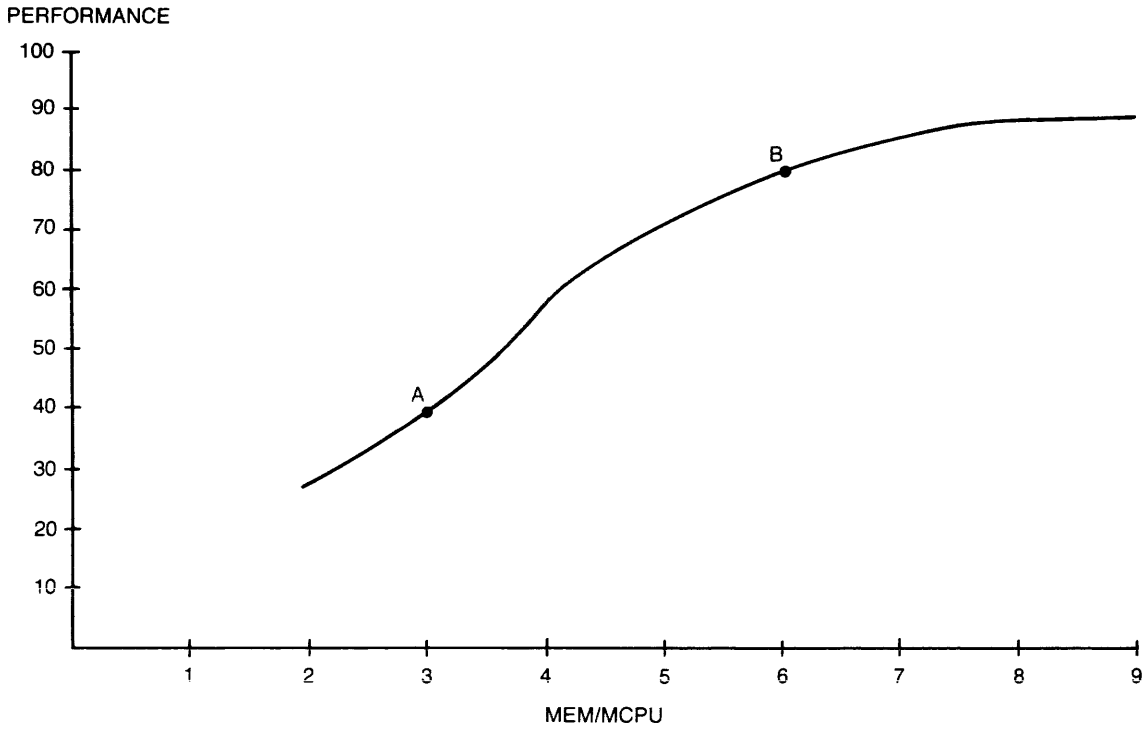


Figure 14-1. Performance vs Memory

Tuning Commands

The following tuning commands display and adjust the values that manage queueing, paging, the traffic controller, scheduling, processor management, dispatching, multiprogramming, and multiprocessing:

change_tuning_parameters
changes the value of several tuning parameters within the system.

define_work_class
assigns CPU time to work classes (scheduling).

print_tuning_parameters
prints the current values of various tuning parameters within the system.

set_work_class
temporarily moves a process or a set of processes from one work class to another without installing a new master group table (MGT -- see below).

tune_work_class
temporarily sets or changes scheduling parameters for a single work class.

work_class_meters
displays currently defined work classes.

The master group table (MGT) is the segment maintained by the system administrator that contains work class information. It can be edited with the ed_mgt command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64).

The installation_parms segment is another table maintained by the system administrator containing site-settable system parameters. It is edited by the system administrator with the ed_installation_parms command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64).

Tuning Parameters

The following descriptions of tuning parameters give the meanings of each parameter, plus the units used for each, the data types used in the change_tuning_parameter command, and the default values. Note that the data types decimal number of seconds and decimal number can be used to express fractional values (e.g., 0.001 for one second, .5 for a multiplier of one-half, etc.). For each parameter, the short name, if any, is given in parentheses. Any of the parameters described below (except for max_max_eligible) can be changed via the change_tuning_parameters (ctp) command. All of the tuning parameters are kept in the tc_data and sst_seg databases.

tefirst

is the amount of CPU time (i.e., initial quantum or time slice) given a process after an interaction. When a process changes from the blocked state to the ready state, is made eligible, and loaded, it is given tefirst seconds of processor time before being made ineligible and placed in the appropriate work class queue. The initial value of tefirst is set by the schd configuration card, which is fully described in Section 7. The tefirst parameter shows the number of seconds, given in decimal number of seconds. The default is 2.0 seconds.

telast

is the amount of processor time given a process for all subsequent executions. The initial value of this parameter is set by the schd configuration card (see Section 7). The telast parameter shows the number of seconds, given in decimal number of seconds. The default is 2.0 seconds.

timax

is the maximum value that ti (time since interaction) can assume for an interactive process. Once a process has used ti seconds of CPU time since its last interaction, the scheduler no longer increments this value; i.e., all such processes are scheduled round-robin, but after processes that have ti less than timax. The initial value of this parameter is set by the schd configuration card (see Section 7). The timax parameter shows the number of seconds, given in decimal number of seconds. The default is 8.0 seconds.

priority_sched_inc (psi)

is the priority scheduler increment. Once a process has been blocked for this number of seconds, any wakeup it receives will be treated as if an interaction occurred. In other words, this parameter can be used to make processes that block always appear to be interactive (thereby getting favorable scheduling). Wakeups sent by the tape or tty interrupt handlers are interactive; wakeups caused by timers or explicit user action are not interactive unless the target process has been blocked for longer than priority_sched_inc. Thus, when priority_sched_inc is set to .125 second, an absentee process that pauses for longer than this appears to be interactive to the scheduler. The priority_sched_inc parameter shows the number of seconds, given in decimal number of seconds. The default is 80.0 seconds.

min_eligible (mine)

is the minimum number of eligible processes. The scheduler attempts to keep at least this many processes eligible. If less than min_eligible processes are eligible, the scheduler makes any non-eligible ready process eligible without regard to working set. If the value of the post_purge parameter is off, min_eligible becomes meaningless, since the working sets of all processes are zero. The initial value of this parameter is set by the schd configuration card (see Section 7). The min_eligible parameter shows the number of processes, given in integers. The default is two.

max_eligible (maxe)

is the maximum allowed number of eligible processes. The scheduler does not (usually) allow more than this number of processes in the eligible queue. Realtime processes can cause this value to be exceeded and idle processes do not count against this number. The initial value of this parameter is set by the schd configuration card (see Section 7). The max_eligible parameter shows the number of processes, given in integers. The default is six.

max_max_eligible
is the maximum value that max_eligible can take during the current bootload of Multics. This is necessary because of the sharing of ring-0 stacks. Max_max_eligible ring-0 stacks are created at bootload time, and this number cannot be exceeded. Thus, there can never be more than max_max_eligible processes eligible, including realtime processes. The max_max_eligible parameter shows the number of processes (expressed as an integer). A value of zero for this parameter (on the schd configuration card) will cause its default value to be max_eligible plus ten. The max_max_eligible parameter cannot be changed by the change_tuning_parameters command. It can only be changed by using the schd card in the config deck (see Section 7).

max_batch_elig (maxabs)
is the maximum number of absentee processes that can be eligible at one time. A value of zero causes this restriction to be ignored. This parameter is also ignored if the scheduler is in deadline mode.

working_set_factor (wsf)
is the multiplier for the working set computation. Once an estimated working set has been computed for a process, it is multiplied by this value. The initial value of this parameter is set by the schd configuration card (see Section 7). The working_set_factor parameter shows the fractional multiplier, given in decimal numbers. The default is 1.0.

working_set_addend (wsa)
is the addend for the working set computation. Once an estimated working set has been computed for a process, and multiplied by the working_set_factor, this value is added to the result. The working_set_addend parameter shows the number of pages, given in integers. The default value is zero.

deadline_mode (dmode)
tells the scheduler whether to operate in deadline mode or percentage (interactive) mode. The value of off implies percentage mode; on implies deadline mode. If the deadline_mode tuning parameter is on, processes in non-realtime work classes are scheduled according to the deadlines and quanta specified for their work classes. If off (which is the default), then non-realtime work classes are scheduled according to the percentages specified for their work classes. The answering service sets this parameter at shift change time to the value specified by the master group table (MGT). For more information, see the ed_mgt command in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

int_q_enabled (intq)
controls the use of the interactive queue. A value of on implies that there is an interactive queue; processes are placed in it after an interaction instead of in their designated work class queue. A value of off disables the interactive queue; processes are placed directly into their designated work class queue after an interaction. If the int_q_enabled parameter is set to off when the scheduler is in percent mode, the specified percentages are more closely observed, but response is degraded. The default is on.

post_purge (pp)

controls the post purging function. If on, the pages of a process are moved to the head of the list for the page removal algorithm when the process loses eligibility. This switch controls the working set computation as well. If off, the working set computation is disabled and the working_set_factor and working_set_addend parameters have no effect. The page table lock is locked during post purging. If this switch is on, the per-process page trace buffer does not describe the location of the faulting instruction. This affects the page_trace command output. The default is off.

pre_empt_sample_time (pest)

is used to set the maximum clock time between invocations of traffic control. This is implemented by limiting the value of the timer register when a process is given a CPU to:

```
pre_empt_sample_time X (number of CPUs online)
```

With parameters gp_at_notify and gp_at_ptlnotify set to off (the default value for both parameters), traffic control checks for higher priority processes to run at least once every pre_empt_sample_time. It shows the number of seconds, given in decimal number of seconds. The default is .040 seconds.

gp_at_notify (gpn)

controls the actions of the scheduler upon notification of the occurrence of a system event (e.g., page read). If on, the process is forced onto a processor. The default is off.

gp_at_ptlnotify (gpp)

controls the action of the scheduler upon notification of a global page table lock. If on, the process is forced onto a processor. The default is off.

process_initial_quantum (piq)

controls the first quantum given to a newly created process. Higher values of this parameter improve the responsiveness of newly created processes. This quantum is in effect until the process goes blocked or exceeds the quantum. A reasonable value is the average virtual CPU time reported in the first "ready" message for a typical user. This provides fast execution of the start_up.ec. The process_initial_quantum parameter shows number of seconds, given in decimal number of seconds. The default if 2.0 seconds.

quit_priority (qp)

is used to recompute the ti of a process after a quit. The new ti value will be its old ti value times the quit_priority. This parameter can be used administratively to control the action of the scheduler when a user quits. Interactive users sometimes quit and restart in order to appear interactive during a long-running command. When quit_priority is 0, users appear interactive. If the value is 1, the users' scheduling priority is unchanged, and if it is 2, the users' priority is actually degraded. This parameter is usually set at 0, but it can be increased if quit/restarts are a problem at a site. (The output from tty_meters shows the frequency of quits on the system.) The quit_priority parameter shows the fractional multiplier expressed in decimal. The default value is zero.

notify_timeout_interval (nto_delta)
is the allowed interval for a system notify to occur. The scheduler expects any system event to occur within this time and complains if it does not occur. The notify_timeout_interval parameter shows the number of seconds, given as a decimal number of seconds. The default value is 30 seconds.

notify_timeout_severity (nto_severity)
is the severity of the occurrence of a notify timeout. Values can range from 0 to five and control the action taken when a notify timeout occurs. The notify_timeout_severity shows the number of the severity, given in integers. The default value is three. The following table shows what actions are taken for each value. The include file syserr_constants.incl.pl1 contains a definition of these values.

Value	Action
-1	don't print message, don't log message
0	print message, log message
1	print message, log message, activate alarm, return to BCE
2	print message, log message, activate alarm, terminate process
3	print message, log message, activate alarm (normal case for nto)
4	try to log message, if log full, print message
5	try to log message, if log full, discard message

write_limit (wlim)
used by page control to determine the size of page flush operations. Page flushes safeguard modified data by writing pages to disk at regular intervals. The number of pages which page control will write in a single page flush operation is equal to approximately 1/2 the value of the write_limit parameter. The maximum number of disk I/O operations that could ever be pending is the same as the value of the dskq parameter on the parm config card. (Refer to Section 7.)

gv_integration
controls the granularity of control for work class governing. It is approximately the interval over which governing is enforced. For example, if it is set to 3600. (one hour), a work class with a limit of 10% is limited to 10% of available CPU time within any given one hour period. The gv_integration parameter shows number of seconds, given in decimal number of seconds. The default value of this parameter causes governing to be instantaneous; that is, the governing is four times the value of telast, and it is the smallest value to which gv_integration can be set.

realtime_io_priority (io_prior)
controls whether IOI interrupts cause high priority. If on, an interrupt from an IOI device causes the process owning the device to get a high priority CPU quantum of short duration immediately after the interrupt. If off, an interrupt from an IOI device is treated the same as receipt of remote terminal input. The default value is off.

realtime_io_deadline (io_deadline)
is the time interval after an IOI interrupt in which the process will be considered for high priority by traffic control. This value is specified in a decimal number of seconds. The default value is 0.0 and indicates the process is to receive high priority immediately after receipt of the IOI interrupt. This parameter has no effect if realtime_io_priority is off.

realtime_io_quantum (io_quantum)
is the length of the high priority quantum (burst), expressed in virtual CPU seconds. After this time has expired, the process is given its normal traffic control priority. This value is specified in a decimal number of seconds. The default value is 0.005 (5 milliseconds). This is sufficiently long to initiate another tape I/O and do a small amount of processing, but not long enough to impact interactive users adversely. This parameter has no effect if realtime_io_priority is off.

dirlock_writebehind (dirw)
causes modified directory pages to be written from main memory whenever a directory is unlocked. Specifying this parameter increases the safety of the system at some cost in increased paging. Process directory pages are not written.

Here is some sample output from an invocation of the print_tuning_parameters command:

```
Current system tuning parameters:

tefirst                          0.5   seconds
telast                           1.    seconds
timax                            8.    seconds
priority_sched_inc              80.    seconds
min_eligible                     2.
max_eligible                    20.
max_batch_elig                   0
working_set_factor               0.5
working_set_addend               0
deadline_mode                  off
int_q_enabled                   on
post_purge                     off
pre_empt_sample_time             0.04  seconds
gp_at_notify                   off
gp_at_ptlnotify                off
process_initial_quantum          2. seconds
quit_priority                    0
notify_timeout_interval         30. seconds
notify_timeout_severity          3
write_limit                    335
gv_intergration                  4. seconds
realtime_io_priority            on
realtime_io_deadline             0. seconds
realtime_io_quantum              0.005 seconds
dirlock_writebehind             on
```

Selected Changes to Certain Tuning Parameters

The following discussion describes the effects of selected changes to certain tuning parameters under various circumstances.

The max_eligible, min_eligible, post_purge, working_set_addend, and working_set_factor tuning parameters are used to control the number of processes that are multiprogrammed (i.e., allowed to compete for pages of memory). If post_purge is on, then each time a process loses eligibility, page control uses information about the process' recent paging behavior to estimate the size of the process' working set (paging behavior information is available to page control in the process definition segment). This estimate is then multiplied by the wsf, and the result is added to wsa to arrive at a final value for the working set of the process. When post_purge is off, the estimated working set is always zero and additional information will be recorded in the per-process trace table on every page fault (see the description of the page_trace command in the *Multics Commands and Active Functions* manual, Order No. AG92).

The scheduler uses the following criteria to determine whether to make an additional process eligible:

1. Another process may be made eligible if fewer than min_eligible processes are currently eligible.

2. Another process may not be made eligible if more than max_eligible processes are currently eligible.

3. When the number of eligible processes is between mine and maxe, another process may be made eligible if the sum of the working sets of the eligible processes and the working set of the process being considered for eligibility is less than the number of pages of pageable (nonwired) main memory.

Checks 2 and 3 are ignored for a process in a realtime work class if that process' deadline has passed.

Thus, if you were to make any of the following changes:

```
increase min_eligible
increase max_eligible
decrease working_set_factor
decrease working_set_addend
```

it would tend to increase multiprogramming, and therefore reduce MP idle and increase paging (possibly causing thrashing if there is too little memory). Conversely, decreasing mine or maxe, and increasing wsf or wsa, would have the opposite effect.

The notify_timeout_interval and notify_timeout_severity parameters control the action the system takes when a notify timeout occurs. (A notify timeout occurs when a process has been waiting for some event for longer than the notify_timeout_interval, and is generally symptomatic of some programming or hardware error.) With the

tuning parameters set at their default values, when a notify timeout occurs, the process is taken out of the waiting state and a message is printed on the bootload console. The notify_timeout_severity parameter controls the "severity" of the call (made by the system program syserr) that prints this message. Useful values for notify_timeout_severity are:

-1 no call to syserr is made, so no record of the notify timeout is printed or logged (useful for temporarily masking a problem).

1 the system is crashed after the message is printed (useful for debugging by system programmers).

3 the message is printed but no further action is taken.

The gp_at_ptlnotify and gp_at_notify parameters are used to control the actions of the get_processor function of the scheduler. In all cases, get_processor attempts to find an idle processor on which to run the notified process. When notifying processes waiting for the page table lock, if gp_at_ptlnotify is on, then an attempt is made to preempt a lower priority process (one deeper in the eligible queue). When processes waiting for other events are notified, the attempt to preempt a lower priority process is made only if gp_at_notify is on. If gp_at_ptlnotify is on, paging throughput increases at the cost of preemption overhead. If gp_at_notify is on, I/O throughput increases at the cost of preemption overhead. The effect of setting these parameters "on" is improved performance for processes that take page faults frequently, at the expense of the considerable overhead of the preempt mechanism.

The tefirst, telast, and timax tuning parameters are irrelevant in deadline mode. If the scheduler is in percent mode, however, the initial quantum awarded by the scheduler after an interaction is tefirst. At all other times, the scheduler awards a quantum of telast. In percent mode, processes within a work class are sorted either by the amount of CPU time used since the process has interacted, or by timax, whichever is less. Thus, timax sets a limit on the depth in its work class queue to which a process can sink. The following are examples of the use of various settings of tefirst and telast in percent mode:

tefirst	telast	motive
1.0 sec	0.5 sec	first trial setting
0.5	0.25	better trivial response
2.0	0.5	better response for commands using 1 to 2 seconds
1.0	2.0	better response for long-running commands

The tefirst, telast, and timax parameters can be set by using a schd card in the config deck (see Section 7 for more information).

The realtime_io_priority, realtime_io_deadline, and realtime_io_quantum tuning parameters jointly control the traffic control priority of a process which has received an interrupt from an IOI device (usually a tape drive). If realtime_io_priority is off, the parameters realtime_io_deadline and realtime_io_burst have no effect. The effect of turning realtime_io_priority on is improved throughput for tape I/O-bound

processes (such as the hierarchy and volume dumpers). This is accomplished by treating a process which has just received an interrupt for an IOI device as a realtime process for one eligibility quantum after receipt of the interrupt. The tuning parameters realtime_io_deadline and realtime_io_quantum control that eligibility time.

SUGGESTED VALUES AND GUIDELINES

Metering Output Values

The following paragraphs give some guidelines to be used when evaluating output from metering commands. Also shown are some threshold values to watch for in the metering output. Guidelines and values are displayed per metering command.

disk_meters

Lowest possible per-system Average Time Between (ATB) I/O:

$$\frac{T + R}{P} + \frac{S}{\text{minimum } (L, D, E)}$$

where:

```
P = # physical disk channels
L = # logical disk channels
D = # disk spindles
E = average number of eligible processes
T = transfer time for a Multics record/VTOCE (average)
R = average rotational latency of the disk drives
S = average seek time per drive
```

There are I/O bottlenecks if:

A) % busy > 50 * # logical channels for the subsystem.

B) ATB I/O approaches the limit given by the above formula.

C) page wait times are > 70 msec. (Values of 50 are typical; values around 30 are approaching minimum.)

D) run lock (count) > 10% of call locks (count).

E) allocations (%waits) > 2%.

F) seek_distance > .33 * # cylinders/disk.

G) one pack has > 2 * (average # drives) of the I/O traffic.

Other factors:

A) # errors > 0 indicates hardware problems.

B) use of alternate tracks increases the time to fetch certain records.

C) root logical volume overload. RLV contains all directories and system libraries. Probably should not contain process directories.

disk_queue (dq)

There are I/O bottlenecks if:

A) # connects in last channel > 50% of # connects in first channel.

B) disk queues always contain entries.

C) one pack shows up continually in the disk queue.

file_system_meters (fsm)

$$\text{Average Time Active} = \frac{(\#\ \text{ASTEs}) * (\text{meter interval})}{\#\ \text{needs}}$$

The ASTE pool sizes are bad if:

A) grace time < 200 seconds.

B) AST locked > 50%.

C) AST lock waiting > 80%.

D) ATB claim runs < .01 minutes.

E) grace times for ASTE pools grossly unbalanced.

Memory lap time should be no less than 10 seconds. Values between 20 and 30 seconds are better.

interrupt_meters (intm)

A) attempt to balance FNP channel interrupts.

B) watch for 1 channel taking > 2 * average number of interrupts.

list_vols

A) spread process directories to as many volumes as possible.

B) balance the remaining space between logical volumes.

post_purge_meters (ppm)

A) thrashing percentage should be < 2% if quanta < 2 seconds.

B) thrashing percentage should always be < 5%.

total_time_meters (ttm)

A) if page faults (avg) > 3000 microseconds, then I/O bottleneck.

B) if segment faults (%) > 6%, then bad ASTE pool sizes in SST.

C) MP idle is controllable by tuning parameters maxe and/or wsf/wsa.

D) the objective in tuning is to maximize virtual CPU time. Values approaching 80% can be achieved on systems having sufficient memory.

E) getwork (%) should not exceed about 5%.

F) both page faults % and interrupts % should be < 15%.

G) idle times should approach zero on heavily loaded system.

traffic_control_meters (tcm)

A) if # wait other + # wait page is high, consider turning gp_at_notify off.

B) response time should be < 1 second.

C) # notify timeouts should be zero.

D) # schedulings/interaction should be < 2; otherwise increase tefirst.

E) # wait PTL high, consider turning post_purge off, as PT lock is locked during post purging.

traffic_control_queue (tcq)

A) the interactive queue should be empty or small. If it always contains entries, then it's probable that no processes from the work class queues will ever run, or will run very infrequently.

B) tssc of processes in interactive and work class queues should be small.

vtoc_buffer_meters

A) configure 64 VTOC buffers ("parm vtb 64." configuration card).

check_cpu_speed

Use the check_cpu_speed command to verify that all processors are running with cache and associative memories enabled. Loss of an associative memory can drastically alter the performance of the system. If the hardware diagnoses an associative memory problem, it disables that associative memory WITHOUT WARNING. Likewise with cache memory problems.

meter_gate (mg)

Use the meter_gate command to determine if there are "hot" gate entries (e.g., hcs_ entries). This can give an indication of the type of activity on the system.

system_performance_graph (spg)

The spg output can be useful as it presents a graphical representation of several of the values contained in other meters.

SST Size Guidelines

An SST that is too small can degrade system performance by 50%, whereas an SST that is too large, but less than 10% of main memory, cannot degrade performance by more than 5%. That is, it is better to have an SST that is too large than too small. A rule-of-thumb is to make the SST big enough so that all permanent wired storage does not exceed 10% of available main memory. Remember, all hardcore segments and all ring-0 segments of all processes (DSEG, PDS, KST) are always active, and thus occupy ASTEs permanently.

Configuration Guidelines

The following configuration guidelines are presented here for two reasons. First, they are intended to emphasize the importance of maintaining a good balance between CPU power and paging hardware (memory, disk channels, and disk arms) and to give some guidance for achieving this balance. Second, they are intended to give an estimate of how many users a given, well-balance configuration can be expected to support.

It must be emphasized that the numbers in these guidelines are subject to significant variation, depending on the nature of the users. Users executing simple requests separated by long think times place a lighter load on a system than users whose requests involve significant amounts of computation and paging. Compute-bound applications require a greater proportion of CPU capacity in a configuration, while

applications with large working sets require a greater proportion of paging capacity (main storage, disk channels, and disk arms). Finally, the number of users that any configuration will support is partly a function of how much delay the users will tolerate, or conversely, how much they are willing to pay for the excess capacity needed to provide them with quick response.

To the extent that a user community varies from the norm in any one of the above respects (overall load, compute bound, working set size, and tolerance for delay), the following guidelines will not accurately describe the configuration requirements. These guidelines are based on observations of several of the current Multics sites. "The norm" is defined as a user community to which these guidelines apply. Variations have been observed between sites of up to 25% in the optimal CPU/memory ratio, and up to 50% in the number of users that a given configuration will support. The former is due to variations in the CPU intensiveness vs. working set size of the applications being used at the site. The latter is due partly to variation in the total processing load generated by the applications and partly to variation in the users' tolerance for delay.

It is useful to take these variations into account in capacity planning, by thinking of the "users" in the guidelines below as "standard load units." Determine, by observation, what fraction or multiple of a standard load unit the average user at your site imposes on the system. If you discover that your average user is worth 1.5 standard load units, then to increase your capacity from 100 users to 150 users, you would have to change from a 150 load unit system to a 225 load unit system, using the guidelines below.

These guidelines are based on observations originally made on systems using Level 68 CPUs. They have been revalidated using observations from the newer DPS 8 CPUs. CPU speeds are typically described in units of MIPS (Million Instructions Per Second). The MIPS rating of a CPU will depend heavily on the instruction mix used to measure it. Some instructions move one word from memory into a register; others move up to one million bytes from one set of memory locations to another. In one second, a CPU will obviously be able to execute many more of the former than of the latter. An instruction mix used to measure a CPU's speed - no matter how fairly it is chosen or how consistently it is applied - is essentially arbitrary. It is not unusual to observe variations of 10% or more between the effective speeds of the same model of CPU at different sites, because of differences in the instruction mix used by applications at the sites.

To avoid causing confusion and misunderstanding by using MIPS in an inaccurate way, we have defined a new unit of CPU speed: MCPU (Multics CPU). It is equal to the processing power of one Level 68 CPU (with cache). A Level 68 CPU has been measured at slightly under 1 MIPS, using one arbitrarily chosen instruction mix. The several models of DPS 8M CPU have been measured at various fractions and multiples of 1 MCPU: 0.95 (DPS 8/52M), 1.3 (DPS 8/62M), 1.46 (DPS 8/70M - early model), 1.64 (DPS 8/70M - current production model), and 1.9 (DPS 8/70M - current production model with 32K cache).

The CPU requirements in the following guidelines are expressed in terms of MCPU.

1. Each 1 MCPU, in a suitable configuration of other hardware, will support about 40 logged in users. However, this number varies from a low of 30 to a high of 60, as indicated by the rules below. These numbers are at the upper end of system capacity; users will be aware of a heavy load, although response will be tolerable.

2. Each 1 MCPU needs a minimum of 4MB of main storage to operate effectively. Increasing main storage beyond the 4MB minimum, in 1MB increments, will increase capacity by up to eight users for the first increment (from 40 to 48) and by successively fewer users for each additional 1MB increment (six, four, and two), for a 60 user/MCPU capacity at 8MB/MCPU.

   The capacity increase resulting from additional memory occurs because adding memory reduces paging overhead, making more CPU time available to users. It has been observed that a virtual CPU time of 50 to 60 percent at 4MB/MCPU can be raised to over 80 percent by doubling memory to 8MB/MCPU. The actual amount of memory needed to raise virtual CPU time above 80 percent will vary between sites. No significant capacity increase should be expected from adding memory when virtual CPU time is already above 80 percent.

3. The first .5MB of total system memory is taken up by system overhead and is unavailable for user paging. The total system capacity must be reduced by four users to account for this.

4. There is a requirement for a minimum amount of disk I/O capacity per MCPU. The requirement is a non-linear function of MCPU, and is given in Table 14-5.

   The disk I/O capacity requirement is not related to the storage capacity required by the total number of registered users. It is based on the need to have a certain number of disk I/O operations in progress simultaneously, in order to support the paging load generated by the logged in users.

   A disk I/O operation is in progress not only when data is being transferred, but also when a disk arm is seeking to a specified cylinder, in preparation for a data transfer. Honeywell disk MPCs allow several logical channels to share a single physical channel. A seek can be in progress on each logical channel; only the (relatively short) data transfer operations contend for use of the physical channel.

   The requirement is for a minimum number of high use disk arms, and at least an equal number of logical channels. A high use disk arm is one which accesses a volume containing either directories (the RLV) or process directory (PDIR) segments. Faults on the pages of directories or PDIR segments account for a very large fraction of all Multics page faults.

Table 14-5. Minimum Disk I/O Capacity per MCPU

MCPU	Minimum High Use Disk Arms and Logical Channels
1	7
2	12
3	16
4	20
5	22
6	23
7	24
8	25
9-10	26
11-12	27
13-14	28
19-18	29
19-23	30
24-29	31
30-36	32

The figures for MCPU above 8 are predictions based on extrapolation; they have not been verified experimentally.

5. There must be a minimum of one physical disk channel for each group of four high use disk arms. These channels must be capable of simultaneous data transfers. That is, they must each be in a separate MPC, or paired in dual channel MPCs that are configured so as to be capable of full simultaneous dual channel operation. (Some dual channel MPCs have only dual connectability and are not capable of simultaneous data transfer.) Of course, the MPCs and drives must be configured such that the I/O load of the high use arms is distributed evenly across all of the physical channels.

6. There should be at least as many logical disk channels as the minimum number of disk arms given by rule 4. Additional logical channels are useful, up to the actual number of disk arms in the configuration or the maximum number of logical channels per subsystem, whichever limit is reached first.

7. There should be no more than four to six high use physical disk channels per IOM. Six high use channels will saturate an IOM, causing occasional transfer timing errors, which will be handled by software retry. (Tape channels that are heavily used for high speed transfer place a heavy load on an IOM, and should be included in the per-IOM limit on high use channels.) Additional low use channels, present for redundancy or connectability, may safely be connected to an IOM. (A low use channel is one that is not required by the above rules.)

8. It is possible to trade memory for disk hardware, to some extent. That is, it is possible to compensate for insufficient disk hardware by adding memory. To a lesser extent, the reverse is possible (i.e. adding disk hardware to compensate for insufficient memory). But this is less effective because insufficient memory creates more problems than extra disk hardware can solve. A system should be

able to operate at almost normal efficiency with half the required disk hardware if it has double the required memory. Beyond that point (even less disk and more memory) the trade is expected to be less effective, to an unknown degree.

SAMPLE CONFIGURATIONS

A. Minimum Configuration

```
1 DPS 8/52M CPU (.95 MCPU)
1 SCU with 4MB (1 MW)
1 IOM (32 channels)
1 Disk MPC, single channel
4 Model 451 disk drives (152K pages)
1 DN66 FNP, 32K
  maxe=4, maxu=26
```

B. Small Configuration

```
1 DPS 8/62M CPU (1.3 MCPU)
1 SCU with 8MB (2MW)
1 IOM (32 channels)
2 Disk MPCs, single channel
9 Model 451 disk drives (342K pages)
1 DN66 FNP, 64K
  maxe=9, maxu=66
```

C. Medium Configuration

```
2 DPS 8/70M CPUs with 32K cache (3.8 MCPU)
2 SCUs, each with 16 MB (total of 8MW)
2 IOMs (32 channel)
5 Disk MPCs, single channel
19 Model 451 disk drives (722K pages)
3 DN66 FNPs, 64K
  maxe=19, maxu=224
```

D. Large Configuration

```
4 DPS 8/70M CPUs with 32K cache (7.6 MCPU)
4 SCUs, each with 16MB (total of 16MW)
2 IOMs (32 channel)
7 Disk MPCs, single channel
25 Model 451 disk drives (950K pages)
6 DN66 FNPs, 64K
  maxe=25, maxu=452
```

E. Largest Configuration

```
 6 DPS 8/70M CPUs with 32K cache (11.4 MCPU)
 2 IOMs (32 channel)
 4 SCUs, each with 16MB (total of 16MW)
 7 Disk MPCs, single channel
27 Model 451 disk drives (1026K pages)
 8 DN66 FNPs, 64K
   maxe=27, maxu=680
```

Tuning Parameters

max_eligible

the max_eligible (maxe) tuning parameter should be set, initially, to equal the required number of high use disk arms given by rule 4, above. Then, it should be fine tuned as follows.

Make measurements and adjustments only during periods of very heavy load. Use the total_time_meters command to determine system load and observe the results of adjustments. Use the -reset argument, and observe system behavior over 15 to 30 minute intervals, to smooth out short term fluctuations and ignore the effects of the more distant past.

Zero idle should be at or near zero during the entire period in which the adjustments are being made. It might be necessary to experiment during the high use periods of several days to determine the optimim value of maxe for the site.

During these metering and tuning operations, proceed slowly and cautiously. Make sure that measurements made after tuning adjustments are repeatable, and are not being affected by unrelated short-term changes in system or user behavior. Measurements should be made several times per day for several days, to confirm repeatability

Tune maxe to minimize MP Idle and maximize virtual CPU time. Decrease maxe, if necessary, until MP Idle starts to increase significantly. Then increase maxe slowly (by 1 for each trial) and record the observed change in MP Idle and virtual CPU time. At some point, increasing maxe will produce no further improvement (decrease in MP Idle and increase in virtual CPU time). Then, either leave maxe at that value, or lower it to the point where no further significant improvement was observed.

On a well balanced hardware configuration, it should be possible to reduce MP idle to less than 1%, or even to zero, without introducing any undesirable side effects. Such undesirable effects will manifest themselves as an increase in some overhead and a resulting decrease in virtual CPU time. On a configuration with limited memory, increasing maxe will cause thrashing, but even on a system with a liberal amount of memory, setting maxe too high will increase scheduling overhead without producing any beneficial results.

If maxe is increased significantly beyond the initial value (given by rule 4), examine disk meters (over the same 15 to 30 minute interval as used for total_time_meters) for average page wait times significantly exceeding 50 ms. If necessary, spread the paging load by adding more high use disk arms. Put process directories on more volumes, adding volumes if necessary, and adding logical and physical channels as needed to meet the constraints given by the configuration rules, above.

min_eligible

The min_eligible tuning parameter should be set to 75-80% of max_eligible. This reduces the amount of computation necessary in the getwork function of the traffic controller, while still allowing some amount of control of main memory thrashing by the working set computations.

Initializer Terminals

The initializer process performs various functions, the most obvious of which is controlling the answering service subsystem that handles logins, logouts, process creation, and process destruction. The initializer needs to be able to display its messages on the console designated to receive output from the various streams of its process. The bootload console is not a good choice for this task, mainly because it is slow and pauses after every few lines. This causes the initializer to suspend operation as well, until the queued messages can be printed. Thus, the faster messages can be printed, the faster the initializer can perform its various functions. It is recommended that at least one high-speed hardcopy device be dedicated as a message coordinator terminal (two would be even better). This provides the added benefit of being able to segregate messages of different types on different terminals (e.g., login/logout messages to one terminal and enter_output_request messages to another).

GLOSSARY OF METERING TERMS

activate

To make a segment active. Done by reading the VTOCE of the segment, setting an ASTE, filling in the ASTE, and hashing it into the AST hash table. The parent directory of the segment must be locked in order to activate it.

active

1. of a segment.
   Having a page table (and AST entry) in main memory; the criterion for whether or not a segment is active is whether or not it is hashed into the AST hash table.

2. (loosely) of a page.
   Belonging to an active segment.

APT

Active Process Table. This table is contained in the segment tc_data (traffic control data database) and contains information on all active processes necessary to the traffic controller. The number of entries in the APT is defined at bootload time by the tcd configuration card. This is a wired, unpaged segment.

APTE

Active Process Table Entry. Each active process contains one APTE in the APT of tc_data. An APTE is 64 words containing information about a single process required by the traffic controller.

AST

Active Segment Table. This table is the uppermost part of the segment sst_seg (System Segment Table segment) and contains information on and the page table for every active, paged segment. The number of entries in the SST is defined at bootload time by the sst configuration card. This is a wired, unpaged, hardcore segment.

AST hash table

A table, kept in active_sup_linkage, that holds the heads of hash threads so that the UID of any segment may be used to find its ASTE, if it is active, or the fact that it is not active.

ASTE pool

There are four sizes of AST entries, those containing page tables of 4, 16, 64, and 256 PTWs. Those of each size form four pools, that are managed separately.

AST trickle

A mechanism implemented in the AST replacement algorithm that periodically updates appropriate ASTEs to their corresponding VTOCEs. It is driven by AST traffic.

ASTE

Active Segment Table Entry. Each active, paged segment is defined by an ASTE. Each ASTE has a 12-word header followed by its page table, which may be 4, 16, 64, or 256 words, depending on the size of the segment.

ATB

Average Time Between; used as an acronym in the output of several meters.

block(ed)

A state of a process when it's awaiting some non-system event (wakeup), such as an interaction, to occur. When a process goes into the blocked state, it loses eligibility.

bootload

1. (verb) to initiate the operation of the Multics system when it is down, i.e., to bring it up by issuing the BCE boot command.

2. (noun) the act of bootloading.

3. a term used to refer to a single Multics session. The time from startup (bootload) until shutdown is referred to as one Multics bootload.

bound fault

A fault occurring when an active segment is referenced beyond the bound of its page table, resident in the ASTE. When this occurs, the segment's page table must be moved to a larger ASTE pool. Bounds fault and boundsfault are other terms meaning the same thing.

branch

A data structure in a directory that describes a segment or directory. A segment's branch contains a physical volume ID and VTOC index for the VTOCE of the segment. The ACL, names, author and bit count of a segment may be found in or from its branch.

cache

A semiconductor buffer memory in the processor port logic. An attempt is made to maintain the most recently fetched words from main memory in the cache. The cache provides a substantially faster access time than that of main memory. As each processor contains its own cache, strategies are needed to prevent confusion about main memory contents.

CME

Core Map Entry. Each frame (1024 words) of main memory has a corresponding CME. CMEs are four words in length and define the current status of each frame of main memory.

connected

of a process and a segment. A segment is said to be connected to a process, or vice versa, if the descriptor segment of that process contains an SDW that describes that segment, and is not faulted.

controller adapter

Also referred to as CA. The CA is the physical connection between an MPC and a device. Each MPC can have one or two CAs, and each CA is capable of addressing up to 64 devices.

core map

A page control database that contains a four-word entry (CME) per frame of main memory. It is protected by the page table lock.

core

An obsolete term used in many program listings and comments for main memory.

CPI

Common Peripheral Interface, also referred to as Common Peripheral Channel or CPI channel. This is a low-speed channel interface in the IOM. Currently this type of channel is only used to connect the bootload console to the IOM.

deadline

A scheduling mode whereby processes are made eligible at, or shortly after, their assigned deadlines. The scheduler chooses the process whose deadline is the earliest.

deadlock

or deadly embrace. A situation where a process having a given resource is waiting for some other process to free a second resource, but unfortunately, the process having the second resource is waiting for the first process to free that first resource. Locking strategies in Multics are designed to prevent this situation.

DIA

Direct Interface Adapter. This is the type of channel used to interface the IOM with the Front-End Network Processors (FNPs).

DIM

Device Interface Module. The program that contains the code for managing the physical operation (as opposed to the logical use) of a device.

dispatching

The act of choosing and placing an eligible process in the running state.

DSBR

Descriptor Segment Base Register, sometimes referred to as DBR. A hardware register containing, among other things, the address of the descriptor segment of the process currently executing on the processor. Each Multics processor has one DSBR.

DSEG

Descriptor Segment. An array of hardware control words called Segment Descriptor Words (SDWs) that specifies the mapping between segment numbers and either segments or taking a segment fault. Every process has its own descriptor segment (segment number 0). When a process is executing on a processor, the DSBR of that processor points to the user's DSEG. This is a paged segment, and the first page of a process' DSEG must be wired in main memory before the process can execute on a processor.

EDAC

Error Detection and Correction. Many of the hardware components contain logic to detect and correct errors, such as main memory and disk subsystems.

EHS

Entry Hold Switch. A switch in the ASTE of a segment telling the supervisor that this segment is not to be deactivated.

eligible

A process is made eligible by the traffic controller at the time that the latter decides that the process should be allowed to consume main memory resources (i.e., take page faults). Only eligible processes can run, although they must be loaded first. All processes are either eligible or ineligible.

entry

a branch or link.

frame

or "main memory frame". A 1024-word block (beginning on a 1024-word boundary) of main memory.

fsmap

The bit map of the volume map of a physical volume.

get_processor

The traffic controller routine that assigns an eligible process to run on a processor. This routine always selects the highest-priority ready process in the eligible queue to run. If there are no ready processes in the eligible queue, get_processor assigns the idle process for that CPU to run.

hardcore process

A process fabricated by the system to perform some periodic overhead function. Currently, there are only two hardcore processes defined, Syserr_Logger.SysDaemon and MCS_Timer_Daemon.SysDaemon. The Syserr_Logger.SysDaemon process copies syserr messages from the wired syserr buffer in syserr_data into the syserr LOG partition. The MCS_Timer_Daemon.SysDaemon controls MCS timer mechanisms. A hardcore process is also called a supervisor process.

idle process

A process associated with a particular processor. The idle process is run when no other process can be found to run on that processor. The idle process cannot withstand most faults. If a forbidden fault occurs, the system crashes with the message "FAULT IN IDLE PROCESS." This is symptomatic of a hardware problem or coding error in an interrupt handling routine.

idle time

Processor time spent running the idle process. Idle time can be categorized into four types, based on the reason for their occurrence: zero idle, there was absolutely no work to be done on the system at that time; NMP or non-multiprogramming idle, there was some work to be done, but not enough for all processors; MP or multiprogramming idle, there was work to be done but the system chose not to perform it due to main memory restraints; and loading idle, the time spent waiting for a process to become loaded so that it could run on a processor.

interactive queue

The scheduling queue where a process is put after an interaction, assuming the tuning parameter int_q_enabled is on.

ITT

Interprocess Transmission Table. This table resides after the APT in tc_data. Its size is defined by the tcd configuration card. It is used, as its name suggests, to transmit wakeups between processes.

KST

Known Segment Table. A per-process table describing the mapping between segment numbers in that process and storage system segments. The segments are identified via pointers to their branches (using other segment numbers in that process) and unique IDs. The KST also contains a list of private logical volumes attached to the process. The KST is a reverse-deciduous segment.

link adapter

Also referred to as LA. The link adapter is the physical connection in an MPC connecting it to an IOM. Each MPC can have one or two LAs (i.e., one or two physical channels). There are options available to have non-simultaneous switched channels to an MPC, which then allow up to four LAs, but only two can be active at once. Physical data transfers can only occur over an LA, one at a time per LA.

loaded

A process is said to be loaded when its two critical process pages have been wired. Processes are loaded by the traffic controller when they are made eligible. Only loaded processes can run on a processor. The two pages necessary are page 0 of the process' DSEG and PDS.

lock

A datum used to serialize processes performing certain actions and using or modifying certain databases. A process locks a lock before performing these actions or using these databases, and unlocks it when done. Only one process may have a lock locked at one time. A process trying to lock a lock that is locked by (or to) another process must wait for that lock. Processes are said to hold locks when they have them locked. In Multics, a lock is typically a single word of storage that is zero when unlocked and contains either the process ID or lock ID of the process that has it locked when it is locked.

locking hierarchy

A conceptual partial-ordering of a set of locks via the arbitrary relation "higher" (>). If lock A > lock B, and lock B > lock C, then lock A > lock C. There is no inverse, and two locks may be totally unrelated. The locking hierarchy is used to prevent deadlock. The rule used by the Multics supervisor states that no process may wait for the unlocking of a lock unless that lock is higher than every lock it has locked.

logical channel

A secondary channel associated with a PSIA channel in an IOM. Control words can be placed concurrently in the IOM for all logical channels associated with a physical PSIA channel. These control words are accessible by the MPC connected to the physical PSIA channel, which allows simultaneous activities other than physical data transfers to take place. For example, if a PSIA channel has four logical IOM channels associated with it, and this PSIA interfaces to an LA of a disk MPC (MSP), and there are four or more seek arms connected to that MSP via the CAs of the MSP, then up to four simultaneous seek operations can be occurring, although only one physical data transfer can be occurring.

main memory

(formerly core) the core or MOS memory device from which the processor normally fetches instructions and data. All pages must be in main memory to be directly used by the processor.

max_eligible

or maxe. A tuning parameter defining the maximum allowable number of eligible processes.

metering cell

Also referred to as cell. These are locations in system tables where various system software modules store metering data.

migrate

To move a page from main memory to secondary storage.

min_eligible

or mine. A tuning parameter defining a minimum number of eligible processes. The traffic controller will allow this number of eligible processes regardless of the sum of their estimated working sets.

MPC

Micro-Programmed Controller. A hardware component that controls peripheral devices via control programs that are loaded into it. The control programs are known as the MPC's firmware.

MSP

Mass Storage Processor. This term is often used to refer to the MPC controlling mass storage devices (i.e., disk devices).

MTP

Magnetic Tape Processor. This term is often used to refer to the MPC controlling magnetic tape units.

multiprocessing

Pertaining to the **simultaneous** execution of two or more processes by a multiprocessor system.

multiprogramming

Pertaining to the **concurrent** execution of two or more processes by interleaving their execution.

notify time-out

A traffic controller event that occurs when an expected system event does not happen within a specified amount of time. The process which was waiting is made ready, as though the event had occurred. Usually, notify time-outs are indicated by a message displayed on the bootload console.

notify

The occurrence of a system event, such as the arrival of a page in main memory. When a process goes into the waiting state, it is awaiting a system notify.

page fault

An exception condition detected by the processor hardware (the appending unit) when an attempt is made to use a PTW that specifies that the page of the segment is not in main memory. This is indicated by the bit ptw.df being off. This causes the unconditional execution of a specific fault vector entry that effects transfer to the page fault handler.

page table lock

A global lock in the header of the SST that controls access to the page control databases. Only one processor is allowed to modify these databases at any time to prevent inconsistencies from occurring.

page table

The array of PTWs that specifies the mapping between addresses in a segment and either main memory frames or page faults. The page table of an segment is part of the ASTE; only active segments have page tables. The SDW of a paged segment contains the absolute address of its page table.

page

A 1024-word extent of data at a 1024-word boundary of some segment. Pages belong to segments; they can exist in main memory frames or on disk records (or PD records).

PDIR

Process Directory. A temporary directory created for a process at process creation time. It contains temporary, per-process segments such as the DSEG, PDS, KST, and process stacks. Often the term is used to describe not the directory itself, but rather all the segments that are found in the process directory (e.g., "process directories are put on the logical volumes specified by set_pdir_volumes").

PDS

Process Data Segment. A per-process (reverse-deciduous) hardcore segment that contains all per-process information needed by a process other than that in the SST. The first page of the PDS of a loaded process is wired.

physical channel

This term refers to a physical connection between an IOM and a device or MPC. Data transfers can only occur over physical channels. CPI and DIA physical channels have no associated logical channels. Only PSIA physical channels may have multiple associated logical channels.

post_purge

A tuning parameter that, when enabled, tells the scheduler to place the pages of a process losing eligibility at the head of the list for the page removal algorithm. This is done with the assumption that the process is most likely to not require these pages for the longest period of time, and thus allows other pages to remain in main memory.

PRDS

Processor Data Segment. A segment containing information about a single processor. This is a paged segment, and acts as a wired, ring-0 stack for operations that cannot take page faults or lose the processor.

pre-empt

Also known as preempt, this refers to the removing of a process from a processor for any reason without that process' consent. For example, when a process has used its allocated time quantum and is preempted, or an event for which a higher-priority process was awaiting occurs and that process preempts the currently-executing process.

process loading

The bringing into main memory and wiring of the first page of a process' DSEG and the first page of its PDS in order that it may be scheduled to run on a processor.

process

An address space and an execution point within that address space. All users of Multics have a process created for them when they log in to the system. As that process touches segments, its address space is expanded accordingly.

processor time

The amount of processor-seconds the system has accumulated since it was bootloaded. This time is the same as real time for a system that has only one processor, but is guaranteed to be different if the system ever had more than one processor configured.

PSIA

PSI Adapter, or PSIA channel. This type of interface channel connects the IOM to the various MPCs of the system. The PSIA channel has as associated patch plug that controls the number of logical channel slots that particular connection inhabits in the IOM. A PSIA patch plug can define up to eight associated logical channels.

PTW

Page Table Word. A processor hardware control word, an element of a page table, that specifies either a main memory frame address or that the processor should take a page fault when attempting to use this PTW.

pxss

the traffic controller. The acronym is historical and stands for process exchange and switch stack. The traffic controller source is a program named pxss.alm.

quantum

An amount of real or virtual processor time, also referred to as time slice. This term is usually used in conjunction with the traffic controller to mean the amount of time a process will be allowed to execute on a processor before being preempted and rescheduled.

ready

A state of a process when it's ready to do some work. A process becomes ready when whatever event occurs that it was awaiting, be it a system event or a non-system wakeup. Processes must be in the ready state before they can be assigned to run on a processor.

real time

The actual time period as measured by a normal clock, often referred to as "wall clock time".

realtime process

A process operating in a realtime work class. Such processes are required to have good response times, but are unlikely to require long quanta, for example, transaction processing applications and I/O daemons.

record

A 1024-word, contiguous extent of disk that can hold a copy of a page.

response time

A term used to refer to the real time between an interaction and the beginning of the output from that interaction. However, the traffic_control_meters command calls response time the average amount of real time between a process' interaction and the scheduling (making eligible) of that process.

running

A state of a process when it is actually executing on a processor.

scheduler

A term used to refer to the scheduling routines of the traffic controller.

scheduling

The act of choosing and promoting an ineligible process to eligible status.

SDW

Segment Descriptor Word. A hardware control word, an element of the descriptor segment, that gives the absolute address of an unpaged segment, or the absolute address of the page table of a paged segment. The SDW also contains access mode and ring brackets, as well as other information. The SDW can also indicate "segment missing," in which case a reference will cause a segment fault.

seek distance

The number of cylinders of a disk that the seek arm must move in order to place itself at the correct cylinder for the next request.

segment fault

An exception condition detected by the processor (the appending unit) when an attempt is made to use an SDW that describes a segment not yet connected to the process in whose descriptor segment the SDW appears. This is indicated by the bit sdw.df being off. A segment fault causes a specific fault vector entry to be unconditionally executed, ultimately invoking the segment fault handler.

setfault

An operation performed by the procedure of the same name at the time a segment is deactivated or its access attributes are changed. This operation modifies or faults all of the SDWs for a given segment, located via the trailer list. The associative memories of all processors are always cleared as the last step of a setfault.

SST

System Segment Table. A per-system database containing metering information, the active process table, and other information. This is a wired, unpaged segment whose size is dependent on the amount of main memory present on the system and the size of the APT.

SSTNT

System Segment Table Name Table. A segment containing the names of currently active segments. The names are kept current only if specified by the parm astk configuration card.

STR

System Trailer segment. This segment contains trailers, which are entries attesting to the fact that a process has an SDW for a given active segment. Each active segment possesses a trailer list of such processes. The trailer identifies the process via the AST offset of its descriptor segment's ASTE, and contains the segment number of the segment in that process. This segment is used for setfaults.

subsystem

A logical grouping of peripheral devices, such as disk and tape drives. Multics allows up to 64 devices per subsystem. Each subsystem is handled relatively independently by the system software. IOM logical channels are unique for a subsystem; i.e., each IOM logical channel will address the devices of only one particular subsystem. Any subsystem can have up to eight logical channels assigned to it.

tc

An acronym for traffic controller.

tc_data

A wired, unpaged segment containing metering information, the APT, work class tables, and the ITT. This segment contains the information pertinent to the traffic controller. Its size is determined at bootload time from the tcd configuration card.

te

The value of the amount of processor time used by a process in its current eligibility quantum.

tefirst

Time Eligible First. The amount of processor time given a process after an interaction.

telast

Time Eligible Last. The amount of processor time given a process in all subsequent executions.

thrashing

Page thrashing is inefficient use of the available main memory whereby a process causes some pages to be moved into main memory only to have them migrated out of main memory before it's through with them. The process must do extra work to bring those pages back into memory. A system that is thrashing may spend almost all of its time processing page faults, each of which only serves to evict a page needed by another process that will immediately cause another page fault when it tries to use that page. Thrashing is caused by having too much concurrent demand for the available frames of main memory. It can be cured by adding main memory. Alternatively, the concurrent demand can be reduced by reducing the number of eligible processes or changing the work done by those processes to require fewer page faults.

Segment thrashing or AST thrashing is inefficient use of the available AST entries. It is very similar to page thrashing, but is caused by too much concurrent demand for the available AST entries. It can be cured by increasing the size of the active segment table (AST), or by changing the work done by processes to require fewer segment faults.

threaded list

A list where each element of the list contains the address or offset of the next element of the list. A doubly-threaded or double-linked list means that each element of the list contains the address of the next and the previous elements.

threaded

This terms refers to an entity being an element of a threaded list.

throughput

The measure of the amount of work performed during a certain period of time. The units associated with this measure vary depending upon that being examined (e.g., interactions per hour).

ti

Time since Interaction. The amount of processor time a process has used since it interacted.

timax

The maximum value that the ti of a process can assume. A process exceeding this value is given tefirst as its next te quantum.

time slice

The same as quantum. An amount of processor time given a process to perform a request.

total CPU time

The total amount of processor time consumed while a process was executing on a processor. This includes the processor time spent performing system overhead functions such as page fault processing, notify processing, and interrupt processing.

traffic control

This term refers to the management of processes in general, especially as it applies to the system scheduler/dispatcher routines.

TRO

An acronym for timer runout. A timer runout fault occurs when a process has exhausted the amount of processor time allocated for that particular scheduling.

ts

Time since Scheduling. The amount of processor time a process has used since its scheduling priority has changed.

tssc

Time Since State Change. The amount of processor time a process has used since its process state has changed, i.e., since its APTE was last touched by the traffic controller.

URC

Unit Record Controller. This term is often used to refer to the MPC controlling unit record devices such as printers, card readers, and card punches.

URP

Unit Record Processor. This term is often used to refer to the MPC controlling unit record devices such as printers, card readers, and card punches.

virtual CPU time

The actual processor time spent in a process or in the system, minus all processor time spent in page fault, segment fault, bound fault, process switching, and interrupt processing.

VTOC

Volume Table Of Contents. The part of a storage system volume (disk) containing information as to the contents of that volume. The VTOC occupies a fixed, contiguous extent at the beginning of a physical volume.

VTOCE

Volume Table Of Contents Entry. A 192-word entry in the VTOC describing one segment resident on that volume. This description includes record addresses and other attributes, one of which is the date-time the segment was created.

wait event

The event for which some process is waiting, such as the arrival of a page in main memory or the waiting for a lock to be unlocked.

wait

A state of a process when it's awaiting the occurrence of some system event, such as the arrival of a page in main memory. A process does not lose eligibility when it goes into the waiting state. A wait is ended by a notify signalling the occurrence of the event.

wakeup

A non-system event that causes a blocked process to become ready.

wired

1. of a page.
   A page that may not be removed from main memory. The bit ptw.wired tells page control not to evict this page.

2. of a segment.
   A segment having some or all of its pages wired.

work class

A scheduler term used to group particular processes into different classes, and to provide different scheduling priorities to classes of processes. The system can accomodate up to 16 separate work classes.

working set

An estimate of the use a process requires of the main memory resources. The size of a process' working set is predictive of future main memory requirements.

working_set_addend

or wsa. A tuning parameter used in computing the estimated working sets of a process, which in turn is used to control main memory thrashing.

working_set_factor

or wsf. A tuning parameter used in computing the estimated working sets of a process, which in turn is used to control main memory thrashing.

SECTION 15

BULK INPUT/OUTPUT

The bulk input/output facility is normally used to manage all card reading, card punching, and printing requests on both local and remote unit record equipment. Printing facilities include a set of prioritized queues for requests submitted by users, the management of one or more printers, the handling of special forms, and numerous commands to control the operation of this facility. An optional operational mode allows the spooling of print requests onto tape for subsequent printing on either the same or another system. The card input facilities include both the input of data and the input and submission of absentee jobs. This facility is integrated with the Multics access control mechanism and the access isolation mechanism (AIM) so that integrity of users' data is maintained. Accounting is provided for bulk input/output.

The software that handles printing, punching, and card input is called the I/O daemon. The I/O daemon is organized into a coordinator process and a number of driver processes; a driver is associated with each local or remote device. The I/O daemon normally runs with highly privileged access on the SysDaemon project, though some of the drivers can run with fewer privileges if the site desires.

The I/O daemon is normally run with message coordinator terminals. The particular terminal or terminals chosen depend on the needs of the site. For remote devices, partial control of the process is from the device itself, using its card reader or input keyboard if available.

You set up the environment in which this facility runs by creating and modifying the I/O daemon databases, creating info segments or other information to inform the user community of what is available, and setting up special operator exec_coms or instructions. The operator runs the facilities according to instructions given by the system administrator, taking care of the needs of the peripheral devices and following special requests made by the system.

I/O DAEMON DIRECTORIES

The Multics I/O daemon software depends on the existence of certain directories and segments. The most important of these directories and segments are created and initialized at a new Multics site by the acct_start_up.ec segment (described in the *Multics System Administration Procedures* manual, Order No. AK50).

In order to set up the I/O daemon to meet the needs of your site, and to manage and supervise the various I/O daemon processes, you must be familiar with that portion of the hierarchy around which the daemon processes are organized. The main node of this hierarchy is the directory named >daemon_dir_dir (with a short name of >ddd). This directory contains segments and directories used to support the various system daemon processes.

Contents of daemon_dir_dir Directory

The >daemon_dir_dir directory contains the following directories of interest:

io_daemon_dir
: holds all I/O daemon databases

cards
: a storage pool for card deck image segments read by the system card input process (local or remote station)

io_msg_dir
: contains mailboxes for each device (station) for which driver to driver messages will be sent or received

These directories and their contents are described in the following paragraphs. The access isolation mechanism (AIM) access class for all these directories is system_low.

CONTENTS OF io_daemon_dir DIRECTORY

The >daemon_dir_dir>io_daemon_dir directory contains a set of administrative databases and working storage used to direct the activities of the I/O coordinator process and the various device drivers.

The main database, iod_tables, is set up by you. Most of the other segments and directories are created and maintained by the I/O coordinator, acting on information contained in the iod_tables segment.

The following segments are contained in io_daemon_dir:

coord_comm.ms
: ring 1 message segment in which driver processes place messages for the I/O coordinator

coord_lock
: a segment used by the process overseer of IO.SysDaemon to prevent initialization of a driver process before an I/O coordinator has been created; also prevents creation of more than one I/O coordinator

iod_tables
: master control tables for the I/O coordinator; may be compiled by using the iod_tables_compiler command

iod_tables.iodt
: source segment for iod_tables; may be updated and compiled to yield iod_tables

iod_working_tables
: working copy of iod_tables used by the device drivers and users; copied from iod_tables during I/O coordinator initialization

iodc_data
: a segment containing the process identifier of the I/O coordinator and the event channel identifier to be used by a new driver for initial communication with the coordinator

printer_notice
: an optional segment containing information that the site administrator wants to be printed on the page following the head sheet of every printer listing, local or remote. The segment must contain ASCII text. It should be no longer than 60 lines and the line length should correspond to the shortest printer device in use. This feature is useful in notifying users of printing charge rates, available request types, when they are processed, stock forms used for each, and other useful information covering the printing operations on the system.

XXX_N.ms
: ring 1 message segment for I/O daemon queues. One such message segment is created for each priority queue of a request type. (You normally create these segments with the create_daemon_queues command.) XXX is the request type name and N is the queue number (1<=N<=4).

There are several directories contained in the io_daemon_dir, some of which are site dependent. Access class information about these directories is included here as an aid to those sites using the access isolation mechanism (AIM).

coord_dir
: working storage for the I/O coordinator, which is created and managed by the I/O coordinator; the access class is the authorization of the I/O coordinator

rqt_info_segs
: created by you to hold any request type info (rqti) segments used by drivers (see the cv_prt_rqti command in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64, and "Request Type Info Segments" later in this section); the access class is system_low

meter_dir
: created by you to hold driver metering data (for future use)

<major device>
: separate directory for each major device currently being run by a device driver. These directories are managed by the I/O coordinator and their names are site dependent. Each major device directory contains a driver status segment for each minor device associated with the major device. The access class is the authorization of the device driver.

CONTENTS OF cards DIRECTORY

The storage pool for card deck image segments consists of a subtree of the directory hierarchy, which is headed by >daemon_dir_dir>cards. One access class directory for each access class (as needed) is contained in the cards directory. Storage is always allocated within the access class directory that corresponds to the access class specified on the ++AIM card. Person directories are contained in the appropriate access class directory. A person directory is created for each person who needs temporary storage. A person directory contains all segments and multisegment files for a given person at a given access class. For example, if a user with Person_id TSmith is at system_low, the following directory is allocated for his card deck image segments:

```
>daemon_dir_dir>cards>system_low>TSmith
```

CONTENTS OF io_msg_dir DIRECTORY

The >daemon_dir_dir>io_msg_dir directory contains mailboxes of the form <device>.mbx for each device and remote station that uses the driver to driver message facility. See "Driver to Driver Message Facility" later in this section.

I/O DAEMON TABLES

In order to manage the use and operation of the I/O daemon, an administrative database exists that can be adapted to the specific needs of a particular Multics site. This database contains several different tables of information and hence is referred to as the "I/O daemon tables." The database is generated from a source language description prepared by you. The iod_tables_compiler command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) is used to translate the source description into the encoded representation of the I/O daemon tables. The encoded representation, which is used by the I/O coordinator, must be named "iod_tables".

For information on setting up HASP RJE station printers, readers, and punches, refer to the *Multics HASP Service and Utility* manual, Order No. GB60.

I/O Daemon Tables Source Language

The purpose of the I/O daemon tables source language is to define the devices and the request types to be used by the I/O daemon. A source file consists of a sequence of statements and substatements that define and describe each device, request type, and shared communications channel. In addition, certain global information items are defined that do not pertain to any particular device, request type, or shared communications channel.

SYNTAX

The syntax of the source language statements and substatements is of the form:

```
<keyword>:  <parameter>;
```

The only exception to this is the "End" statement. The keyword of a statement begins with a capital letter; the keyword of a substatement is entirely in lowercase letters. Substatements describe attributes of devices, communication lines, or request types for the given statement. Each group of one statement and its substatements constitutes a statement description.

PL/I style comments beginning with "/*" and ending with "*/" may appear anywhere within the source file. Similarly, blanks, tabs, and newlines not embedded within a keyword or parameter are ignored. However, in order to include blanks, tabs, newlines, colons, or semicolons in a parameter, you must enclose them in quotes. If a parameter begins with a quote, all immediately following characters up until the next quote are taken as the parameter. It is possible to embed quotes within a quoted string using the double quoting escape convention of PL/I.

STATEMENTS

The following statements may appear anywhere within the source file.

Time: <number>;
defines the number of minutes that the coordinator saves a processed request. When segment deletion is requested, it is delayed this amount of time. If some problem is discovered that necessitates the reprocessing of requests, those requests performed less than <number> minutes ago can be restarted. One, and only one, Time statement must appear in the file.

Max_queues: <number>;
defines the default number of priority queues for each request type. The maximum value of <number> is 4. (Queue 1 is the highest priority and queue 4 is the lowest priority.) One, and only one, Max_queues statement must appear in the file.

Line: <name>;
defines the name of a logical line_id and denotes the beginning of a shared communications line description. Any subsequent substatements (see below) apply to this line until the next Line, Device, or Request_type statement is encountered. Any <name> may be chosen; it can be a maximum of 32 characters and cannot contain spaces or periods. There may be up to 360 Line statements. This statement is optional.

Device: <name>;
defines the name of a major device and denotes the beginning of a device description. Any subsequent substatements (see below) apply to this device until the next Line, Device, or Request_type statement is encountered. Any <name> may be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces. At least one Device statement must appear in the file.

Request_type: <name>;

defines the name of a request type and denotes the beginning of a request type description. Any subsequent substatements (see below) apply to this request type until the next Line, Request_type, or Device statement is encountered. Any <name> (not containing periods or spaces) may be chosen; it can be a maximum of 24 characters. At least one Request_type statement must appear in the file.

End;

marks the end of the source language description. Unlike all other statements, it has no parameter. Any text occurring beyond the End statement is ignored. One, and only one, End statement must appear in the file.

SUBSTATEMENTS FOR LINES

The following substatements describe various attributes of a shared communications line and may appear in any order following a Line statement.

channel: <name>;

defines the name of the communications channel to be attached when using the logical line_id (defined in the Line statement). It is normally a communications channel identifier for an RJE station. The <name> may be up to 32 characters and cannot contain any spaces. One, and only one, channel substatement must be given for each Line statement.

att_desc: <string>;

defines the attach description to be passed to the remote_teleprinter_, remote_printer_, and remote_input_ I/O modules. The <string> may be up to 256 characters and should appear in quotes since there will be imbedded spaces. If the control variable ^a appears in <string> it will be replaced by the channel <name> (described above). One, and only one, att_desc substatement must be given for each Line statement.

device: <name>;

defines a major device that can use this logical line_id. At least one device substatement must be given for each Line statement. Any major device specified must also have the line: variable; substatement under the Device statement.

SUBSTATEMENTS FOR DEVICES

The following substatements describe various attributes of a device and may appear in any order following a Device statement.

driver_module: <name>;

defines the name of a procedure to be executed by a driver process when running the associated device. The <name> can be a full pathname or simply an entryname. In the latter case, the search rules are used to locate the procedure. Several standard driver modules are provided by the system (see "Standard Driver Modules" below). One, and only one, driver_module substatement must be given for each Device statement.

default_type: <name>;

defines the default request type for the associated device. The <name> must appear as the parameter in a Request_type statement. Unless overridden by the operator when a driver is initialized, the driver processes requests of this default type. A default_type substatement must not be given for a major device if the driver has minor devices; i.e., it must not be given for a Device statement if it is given for a minor_device substatement.

args: <string>;

defines an argument string to be interpreted by the driver module for the associated device. The <string> may have any arbitrary format up to a maximum of 256 characters. In practice, the composition of the <string> depends on the particular driver module that interprets it. Each driver module has its own conventions for the <string> format (see "Standard Driver Modules" below).

The following three substatements describe alternate methods by which a driver may attach the associated device. These substatements are mutually exclusive. One, and only one, of these substatements must be given for each device statement.

prph: <name>;

names an input/output multiplexer (IOM) peripheral channel through which the associated device can be attached. The <name> must appear on a prph card in the configuration deck.

line: <name>;

names a dedicated communications line channel through which the associated device can be attached. If <name> is "variable" the channel can be any logical line_id defined by a Line statement. The driver process must have the dialok attribute in the project definition table (PDT) and the communications channel must be defined as slave in the channel definition table (CDT). See the *MAM Communications* manual, Order No. CC75, for more information about the PDT and the CDT.

dial_id: <name>;

defines the dial identifier to be used if the associated device is to be dialed to the driver process over a communications line. The driver process must have the dialok attribute in the PDT and the communications channel must be defined as a login channel in the CDT.

The following three substatements describe alternate methods by which the driver may attach a control terminal. These statements are mutually exclusive. If none is specified, the driver assumes that no control terminal is desired.

ctl_line: <name>;

names a dedicated communications line channel through which the control terminal can be attached. The driver process must have the dialok attribute in the project definition table (PDT) and the communications channel must be defined as slave in the channel definition table (CDT). See the *MAM Communications* manual, Order No. CC75, for more information about the PDT and the CDT.

ctl_dial_id: <name>;

defines the dial identifier to be used if the control terminal is to be dialed to the driver process over a communications line. The driver process must have the dialok attribute in the PDT and the communications channel must be defined as a login channel in the CDT.

ctl_source: <name>;

defines a message coordinator source name to be associated with the driver. A single control terminal accepted by the message coordinator can be used to control many different drivers.

SUBSTATEMENTS FOR REQUEST TYPES

The following substatements describe various attributes of a request type and may appear in any order following a Request_type statement.

accounting: <name>;

defines the name of an accounting procedure to be executed by a driver when processing requests of the associated type. The <name> can be a full pathname or simply an entryname. In the latter case, the search rules are used to locate the procedure. Also, the special <name> "system" can be used to indicate the standard system accounting procedure. If this substatement is omitted, "system" accounting is assumed.

The special <name> "nothing" indicates that no accounting is to be performed. You can use this for running I/O daemons in proceses which don't have access to charge users. When the special <name> "nothing" is used, any tail sheet or its equivalent will indicate that there was no charge for the request.

card_charge: "p1, p2, p3, p4";

defines the resource price names for the card_charge of each queue of the request type. This substatement is optional. If it is not specified, the prices from system_info_$io_prices are used. p1 through p4 are resource price names, which are defined using the ed_installation_parms command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64). The prices must be defined before the iod_tables segment is compiled, or the compilation will fail. The price names for each queue must be given in order, from queue 1 to the maximum number of queues for the Request type description. Each price is defined in units of dollars per 1000 cards. Note: card_charge must not be specified if line_charge is specified.

default_queue: <number>;

The default_queue substatement is used to define the default queue for a request type. The value of <number> may be from 1 to max_queues. If not specified, it is set to the value defined in the max_queues substatement, but it will not be greater than 3.

device: <name>;

specifies a device that can be used to process requests of the associated type. The <name> must appear as a parameter in a Device statement. More than one device substatement may be specified for a request type.

driver_userid: <access_name>;

defines the required person and project names for a driver of the associated request type. If omitted, the <access_name> defaults to IO.SysDaemon, which is the standard system driver. Other access names may be used, for example, to provide a project with its own private driver. The <access_name> you specify must have enough access to run the accounting procedure you've specified under the "accounting" substatement described earlier. For the standard procedure "system," read access is required to the system administration table (SAT) and read/write access is required to the PDTs.

generic_type: <name>;

defines the generic type of the associated request type. If the generic type name matches the request type name, then the request type is the default for the generic type. One, and only one, generic_type substatement must be given for each Request_type statement. If the generic type is neither "printer" nor "punch", the list_daemon_requests command must be used with the -brief control argument, and the cancel_daemon_requests command cannot be used. (Both of these commands are described in the *Multics Commands and Active Functions* manual, Order No. AG92.)

line_charge: "p1, p2, p3, p4";

The line_charge substatement defines the resource price names for the line charge of each queue of the request type. This substatement is optional. If not specified, the prices from system_info_$io_prices will be used. If specified, each price name must be defined in the system price table, or the compilation of the the iod_tables will fail. The price names for each queue must be given in order, from queue 1 to the maximum number of queues for the Request_type description. Each price is defined in units of dollars per 1000 lines.

max_queues: <number>;

The max_queues substatement may be used to define the maximum number of queues for a request type, when it is different from the global Max_queues value. This substatement is optional. The value of <number> may be from 1 to 4.

page_charge: "p1, p2, p3, p4";

The page_charge substatement defines the resource price names for the page charge of each queue of the request type. This substatement is optional. If it is not specified, the prices from system_info_$io_prices are used. If specified, each price name must be defined in the system price table, or the compilation of the iod_tables fails. The price names for each queue must be given in order, from queue 1 to the maximum number of queues for the Request type description. Each price is defined in units of dollars per 1000 pages. Note: page_charge must not be specified if generic type is "punch".

rqti_seg: <name>;

The rqti_seg substatement is used to define the name of the request type info (rqti) segment to be used with the Request_type statement. This substatement is optional. When specified, <name> must correspond to a segment entryname in the >ddd>idd>rqt_info_segs directory, or the driver will fail initialization. When not specified, no driver will look for an rqti segment for this Request_type statement.

I/O DAEMON TABLES SOURCE FILE EXAMPLE

The subset of the source language described so far is sufficient to prepare a complete I/O daemon tables source file. Many sites will find they have no need of any other features. Thus, before describing some of the less commonly used statements and substatements, an example of a source file containing just the ones described above is presented.

```
/* Example of an I/O daemon tables source file */

/* Global parameters */

Time:       60;   /* save requests for 60 minutes */
Max_queues: 3;    /* 3 priority queues per request type */

/* Devices */

Device:             printer_1;        /* onsite printer */
  driver_module:    printer_driver_;
  prph:             prta;
  default_type:     printer;

Device:             punch_1;         /* onsite punch */
  driver_module:    punch_driver_;
  prph:             puna;
  default_type:     punch;

/* Request types */

Request_type:       printer;    /* onsite printer requests */
  generic_type:     printer;
  device:           printer_1;

Request_type:       punch;      /* all punch requests */
  generic_type:     punch;
  max_queues:       1;
  default_queue:    1;
  line_charge       punch_price;
  device:           punch_1;

End;
```

In this sample source file there are two devices and two request types. The request types are handled by the standard system driver, IO.SysDaemon, as implied by the absence of any driver_userid substatement. These two request types, printer and punch, are the default types for the enter_output_request and dpunch commands respectively. The majority of users concern themselves only with these two request types. Output is produced onsite by the printer_1 and punch_1 devices.

MAJOR AND MINOR DEVICES

Special provisions have been made to handle "combination" devices that contain more than one logical device (e.g., a printer and a punch) in a single physical unit. The combination device as a whole is referred to as a "major device"; the multiple subdevices, such as a printer and a punch, are referred to as "minor devices". A major device is connected to Multics via a single communications channel; it can be attached by one process only. Therefore, it is not possible to have separate driver processes running the separate logical devices. To overcome this problem, the driver software has been designed to simulate multiple drivers within a single process. This means that from the coordinator's point of view, each logical device is distinct and run by an independent driver process. Consequently, each one of these logical devices can be fed requests of a different request type and generic type.

Major devices are defined by the Device statement described earlier. Similarly, minor devices are defined by a minor_device substatement. The minor_device substatement is treated as a substatement for devices and, as such, can be freely intermixed with other substatements for devices.

minor_device: <name>;
: defines the name of a minor device belonging to the associated major device and denotes the beginning of a minor device description. Any subsequent substatements (see below) apply to this minor device until the next minor_device substatement or Line, Device, or Request_type statement is encountered. Any <name> may be chosen up to a maximum of 24 characters.

If no minor devices are explicitly defined for a major device, then a default minor device is defined by implication. The primary purpose of a default minor device is to allow certain minor device substatements to be specified for a major device when it has no explicit minor devices. One such substatement is the default_type substatement that was previously described under "Substatements for Devices." In fact, the default_type substatement is actually a substatement for a minor device. When no minor devices are explicitly defined, the default_type substatement applies to the default minor device of the preceding major device. The same is true of all substatements for minor devices.

SUBSTATEMENTS FOR MINOR DEVICES

The substatements below describe attributes of a minor device and may appear in any order following a minor_device substatement or following a Device statement if no minor devices are specified.

minor_args: <string>;
: defines an argument string to be interpreted by the driver module for the associated major device. The string may have any arbitrary format up to a maximum of 256 characters. Conventions for the <string> format expected by standard system driver modules are described under "Standard Driver Modules" later in this section.

default_type: <name>;
: defines the default request type for the associated minor device. The <name> must appear as a parameter in a Request_type statement.

The device substatement described earlier is a substatement for a request type and is used to name devices that can process requests of a given type. Usually, the parameter of a device substatement is a major device name. However, if minor devices are defined for the major device, then the device substatement parameter must include both the major and minor device names separated by a period (e.g., xyz.printer).

SOURCE FILE EXAMPLE USING MINOR DEVICES

This example shows a portion of a source file that illustrates the use of minor devices.

```
Device:             xyz; /* a combination device */
  driver_module:    dummy_driver_;
  args:             "dim= xyz";
  line:             a.h100;
minor_device:       printer;
  minor_args:       "dev= printer";
  default_type:     xyz_prt;
minor_device:       punch;
  minor_args:       "dev= punch";
  default_type:     xyz_pun;

Request_type:       xyz_prt;
  generic_type:     printer;
  device:           xyz.printer;

Request_type:       xyz_pun;
  generic_type:     punch;
  device:           xyz.punch;
```

AIM FEATURES

The I/O daemon incorporates certain features in support of the access isolation mechanism (AIM). If your site does not use access classes above system_low, you do not need to read the following and should skip to "Standard Driver Modules" below.

Every request processed by the I/O daemon has an access class. The access class of a request is equal to the authorization of the process that submitted the request. Each piece of output normally has an access class banner. For print requests, the access class banner appears on the head sheet of each printout. For punch requests on the local punch, the access class banner appears in the flip cards at the beginning of each deck. At remote sites, no access class banner appears. However, if the access class of a request is system_low and the access class name for system_low is null, then the access class banner is omitted.

In the interest of security, some sites may find it desirable to have requests of the same type automatically separated according to access class. To illustrate how this access class separation might be used, imagine a site at which two different access classes are defined. One of these, called "public," is available to all users. The other, called "confidential," is available to only a limited number of users who deal with sensitive information. Further, suppose that the site has two printers, both of which are used to process requests of the same type. Assume that different distribution points for public and confidential output exist so that stricter control can be exercised over the release of confidential output. In this case, the operators must separate confidential output from public output by examining the access class banners. An error in bursting or separating the output could result in confidential output being accidentally released with public output. In addition to this security weakness, there is also the operational burden of separating the output according to access class.

The I/O daemon offers a solution to the above problems. Each driver process can be made to handle only requests of a single access class or a range of access classes. Therefore, all public output could be directed to one printer and all confidential output to the other. Hence, both the operational burden and the potential for operational errors mentioned above are eliminated. To facilitate output handling, the public printer could actually be located in the public distribution area while the confidential printer could be located in the confidential distribution area.

The benefits of this automatic separation are not obtained without cost. It is probable that printer utilization and hence turnaround time for output will be somewhat degraded on the whole. This is because it is unlikely that the amount of output will be evenly divided between the access classes. For example, the number of public requests might be much larger than the number of confidential requests. In this case, the confidential printer would be underutilized.

Other disadvantages become evident if one considers the situation where there are fewer printers than access classes. If instead of two printers, only one were available at the hypothetical site, then this printer would have to be switched back and forth between public and confidential output. This switching, of course, increases the operational burden. Also, it upsets the priority selection of requests. Suppose, for example, that the site decides to switch between public and confidential output every 30 minutes. A print request submitted to queue 1 might have to wait this amount of time before being processed. By contrast, if the printer were processing both access classes at once, the request would be performed immediately (assuming queue 1 were empty).

Unfortunately, even with the switching of printers from one access class to another, automatic access class separation of output simply does not scale up for a large number of access classes. Clearly, at some point it becomes impractical to rotate a small number of devices among a larger number of access classes. Therefore, sites using a large number of access classes or sites not willing to tolerate some of the drawbacks cited above may choose to forego automatic access class separation of output. In this case, each device can be made to handle the full range of access classes from system_low to system_high. Care must be taken to ensure proper distribution of output. Control forms can provide a helpful receipt for each piece of output.

Device Classes

The mechanism for separating output according to access class is the "device class." Each request type can be partitioned into any number of separate device classes. One or more devices can be specified for each device class. Also, a range of access classes can be specified for each device class. When a driver process is initialized, the operator normally indicates the device to be run and the request type. However, if device classes are defined for the request type, then the operator must also indicate a device class. This determines the access class range of requests that the driver processes.

It is important to note that the device class of a request is not something the user can specify. In fact, the entire device class concept is invisible to users. Unlike the type and priority queue of a request, the device class is not determined at request submission time. Rather, it is determined at request processing time. Hence, it is possible to modify the I/O daemon tables and change the predicted device classes of requests stored in the queues.

A device class is defined by a device_class substatement. The device_class substatement is treated as a substatement for request types and, as such, can be freely intermixed with other substatements for request types.

device_class: <name>;
: defines the name of a device class belonging to the associated request type and denotes the beginning of a device class description. Any subsequent substatements (see below) apply to this device class until the next device_class substatement or Request_type, Line, or Device statement is encountered. Any <name> may be chosen up to maximum of 24 characters.

If no device classes are explicitly defined for a request type, then a default device class is defined by implication. The primary purpose of a default device class is to allow certain substatements for device class to be specified for a request type when it has no explicit device classes. One such substatement is the device substatement that was previously described under "Substatements for Request Types." In fact, the device substatement is actually a substatement for a device class. When no device classes are defined, the device substatement applies to the default device class for the preceding request type. The same is true of all substatements for device classes.

Substatements for Device Classes

The substatements below describe various attributes of a device class and may appear in any order following a device_class substatement or following a Request_type statement if no device classes are defined.

min_access_class: <access_class>;
: defines the minimum access class of a request to be processed in the associated device class. The <access_class> must be a standard access class string as defined by the convert_authorization_ subroutine. If omitted, the default minimum is system_low.

max_access_class: <access_class>;
defines the maximum access class of a request to be processed in the associated device class. The <access_class> must be a standard access class string. If omitted, the default maximum is the access_class string given in min_access_class.

min_banner: <access_class>;
defines the minimum access class banner to be placed on the head sheet of printed output, on the flip cards of punched output, and on the control forms for all output. Normally, the access class of the request is used. However, if this access class is less than that specified for min_banner, then the min_banner value is used. The <access_class> must be a standard access_class string. If omitted, the default min_banner is the access_class string given in min_access_class.

device: <name>;
specifies a device that can be used to process requests of the associated device class. The <name> must appear as the parameter of a Device statement. More than one device substatement may be specified for a device class.

Care should be taken to ensure that the full system access range (system_low to system_high) is covered by the union of access ranges of the device classes for each request type. (If no device classes are defined for a request type, the max_access_class substatement should be set to system_high for the default device class.) If not, requests of access classes that are not included are never processed. Upon discovering such a request, the I/O coordinator prints an error message and skips the request. Also, it should be noted that if two or more device classes from the same request type have overlapping access ranges, then a request falling in this overlap is assigned to the device class defined first in the I/O daemon tables source file.

As mentioned above, when multiple device classes are defined for a request type, requests are generally not performed in the usual order dictated by priority and submission time. This phenomenon is most noticeable when one device must be shared among several device classes. In order to aid the operators in determining when to switch a device to a different device class, the I/O coordinator keeps track of "waiting" requests. A waiting request is one that is passed over in the normal request selection order while the coordinator looks for a request to satisfy a different device class, or is explicitly requested to run at high priority by an operator command. A count of waiting requests is kept on a per device class basis. When the number of waiting requests for a device class becomes large, this indicates that the device class is receiving inferior service relative to some other device class for the same request type. Thus, operators could be instructed to switch a device to another device class whenever the number of waiting requests reaches some limit. (See the coordinator command, wait_status, in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.)

Substatement for Default Request Type

The default_type substatement described earlier under "Substatements for Devices" names the default request type that a device processes unless overridden by the operator. However, if device classes are defined for the request type, then the parameter of the default_type substatement must include both the request type and device class names separated by a period (e.g., printer.confidential).

Source File Example Using AIM

This example shows a portion of a source file that illustrates the use of AIM features.

```
Request_type:        printer;
  generic_type:      printer;

device_class:        public;        /* for system_low output */
  device:            printer_1;     /* primary public printer */
  device:            printer_2;     /* can use this one in
                                       emergencies*/

device_class:        confidential; /* for output above system_low */
  min_access_class:  level1;
  max_access_class:  system_high;
  min_banner:        level2;        /* all confidential output
                                       has at least a level 2 banner
                                       authorization */

  device:            printer_2;      /*  use only this printer located
                                       in secure area */

Device:              printer_1;
  driver_module:     printer_driver_;
  prph:              prta;
  default_type:      printer.public;

Device:              printer_2;
  driver_module:     printer_driver_;
  prph:              prtb;
  default_type:      printer.confidential;

Request_type:        punch;
  generic_type:      punch;
  max_access_class:  system_high;     /* handle all access classes */

Device:              punch_1;
  driver_module:     punch_driver_;
  prph:              puna;
  default_type:      punch;
```

Standard Driver Modules

A driver module must be specified for each device defined in the I/O daemon tables. A driver module is a program that embodies specific knowledge of how to manipulate a particular device. The standard driver modules provided by the system are described below.

As mentioned earlier in this section, the <string> argument of the args or minor_args substatements are interpreted by each individual driver module. Even though the format of these strings is defined as arbitrary, each of the standard driver modules support a basic <string> having the following syntax:

```
key= value
```

The key must be unique in <string> and acts like a control argument. The value is the argument associated with the key. Keys and values may not contain commas, but may contain spaces. The key/value pairs are separated from one another by a comma. For example:

```
args:       "dim= device_dim_, form_type= xxx";
```

The complete <string> must appear in quotes and standard Multics quoting conventions apply within <string>. The total length of <string> cannot exceed 256 characters.

The following paragraphs describe the args and minor_args keys that are supported by each of the standard driver modules, as well as other attributes of the I/O daemon tables device specification.

printer_driver_ MODULE

This driver module should be specified for standard Multics printers. The prph substatement must be specified for the associated device. Multiple minor devices are not supported and the minor_args substatement is ignored. For standard printer operation, no args substatement need be specified. However, the args substatement can be used to define a nonstandard device interface module (DIM) and/or a nonstandard control terminal accountability form type. This is done by including the following key-value pairs in the args substatement.

```
dim= <DIM_name>
```

The "dim=" key defines <DIM_name> to be the DIM through which the device is attached. The default DIM for printer_driver_ is prtdim_.

```
form_type= <form_name>
```

The "form_type=" key defines <form_name> to be the control form type. If not specified, a default control form type is used.

punch_driver_ MODULE

This driver module should be specified for standard Multics punches. The prph substatement must be specified for the associated device. Multiple minor devices are not supported and the minor_args substatement is ignored. For standard punch operation, no args substatement need be specified. However, punch_driver_ accepts an args statement of the same form as printer_driver_. The default DIM is cpz.

reader_driver_ MODULE

This driver should be specified for standard Multics card readers. The prph substatement must be specified for the associated device. Multiple minor devices are not supported and the minor_args substatement is ignored. For standard reader operation, no args substatement need be specified. However, the following key-value pairs may be specified in the args substatement:

```
dim= <DIM_name>
```

The "dim=" key defines <DIM_name> to be the DIM through which the device is attached. The default dim for reader_driver_ is crz.

```
station= <Station_id>
```

The "station=" key defines <Station_id> to be the name of the card input station to be associated with this card reader. The default station id is "reader". For example:

```
Device:          reader;
driver_module:   reader_driver_;
prph:            rdra;
default_type:    dummy;

Request_type:    dummy;
generic_type:    dummy;
max_queues:      1;
device:          reader;
```

While the reader_driver_ does not process requests from the coordinator, the syntax of the iod_tables requires the presence of a request_type substatement. It is recommended that the punch request type be used for this purpose, as well as for the reader minor device of the remote_driver_.

Sites with CCU (combined card unit) devices should define two devices: one with punch_driver_ for the punch, and one with reader_driver_ for the reader.

spool_driver_ MODULE

This driver module should be specified for a major device that will be used to write user print requests onto tape instead of the printer. The prph substatement must be specified, but the <name> need not be an IOM channel. (It is used as an I/O switch name for the tape attachment.) The default type may be omitted if the operator is required to specify the request type each time the "device" is used. For example:

```
Device:          spooler;
driver_module:  spool_driver_;
prph:           tape;
```

The spool_driver_ ignores all args substatements. It does not accept multiple minor devices and does not accept any control terminal specifications.

remote_driver_ MODULE

This driver module should be specified for all remote printer/punch/reader stations. Two types of stations are supported by the remote driver. A Type I station can be initialized from any one of several communications lines. A Type II station, which does not have an input device, is initialized on a dedicated communications line as a predefined station. The two station types are described separately below because the iod_tables description of each is different.

The driver process must have the dialok attribute in the PDT, and it must have rw access to the access control segment (ACS) of the communications line it will attach. The remote_driver_ can handle one minor device for a card reader and an arbitrary number of minor devices for printers and punches within the limits of the physical remote device and the line protocol. (A minor device for the reader must be specified if the remote device is to read card input.)

The remote_driver_ is designed for maximum flexibility, so its description is rather complex. You should examine the entire subsection before you attempt to set up a remote driver in the iod_tables.

Normal Setup of the remote_driver_ (Type I Stations)

To set up the remote_driver_ for a remote station, you must define a set of communication lines for remote stations, with driver processes listening to each line as stations dial in.

The operator types in the station_id and password via the station command. Once validated, the driver locates the major device that has the same name as the station_id, and begins initialization. If default request types are defined for the minor devices, they are used. If the default request type is omitted for one or more minor devices, the remote station operator is asked to specify the request type. Of course, the minor device must be allowed to use the request type by a device substatement in the Request_type description.

There must be at least one Line statement for each communications line. The Line statement defines the logical line_id and specifies the channel, the attach description (which defines the terminal type), and which stations may use the line_id. For example:

```
Line:              2780_1;
   channel:        a.h001;
   att_desc:       "-tty ^a -terminal ibm2780_ -comm bisync_
                   -ebcdic -ttp IBM2780 -runsp 5 -ttd_limit 2
                   -bretb -multi_record";
   device:         station_a;
   device:         station_b;

Line:              2780_2;
   channel:        a.h002;
   att_desc:       "-tty ^a -terminal ibm2780_ -comm bisync_
                   -ebcdic -ttp IBM2780 -runsp 5 -ttd_limit 2
                   -bretb -multi_record";
   device:         station_a;
   device:         station_b;
```

Each logical line_id (e.g., 2780_1) describes a communications line that a station may dial into. The attach description defines the type of station using the channel. If a single channel (communications line) is to be used for more than one device type, separate line_ids can be defined with the same channel to allow the central site operator to choose the device type during driver initialization.

The attach description string is the one used to attach the teleprinter device, or console, of the station. It is also the basis for the attachment description of the other minor devices. If the attach description for a minor device is to be different from the teleprinter device for that minor device, the attach options may be put into the minor args following a desc= key. Any attach options found in the minor args will override those of the teleprinter device for that minor device.

Each station_id that may use a given line_id is listed as a device in the device substatement of the Line statement, and each must correspond to a major device in a Device statement.

There must be a Device description specified for each station_id. The Device description must include a line substatement with the keyword "variable" specified. This will allow the driver to use some or all of the communication lines defined in Line statements. There must also be minor_device substatements defined for each device attached to the remote terminal. The default_type substatements may be omitted if the remote station operator is to specify the request type for the minor devices. Normally, the Device description will be general enough to allow the station to run any device type, as shown in the following example. (This might not be true if special attach options are defined for one or more minor devices using the desc= key in the minor_args substatement.)

```
Device:             station_a;
   line:            variable;
   driver_module:   remote_driver_;

   minor_device:    prt;
    minor_args:     "dev= printer";
                    /* no default type has been specified */

   minor_device:    pun;
    minor_args:     "dev= punch";
    default_type:   sta_pun; /* makes this rqt required */
                    /* for this minor_device */

   minor_device:    rdr; /* so we can read cards */
    minor_args:     "dev= reader";
    default_type:   sta_pun;  /* just a dummy entry */

Device:             station_b;
   line:            variable;
   driver_module:   remote_driver_;

   minor_device:    prt;
    minor_args:     "dev= printer";
    default_type:   stb_prt;  /* always true for station_b */

   minor_device:    pun;
    minor_args:     "dev= punch";
    default_type:   stb_pun; /* makes this rqt required */
                    /* for this minor_device */

   minor_device:    rdr; /* so we can read cards */
    minor_args:     "dev= reader";
    default_type:   sta_pun;  /* just a dummy entry */
```

Each station may dial in on either Line (see example above); its operating characteristics will be the same. The Device descriptions for the two stations shown have the following difference. Station_a is allowed to specify its printer request type after giving its station command, but station_b will always use the stb_prt request type because this request type is specified in a default_type substatement.

The request type that is used for the default type of the reader minor devices is needed to suppress questions to the operator and to satisfy the syntax rules of the iod_tables_compiler. The request type specified can be any existing request type or it can be a dummy request type used for the reader. No requests will ever be sent by the coordinator for the reader.

A Type I station is always assumed to have an input device, which acts as a slave terminal for the driver. As such, any control terminal definitions associated with the major device will be accepted for the preparation of accountability forms only (see "Terminals that Control the Driver" later in this section). However, a Type II station may accept a control terminal as a slave terminal if specified.

Setup for Stations That Cannot Input Commands (Type II Stations)

Because it has no input device, the Type II station can be identified only by the line it dials into. Therefore, the line substatement for the major device specifies the exact channel name to be used. There are no Line statements associated with this type of station.

```
Device:             station_c;
   line:            a.h003;
   driver_module:   remote_driver_;
   args:            "station= station_c, slave= no,
                    desc= -terminal tty_printer_ -comm tty_;

   minor_device:    prt1;
     default_type:  stc_text;
     minor_args:    "dev= printer, desc= -pll 85 -ppl 66
                     -htab -ttp LA120_10061_8X11";

   minor_device:    prt2;
    default_type:   stc_prt;
    minor_args:     "dev=  printer, desc= -pll 140
                     -ppl 88 -htab -ttp LA120_160L_8X11";
```

A default request type should be specified for each minor device. This is done to avoid making the central site operator answer questions from the driver for each minor device during driver initialization and reinitialization.

Remote Driver <string> Arguments

All the <string> arguments acceptable to the args and minor_args substatements that are defined for the remote_driver_ are described as follows:

a. Arguments which apply to both Type I and Type II stations.

desc= <attach_description>
The desc= key is used to specify additional parameters to the I/O module in the form of an iox_ attach description. This key may be used for any minor_args substatement. Any attach options specified will override attach options for the teleprinter device and/or any attach options which are common to all minor devices. This key is also used in the args substatement for Type II stations, to specify attach options for the teleprinter device. For more information, see the definition of the attach description for the communications module or terminal module associated with the major or minor devices. Also, see the attach options for these device modules: remote_teleprinter_, remote_printer_, remote_punch_, and remote_reader_. For descriptions of these modules and other I/O modules, refer to the *Multics Subroutines and I/O Modules* manual, Order No. AG93.

dev= <minor_device_type>
The dev= key is used to specify the device type of a particular minor device. This key is required for each minor device. The value of minor_device_type must be printer, punch, or reader.

form_type= <ctl_term_form_type>

The form_type= key is used to specify the name of a control terminal accountability form. This is an optional argument and is used for major devices only.

b. Arguments which apply only to Type II stations.

station= <station_id>

When the station= key is used in an args substatement, the driver will accept any station_id (other than blank). The driver will accept any device dialing in on this channel as this station_id without authentication controls. All specified minor devices and default request types will be used. Normally, the value will be the name of the Device (i.e., the station name). This is used for a station without an input device or with a dedicated communications line.

slave= <yes_or_no>

The slave= key value of "yes" is used to tell the driver that it should accept commands from the remote terminal as a slave terminal, as well as from the central site terminal (master terminal). The slave= yes argument can be used to make a Type II station into a Type I station over a dedicated phone line. This key is optional and is only used in the args substatement of the major device. The default is "no".

I/O Modules for Remote Stations

The following paragraphs describe how to define remote stations in the iod_tables using two of the available I/O modules.

hasp_workstation_ I/O Module

The hasp_workstation_ I/O module allows one or more I/O daemon processes to control the devices attached to a remote HASP workstation.

Each device of the workstation should be configured as a separate Type II I/O daemon; the slave parameter of the args substatement for each driver must be given as "slave= no"; the line substatement must specify the appropriate subchannel of the HASP multiplexed channel on which the remote station will be connected.

All commands needed to control the I/O daemons driving the devices of a HASP workstation must be entered by the central system operator.

For more information about HASP and the hasp_workstation_ I/O module, refer to the *Multics HASP Service and Utility Manual*, Order No. GB60.

tty_printer_ I/O Module

The tty_printer_ I/O module will allow the I/O daemon to run a hardcopy terminal as a printer for enter_output_request requests. It is used with the I/O Daemon running the remote_driver_ module for type II Stations. A typical iod_table definition for a polled VIP Station would look like:

```
Device:           vip1;
  driver_module:  remote_driver_;
  line:           b.h006.p01; /* MUX channel for printer */
  args:           "station= vip1,
                  desc= -terminal tty_printer_ -comm tty_
                  -pll 118 -ttp VIP7714 -htab -vtab";
minor_device:     prt;
  minor_args:     "dev= printer";
  default_type:   vip1_prt;
```

The hardcopy device of a polled VIP station is typically an OEM version of a TN1200 with no keyboard. This means that the usable line length is 118 characters. If this line is to be set correctly during driver initialization, the request type definition in the iod_tables should specify an "rqti_seg" which sets the physical line length to 118 (otherwise, remote driver_ will use a default line length of 132 for no rqti segment). The paper_info driver command can be used to correct a bad line or page length setting.

If the hardcopy device uses a VFU tape or wheel for Form Feed (FF) and Vertical Tab (VT) control, set the stops as follows:

```
VT and FF at Line 1
VT only at Lines 11, 21, 31, 41, 51, 61
  (assumes 66 lines per physical page)
```

It is always assumed that the terminal will support FF control characters.

Adding the -vtab option to the attach description of the args keyword will enable vertical tabs to be sent whenever it is more efficient than multiple New Line characters. Some terminals do a top of form function for both FF and VT control characters. Hence, the -vtab option should not be used for these terminals.

Adding the -htab option to the attach description of the args keyword causes Horizontal Tab (HT) characters to be sent instead of multiple space characters whenever possible.

Often, head and tail sheet banners and separator bars are not needed at a polled VIP station (they take a long time to print.) These can be suppressed in the rqti_segment or by the driver commands:

```
banner_type  none
banner_bars  none
```

The tty_printer_ I/O Module can also be used to make the login terminal of a non-system driver act as a printer. You would specify this as follows:

```
Device:             my_prt;
  driver_module     remote_driver_;
  line:             user_i/o;
  args:             "station= my_prt,
                     desc= -terminal tty_printer_ -comm syn_
                     -inhibit close -pll 118 -htab -vtab";

minor_device:       prt;
  minor_args:       "dev= printer";
  default_type:     private_rqt;

Request_type:       private_rqt;
  driver_userid     Person_a.Project_b;
  rqti_seg:         private_rqt_info;
  accounting:       nothing; /* use the nothing command */
  device:           my_prt.prt;
```

This will cause the printer output switch to be connected via syn_ to the user_i/o switch. You can use the -inhibit close attach option to prevent driver commands, which result in device detachment, from detaching the user_i/o switch.

Creation and Maintenance of I/O Daemon Tables

Creation of the I/O daemon tables begins with the preparation of a source segment using the language described earlier in this section. This source segment can be produced with any text editor. As mentioned earlier, the source segment is translated into a binary representation by the iod_tables_compiler command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64). By convention, this command assumes that all source segments have a name ending with the iodt suffix. The standard name for the I/O daemon tables source segment is iod_tables.iodt. When this segment is compiled, an object segment is created with the name iod_tables.

The I/O coordinator looks for the iod_tables segment during its initialization. It expects to find this segment in the directory >daemon_dir_dir>io_daemon_dir. (Normally, the source segment is kept in this same directory, although it is not essential.) The I/O coordinator also looks for a second segment named iod_working_tables. This segment is the working copy of the I/O daemon tables and is the one referenced by driver processes and user processes. The reason for this second segment is to facilitate making changes to the I/O daemon tables. Clearly, the source segment can be modified at any time since it is not referenced by the I/O daemon or by users. Also, the source segment can be recompiled at any time. Doing so changes the iod_tables segment, but not the iod_working_tables segment.

Each time the I/O coordinator is initialized, it replaces the contents of iod_working_tables with the contents of iod_tables. If no iod_working_tables segment exists (as would be the case at a new site), one is created with the contents of the iod_tables segment. Hence, changes to the I/O daemon tables do not take effect until the next I/O coordinator initialization. If an immediate change is necessary, then the coordinator must be logged out and logged in again.

At times it may become necessary to examine the contents of iod_tables, iod_working_tables, or some other object segment produced by the iod_tables_compiler. For example, one might suspect that the iod_working_tables segment has been damaged or one might lose the source segment from which iod_tables was generated. The print_iod_tables command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) essentially performs the inverse translation of that performed by iod_tables_compiler. Given any object segment generated by the iod_tables_compiler, print_iod_tables prints a source language description of that object segment. In fact, if the output from this command is directed to a segment, the segment can be compiled by the iod_tables_compiler to reproduce the object segment.

CREATION AND MAINTENANCE OF I/O DAEMON QUEUES

The I/O daemon queues are created automatically by use of the create_daemon_queues command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64). The queues are created in the same directory as the I/O daemon tables, i.e., >daemon_dir_dir>io_daemon_dir. The command determines what queues to create based on information contained in the iod_tables segment. For each request type, one to four queues are created depending on the value of Max_queues or the per request type max_queues, whichever is in effect. The name of each queue is of the form XXX_N.ms where XXX is the request type name and N is the priority number. The ms suffix indicates that each queue is a ring 1 message segment.

Because the I/O daemon queues are message segments, access to the queues is determined by extended access modes. The IO.SysDaemon identity is given full extended access, i.e., add, delete, read, own, and status (adros) to all queues. For standard system queues (i.e., queues for which the driver_userid of the corresponding request type is IO.SysDaemon), aros permission is given to all users. Otherwise, the assumption is made that the queues are dedicated to the particular project named in the driver_userid. In this case, aros permission is given just to users of that project. The list_acl command can be used to list the extended access on the queues and the set_acl command can be used to change the extended access on the queues.

Changes to the I/O daemon tables must sometimes be coordinated with changes to the I/O daemon queues. In particular, when a new request type is added, new queues must be created for this request type. This can be done as soon as the iod_tables segment has been recompiled. Use of the create_daemon_queues command does not affect any existing queues, but does create new queues for any newly defined request types. If a request type is removed from the I/O daemon tables, the queues are not automatically deleted. The delete command can be used to delete obsolete queues.

MAINTENANCE OF AIM FEATURES

At sites using access classes above system_low, a special awareness is required of the way in which AIM affects the I/O daemon. To begin with, the I/O coordinator should always be logged in at system_high authorization. This is appropriate because the coordinator must distribute requests of all access classes. A driver process, on the other hand, does not necessarily process requests of all access classes. A driver is associated with a device that in turn is associated with a device class. The max_access_class for the device class defines an upper limit on request access classes handled by the driver. Hence, a driver authorization need be no higher than the associated max_access_class.

The access class of the io_daemon_dir directory must be system_low so that users of all authorizations have access to its various databases. The iod_tables and iod_working_tables segments, for example, both have a system_low access class. This implies, of course, that the iod_tables segment can only be compiled at system_low authorization. The I/O daemon queues should have a system_high access class. This is possible because the queues are message segments which, unlike ordinary segments, can have a higher access class than their containing directory. The access class of a message segment is determined by the maximum authorization of the process that creates it. This implies that the create_daemon_queues command should only be used by persons having a system_high maximum authorization. Furthermore, because the queues are created in a system_low directory, the user of create_daemon_queues must have a system_low authorization.

The directories contained in the io_daemon_dir directory are potentially "upgraded," i.e., they may have access classes higher than that of io_daemon_dir. Specifically, the access class of the coord_dir directory equals the coordinator authorization while the access class of a driver directory equals the authorization of the corresponding driver. Thus, at sites using authorizations above system_low, upgraded subdirectories are created in io_daemon_dir. This implies that io_daemon_dir must have a quota so that quota can be moved to upgraded subdirectories (as required by AIM). The coord_dir directory, if upgraded, is assigned a quota of 250 records. Each driver directory is assigned a quota of 2 records if no minor devices are defined, or else 2 records per minor device. The quota initially assigned to the io_daemon_dir directory must take into account the requirements of these subdirectories plus the I/O daemon queues and the other segments in io_daemon_dir. Somewhere between 300 and 350 records is usually sufficient.

REQUEST TYPE INFO SEGMENTS

Each printer request type may have an optional request type info segment (rqti segment) associated with it that defines the physical paper characteristics, the logical VFU channel stops, and some additional driver control data. It is recommended that a special form have a specific request type and thus a separate set of channel stops. The channel stops are set only during driver initialization and remain constant for all requests done by the driver.

In addition, a site may wish to use the request type feature to group requests that use the same VFU tape, regardless of what preprinted form stock is needed for the request. By using the "^auto_print" driver mode, the operator may run requests associated with a given VFU tape (request type) in sequence and change the form stock on the printer to meet the needs of each request.

Printers that have firmware loadable VFC images are loaded by the driver during driver initialization (the paper may have to be realigned by the operator). For printers that use punched paper VFU tapes, the physical VFU tape for the request type must be mounted on the printer at the time the driver is initialized. The driver indicates the number of lines-per-page and the lines-per-inch switch setting that the operator should use.

The size of the head and tail sheets is set automatically to the physical dimensions of the paper as defined in the request type info segment.

The directory named >daemon_dir_dir>io_daemon_dir>rqt_info_segs must give sma access to the administrator and s to all other users. The initial ACL for segments must be set to rw for the administrator and r to all other users. AIM access, for those sites using the access isolation mechanism, should be system_low (the default).

This directory contains all request type info segments. If a single segment describes the paper characteristics for more than one request type, added names may be used in place of separate identical segments. Info segments are only required for printer request types that have the rqti_seg substatement in the iod_tables. When no rqti segment is used, the defaults described for the cv_prt_rqti command are used (see "Syntax for the Request Type Info Source Segment" below).

The printer rqti segments are created by the cv_prt_rqti table conversion command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64). A sample source file is shown in "Example of a Request Type Info Segment" below.

The contents of an rqti segment may be printed by the display_prt_rqti command. This command formats its output so that when directed to a file, the file can be used as input to the cv_prt_rqti command.

Syntax for the Request Type Info Source Segment

The request type info source segment contains keywords that define certain values put into the request type info segment. The general syntax is of the form:

```
keyword:    <value>;
```

where the keyword defines a parameter to be set, and the <value> defines what the value of the parameter is.

The keywords and a description of the values acceptable to the cv_prt_rqti command are defined as follows:

driver_attributes: [^] value {, [^] value...};

The driver_attributes keyword is used to establish some operating parameters for the driver. There are two values defined: auto_go and meter. Each value may be preceded by the character "^" to negate the parameter. The driver_attributes keyword is optional (the default is ^auto_go,^meter).

The auto_go value is used to make the central site or remote printer driver request service from the coordinator immediately after initialization without asking for a go command. For printers on remote stations that are always made ready to accept print files (e.g., where another computer simulates an RJE station), the auto_go value is particularly useful as a means of starting or resuming the processing of print requests without operator intervention.

The meter value is used to tell the driver to maintain internal metering data about its operation. (Note: metering is done according to the driver module design and not all driver modules implement metering.)

driver_wait_time: <number>;

The driver_wait_time keyword is optional (the default is 30 seconds.) It is used to set the time interval that the driver will sleep if there are no more requests in the queues. At the end of the interval, the driver will again ask the coordinator to check the queues for requests. The value is a decimal number between 30 and 300 seconds.

banner_type: standard | brief | none;

The banner_type keyword is optional (the default is "standard"). This keyword specifies to the driver whether the standard head/tail sheets will be printed for each copy of a request, a brief version, or none (separator bars only). The value must be either "standard", "brief", or "none".

banner_bars: double | single | none;

The banner_bars keyword is optional (the default is "double"). This keyword specifies to the driver how the separator bars at the bottom of the head sheet are to be printed. "Double" means overstruck separator bars, "single" are non-overstruck bars, and "none" causes the bars to be suppressed.

prt_control: [^] value {, [^] value...};

The prt_control keyword is used to set some driver request processing modes. There are five values defined: auto_print, force_esc, force_nep, force_ctl_char, and force_nsep. Each value may be preceded by the character "^" to negate its value. The prt_control keyword is optional (the defaults are auto_print, ^force_esc, ^force_nep, ^force_ctl_char, and ^force_nsep).

auto_print

This mode causes the driver to start printing each request as soon as it is received from the coordinator (after a go command has been given). This is the normal mode of operation. When this mode is turned off (^auto_print), the driver goes to request command level immediately after printing the log message. This allows the operator to align the paper, change the paper, print sample pages and issue all other commands allowed at request command level (including the cancel and kill commands).

force_esc

This mode turns on the esc mode of the printer DIM during the processing of each request. This mode must be on if the slew-to-channel functions are to operate. (Note: users cannot set this mode with the enter_output_request command.)

force_nep

This sets the noendpage (nep) mode of the printer DIM during the processing of each request, whether the user has requested that mode or not. This mode should be used for any request type that uses preprinted or preformatted paper (e.g., gummed labels, invoice forms, etc.) This causes the request to be properly formatted even though the user may forget to give the "-nep" control argument to the enter_output_request command.

force_ctl_char

This sets the ctl_char mode of the printer DIM during the processing of each request, which allows an I/O daemon to send control sequences directly to a remote printer instead of discarding the characters or printing their octal equivalents. Setting this mode enables users who prepare print files through Compose to activate special printer features such as superscripting or multiple fonts. This mode is honored only by the remote printer driver module, remote_driver_. (Note: users can not set this mode with the enter_output_request command.)

force_nsep

This mode forces the driver not to produce multiple header and tail sheets for a request which generates multiple copies. Instead, a single header sheet is produced before the first copy and a single tail sheet is produced after the last copy. If this mode is off (^force_nsep), the generation of separators is controlled by users with the enter_output_request command.

message: <"string">;

The message keyword is optional. If specified, the value must be a character string enclosed in quotes, and may include newline characters. This character string must not be longer than 256 characters. Any defined message is displayed on the message coordinator terminal during the initialization of an I/O daemon driver for this request type. Typically, this message would tell the operator to mount some special form stock or which VFU tape number to use for this request type.

paper_length: <number>;

The paper_length keyword is optional (the default is 66.) The value is a decimal number between 10 and 127 which specifies the number of lines on one physical page of the paper. The number of lines depends on the number of lines per inch that is used (see the "lines_per_inch" keyword). This number includes all lines, even though they may normally be used for top or bottom margins. For example, there are 66 lines on an 11-inch page at six lines per inch.

paper_width: <number>;

The paper_width keyword is optional (the default is 136 for local printers and 132 for remote printers). The value is a positive decimal number that specifies the maximum number of character positions on one printed line. A warning message is given if a value greater than 136 is specified.

lines_per_inch: <number>;

The lines_per_inch keyword is optional (the default is 6.) The value is a number that specifies the vertical spacing used by the printer for this request type. The value must be 6 or 8.

line(<line_no>): <ch_1,ch_2,ch_3,...,ch_n>;

The line keyword is optional. There may be one line keyword for each line from 1 to the paper_length. The line keyword specifies which logical VFU channels are defined to stop at <line_no>. There may be 1 to 16 channel stops for any given line, each ch_i is a number between 1 and 16.

For example:

```
line(20):   1,5,11;
```

specifies that a slew to channels 1, 5, or 11 causes the printer to stop at the beginning of line 20.

NOTE: Line 1 is always defined as the form feed position. Typically, the operator positions line 1 at the fourth printable line on a page.

end;

This keyword is required. The end keyword has no value. It specifies the end of the request type info source segment.

Example of a Request Type Info Source Segment

```
/* SAMPLE SOURCE FILE FOR A PRINTER REQUEST TYPE INFO SEGMENT  */
/* Source file:    invoices.rqti                               */
/* Data segment:   invoices                                    */

/* The first two keywords apply to the header data only.  */

driver_attributes:  ^auto_go, ^meter;  /* the default */
driver_wait_time:   30;   /* number of seconds driver will   */
                          /* wait before asking coord again  */

/* The following keywords apply only to the printer_driver_    */

banner_type:        standard; /* normal head/tail sheets */
                              /* otherwise say "brief" or "none" */

banner_bars         double; /* overstruck separator bars */
                            /* can be "single" or "none" */

prt_control:        auto_print, ^force_nep, ^force_esc,
                    ^force_ctl_char, ^force_nsep;
```

```
/* Message to the operator during driver initialization     */

message:
"For the invoices, use VFU tape number 12.
The form stock is in storage bins 22, 23, and 24.";


/* Physical Paper Info */

/* The form stock is only 80 print positions wide and
   72 lines per page at 8 lines per inch                    */

paper_width:        80;       /* default is 136 */
paper_length:       72;       /* default is 66 */
lines_per_inch:     8;        /* default is 6 */


/* Channel Stops */

/* The logical channel stops are defined as follows:        */

line(1):            1;   /* channel 1 is top of form */
line(3):            4;   /* chan 4 is the address line */
line(12):           7;   /* chan 7 is the first entry line */
line(60):           7;   /* and is also the bottom line */


end;
```

OPERATION OF THE I/O DAEMON

The following paragraphs describe the capabilities of the I/O daemon and many of the commands and operating procedures needed to make use of these capabilities. In practice, at most sites, the commands needed for normal I/O daemon operation are contained in the exec_com segments, system_start_up.ec and admin.ec, and they need not be typed by the operator. However, you must become familiar with the material in this section so you can handle special requests and other unusual circumstances correctly.

All I/O daemons (coordinator and drivers) use the iod_overseer_ process overseer. The system administrator should specify this process overseer for each daemon in the project master file (PMF) of the daemon's project. Also, he should specify the "^vinitproc" attribute in addition to this overseer.

I/O daemons set their search rules differently from ordinary users. Instead of using the "default" set of search rules from the system search rules, they use the "io_daemon" set. You can change these by using the set_system_search_rules command in the system_start_up.ec.

I/O daemons running in test mode (via the test_io_daemon command) do not change their search rules. The search rules in effect at the beginning of the test remain in force.

For information on how to operate a HASP I/O daemon, refer to the *Multics HASP Service and Utility Manual*, Order No. GB60.

LOGIN AND INITIALIZATION OF THE I/O COORDINATOR

The step-by-step procedure for logging in and starting up the coordinator is available in the *Operator's Guide to Multics*, Order No. GB61.

COMMUNICATING WITH THE COORDINATOR

The coordinator performs its job automatically without requiring any instructions from the operator. Therefore, it is rarely necessary for the operator to communicate with the coordinator. Occasionally, however, you may wish to issue one of the commands described below under "Coordinator Commands."

Interrupting the Coordinator

It is never necessary to interrupt the coordinator in the course of normal operation. However, in the event of a coordinator malfunction, or other unusual situation, it is possible to send a quit signal to the coordinator. This signal causes the coordinator to suspend its communication with drivers and thus eventually bring all drivers to a standstill. For this reason, the quit signal should not normally be used.

The method for sending a quit signal to the coordinator depends on the coordinator terminal. If the coordinator is logged in from an ordinary terminal, you simply press the proper key to issue a quit signal (e.g., ATTN or INTERRUPT). If the coordinator is logged in from the initializer terminal as a consoleless daemon, you type:

```
quit source_id
```

where source_id is the source name for the coordinator. The coordinator acknowledges the quit signal with the message:

```
"QUIT" received.
```

The coordinator then comes to command level and prints:

```
Enter command:
```

At this point, you may type any of the commands described below under "Coordinator Commands" with the exception of the term command. After each command is processed, the coordinator returns to command level and again prints:

```
Enter command:
```

You should not send a second quit signal to the coordinator at this time. If a second quit signal is received, the coordinator ignores it and points out the mistake by printing the message:

```
io_coordinator:  QUIT already pending.
```

The coordinator should not be left in the quit state for an extended period of time since this effectively halts all active driver processes. You should use the start command to return the coordinator to normal operation following a quit signal.

Coordinator Commands

The following is a list of available coordinator commands.

1. logout

   logs out the coordinator. Normally, all driver processes should be logged out before the coordinator. If driver processes are not logged out, however, they automatically detect the fact that the coordinator has been logged out. The drivers reinitialize and wait for a new coordinator to be logged in.

2. list

   causes the coordinator to print a list of active devices, i.e., devices currently assigned to drivers. The request type and current request number are printed for each active device.

3. print_devices

   causes the coordinator to print a list of all devices managed by the I/O daemon. The devices are grouped according to the request types they service. An asterisk (*) appearing before a device indicates that the associated request type is the default for the device. The driver access name and the driver authorization (if any) are given for each request type.

4. wait_status

   causes the coordinator to print a list of device classes for which requests have been added to the wait list. The number of waiting requests for each of these device classes is also printed. Requests are added to the wait list whenever a driver gives the "next" command, or if the coordinator finds a request for a device class that is not currently active. At sites having only one device class per request type, no requests are automatically added to the wait list. At sites having multiple device classes per request type, requests may be held waiting whenever one or more drivers are active for a request type. By examining how many requests are waiting for various device classes, you can judge when it is appropriate to switch a device from one device class to another so that all device classes receive adequate service.

5. term device_name

   terminates a driver so that the major device (and all minor devices) assigned to it can be assigned to another driver. The device_name for the driver must be specified following the command. Normally, driver termination is performed automatically when a driver logs out. In the case where a driver process terminates abnormally, the coordinator does not discover that the process is terminated until a new driver attempts to log in; then it is unassigned from the old driver process and is assigned to the new driver process. Therefore, the only time it is necessary to use the term command is when you wish to terminate an active driver that cannot be logged out. This might be necessary, for example, if the driver is logged in from a remote location. (If the driver process is running, the term command will not cause the driver to detach the channel associated with the major device. It will cause the driver to eventually fault and probably destroy itself.)

6. restart_status

   causes the coordinator to print the number of restartable requests for each different request series and to identify those request series for which a restart cycle is in progress.

7. start

   returns the coordinator to normal operation following a quit signal.

8. help

   lists commands acceptable to the coordinator.

LOGIN AND INITIALIZATION OF DEVICE DRIVERS

Step-by-step procedures for logging in and starting up printers, card punches, card readers and remote devices are available in the *Operator's Guide to Multics*, Order No. GB61.

TERMINALS THAT CONTROL THE DRIVER

A driver process is capable of receiving commands from two sources: the normal login terminal (master terminal) and a slave terminal. The driver MUST have a master terminal, but a slave terminal is optional. For most devices, a slave terminal is an additional terminal attached to the driver. It is also called a control terminal. Any driver can have a control terminal specified (but it is meaningless for some drivers, e.g., the spool driver). For devices that have a multifunction device, the device itself can act as a slave terminal.

When the system administrator has specified that a control terminal is to be used with a device, the driver is not able to complete its initialization until a control terminal has been attached, except in the case of a remote station.

The slave terminal functions as a source of driver commands and a place to write error messages, operational messages, and log messages.

The control terminal is primarily used to prepare receipts or accountability forms to control the distribution of output. The control terminal will take on the functions of a slave terminal if there is no other slave terminal defined.

The control terminal is not always a slave terminal for a remote station. A Type I station must login with the device providing command input. Hence, the device will remain as the slave terminal, even though a control terminal may be specified for the device. A Type II station may use a control terminal as a slave terminal. Also, by entering "slave= yes" in the device args string of the iod_tables, the system administrator may make the device become the slave terminal even though a control terminal is already attached.

Master Versus Slave Functions

The authority of the master terminal over the slave terminal ensures that central operations has full control of the driver at all times. The slave terminal is provided for decentralization of operational control when this feature is needed. When the driver has a slave terminal, most operational messages, such as requests for commands, are sent to the slave terminal instead of the master terminal.

The master terminal is assured control at command level by not allowing a quit signal to be issued from the slave terminal while the master terminal is executing a command. Also, the master terminal can hold the driver at command level indefinitely if necessary. (See "Standard Driver Commands" later in this subsection.)

Otherwise, the slave terminal can perform almost every function that the master terminal can. When the slave terminal is a control terminal, a short message is printed every time a request is processed. This message can be reformatted at the site to provide a more formal accounting for output generated by a driver. (See "Using Preprinted Accountability Forms on the Control Terminal" later in this subsection.)

Driver Initialization with a Control Terminal

When a control terminal is to be attached by the driver, there is an additional step performed just before the driver comes to command level. The driver waits for the control terminal to be assigned to it by the system control process. The driver may request a specific terminal to be assigned or it may wait for a terminal to "dial" the driver process. Normally, no action is required by the central site operator. Only the control terminal operator is allowed to take action to connect the terminal. The sequence of messages might look like this on the master terminal:

```
Enter command or device/request_type:
! prt_x
prt_x driver waiting for control terminal "<dial_id>" to dial.
```

After the control terminal operator "dials" in the control terminal, the driver continues with:

```
Control terminal accepted.

prt_x driver ready at 02/02/78 0200.0 est Thur
```

Now the driver is at command level.

While the driver is waiting for the control terminal, the control terminal operator must dial the terminal to the driver. First, he must complete the terminal phone connection. After the normal greeting message, instead of using the login command, the operator types the dial command:

```
dial <dial_id> <driver_userid>
```

where <dial_id> is the identifier specified in the driver message above, and <driver_userid> is the login identifier of the driver (normally IO.SysDaemon). (For more details, refer to the description of the dial command in the *Multics Commands and Active Functions* manual, Order No. AG92.) The control terminal operator must know the <dial_id> for the particular driver. However, this is not protected like a password. It only serves to distinguish between the various driver processes with the same <driver_userid>. A password and User_id may also be required.

When the dial command is accepted, a connection message followed by the driver's ready message is printed on the control terminal:

```
prt_x driver ready at 02/02/78 0800.0 est Thur

Enter command:
```

At this point commands are accepted from either the master or slave (control) terminals.

DRIVER COMMAND LEVELS

The driver supports several different command levels, each associated with the function to be performed. These are not the normal Multics process command levels and only limited sets of commands specific to a driver process are accepted. Each command level other than the normal driver command level identifies the function by a word in parentheses following the command request (e.g., "Enter command(quit):" for quit command level). Not all commands may be used at every command level. You should be aware of any command level restrictions identified in the command descriptions below.

Normal Driver Command Level

A driver process indicates that it is at normal driver command level by printing the request:

```
Enter command:
```

The driver comes to normal command level as soon as all initialization is complete and also after each request is finished, if the operator has given commands or set step mode.

Request Command Level

Request command level is used by some device drivers to allow the operator to modify the normal processing of a request. For example, the printer driver uses the request command level to allow the operator to specify the starting page of the request or print sample pages for alignment.

The printer driver comes to request command level when it is running in ^auto_print mode. The remote driver comes to request command level when it is running in ^auto_print mode or ^auto_punch mode. (You can set this mode by using the prt_control command or by entering the correct value in the rqti segment.) Request command level is distinguished from normal command level by the word "request" in parentheses.

```
Enter command(request):
```

The use of request command level is a device specific function. For a list of commands that may be issued from request command level, refer to "Device Specific Driver Commands" later in this section. Most other standard driver commands are also available at request command level.

Quit Command Level

A quit signal is transmitted to the driver in a manner similar to the way it is transmitted to the coordinator (as described earlier). When a quit signal is received, the driver suspends its current operation and comes to quit command level. Quit command level is distinguished from normal command level by the word "quit" in parentheses.

```
* QUIT *
Enter command(quit):
```

Several standard driver commands can only be used at quit command level; they are described below. Most driver specific commands may also be used at quit command level.

STANDARD DRIVER COMMANDS

The two classes of commands for a driver are: standard driver commands and device specific driver commands. The device specific driver commands are described later in this section under "Device Specific Driver Commands." The standard driver commands are described here, grouped by their function. Detailed descriptions of all the driver commands are available in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. general control

ready	makes a device ready to process requests
go	begins processing of requests
halt	stops processing of requests on a device
logout	causes a driver to log out, except for remote drivers
step	causes a driver to wait after each request
hold	holds a driver at command level
new_device	causes a driver to request a new device
inactive_limit	sets time limit for inactivity logout
auto_start_delay	sets wait time between quit signal and start command
defer_time	sets time limit for automatically deferring requests
x	executes site-defined exec_coms

2. control after interrupting a request (quit)

start	resumes driver operation
kill	terminates the current request
cancel	terminates and discards the current request
restart	causes reprocessing of the current request
defer	sends the current request back to its queue
save	saves the current request for possible restarting

3. information

help	lists all driver commands
status	lists current status of the driver

4. coordinator communication

restart N	causes reprocessing of previous requests
save N	saves requests for possible restarting
restart_q	returns to the head of each queue
next	runs a specified request next

5. terminal control

slave_term	controls use of a slave terminal
ctl_term	controls operation of a control terminal
slave	sends a message to the slave terminal
master	sends a message to the master terminal

6. error recovery

reinit	reinitializes the driver
release	returns the driver to normal command level

Some commands perform more than one function. However, these are clearly distinguished by control arguments.

General Control Commands

General control commands (item 1 above) are used at normal command level to initiate and control the operation of the driver. This set of commands is sufficient to run the driver if the operator doesn't encounter any unusual circumstances.

Control Commands after Interrupting a Request

The operator interrupts a request by giving the driver a quit signal. The commands shown in item 2 above can only be used at quit command level, although two have other uses in different contexts. These commands are useful for modifying the driver's sequence of operations:

```
cancel          kill          restart
defer           logout        save
halt            new_device    start
help            ready         status
hold            release       step
```

If the operator hasn't given the driver any commands within 60 seconds following a quit signal, an automatic start command is executed by the driver. You can adjust the 60 second delay by using the auto_start_delay command.

Information Commands

The commands in item 3 above provide additional information to the operator. For device specific driver commands, the help command identifies those commands that may be used for a given driver.

Coordinator Communication Commands

These commands (item 4 above) are used by the operator to instruct the coordinator in how to handle requests. The operator must be able to prevent the loss of requests due to device malfunction. To this end, the coordinator retains each completed request in a "saved" list for a period of time to allow each one to be reprocessed if needed. The operator is able to shift the priority of individual requests.

The coordinator keeps track of the requests in the list by their request numbers. The request number argument to the save and restart commands is used to identify requests to the coordinator. A request number is composed of a request series and a sequential number indicating the order in which the request was processed. For example, request number 50289 is the 289th request processed by the device within the 50000 request number series. Each device or minor device is assigned a series of 10000 sequence numbers during initialization. The first series after coordinator initialization begins at 10001, the second series begins at 20001, and so on. This ensures that each request in the coordinator's "saved" list is uniquely identified.

Commands for Terminal Control

To ensure the master terminal's ability to define the functions of the slave terminal, two commands are provided (item 5 above) to control how the driver treats slave terminal input and output.

Since the slave terminal can be the device itself or an additional terminal, the functions that allow the site operator (or device operator) to control the slave are separated into two commands: 1) those applying to all slave terminals (the slave_term command), and 2) those that only apply to an additional control terminal attached to the process (the ctl_term command).

Error Recovery Commands

The commands in item 6 above are provided for error recovery. There may be circumstances that make the driver unable to continue its operation. This could occur if the coordinator process were terminated or if some control data were destroyed. When the driver can identify the problem, it takes some action, if possible, to correct the situation.

Under some conditions, it may be necessary for the operator to reinitialize the driver or even to log out without completing any pending requests.

DEVICE SPECIFIC DRIVER COMMANDS

The device specific driver commands allow the operator to control the operation of different devices. Each driver module is capable of implementing any commands necessary to control the operation of its device.

Driver modules are designed to be molded by a site into a form necessary to support its own devices, each with its own set of commands. You should familiarize the operator with the commands associated with the driver modules used at your site.

Standard device drivers operate printers and punches and can read card decks from remote multifunction devices. One driver even writes printer requests onto tape (spool_driver_). There are different device specific commands for these generic functions and some additional commands associated with operation of physical devices. The operator can use the help command of the driver to display the full set of commands the specific driver can accept. The device specific commands implemented by standard device drivers are listed as follows:

1. commands for printers:

banner_bars	defines printing of separator bars
banner_type	defines what is printed on the banner
paper_info	defines paper length, width, and lines per inch
prt_control	defines printing control functions
sample_hs	prints a sample head sheet banner
single	single spaces on formfeed and vertical tab

2. commands for printers (request command level only)

copy	sets the copy number of the next copy
print	prints the next copy starting at the current page
req_status	gives status info about the current request
sample	prints a sample of the current page

3. commands for local punches

(no special punch device commands are required; standard commands may be used)

4. commands for remote punches

pun_control	sets the punch control modes (does not apply to the central site punch driver)
sep_cards	controls punching of separator cards between each output deck

5. command for remote punches (request command level only)

copy	sets the copy number of the next copy
punch	punches the next copy of current request
req_status	gives status info about the current request

6. commands for card input

clean_pool	deletes old card decks
read_cards	starts card input

7. command for control of terminal operation (most drivers)

sample_form	prints a sample control form

8. commands for remote device control

pause_time	sets pause time between requests
runout_spacing	sets paper advance after a command request

9. commands for the spool driver

banner_bars	defines printing of separator bars
paper_info	defines paper length, width, and lines per inch
prt_control	defines printing control functions
sample_hs	prints a sample head sheet banner
single	single spaces on formfeed and vertical tab

MAKING THE DRIVER ASK FOR A COMMAND

A command may be entered from the master or slave terminals at any time after the driver has been initialized. However, when requests are being processed continuously, the messages printed on the terminal may interfere with operator input. Therefore, it is better to make the driver ask for a command and wait for the operator to respond. This may be done in two ways:

1. During a pause in terminal printing, the operator may simply press the newline key of the terminal. This causes the driver to ask for a command before processing the next request. (A go command is required to allow the driver to continue.) When using the message coordinator, the operator should send the hold command to the driver process.

2. At any time the operator may issue a quit signal to the driver. This suspends the current request while the driver asks for a command. After a quit signal, if the operator wishes the driver to finish the current request and return to command level, he may give the step command followed by the start command.

   NOTE: When the driver is simulating form feeds on the control terminal, a quit signal terminates form alignment. The driver completes the control terminal message, if possible, before asking for a command. However, the ctl_term aligned command or the sample_form command must be given before the driver can accept a start command. (See "Using Preprinted Accountability Forms on the Control Terminal" later in this section for a more detailed explanation of these commands.)

ENTERING COMMANDS FROM A MULTIFUNCTION DEVICE CARD READER

A card reader in certain multifunction devices can be used as a slave terminal to input commands. Driver commands must be punched on cards, one command line to a card.

USING PREPRINTED ACCOUNTABILITY FORMS ON THE CONTROL TERMINAL

A control terminal may be used to produce accountability records which correspond on a one to one basis with each copy of each request processed by the driver. The format of the accountability record may be redefined by each site for each driver. In some security related applications, this feature may be used to fill in the blanks on preprinted document accountability forms to provide a record of each piece of output.

The format of preprinted forms is likely to be different at each Multics site. The system administrator must ensure that a program is provided to correctly print the request data on each form. He also must specify the form type identifier to be used with the ctl_term command to establish the site program for printing forms. (The default form type for all drivers is a one-line message per request.)

The operator must use the ctl_term form_type command to change the control terminal message to the desired format. Once the new form type has been accepted by a driver, the operator should use the sample_form command to ensure correct alignment of the data on the form. This is normally all that is needed as long as the terminal hardware provides a form feed capability.

Otherwise, the operator uses the form feed simulation functions of the driver to ensure continued alignment. It is very important that the dimensions of the form be specified by the driver command. (The commands that set form feed simulation and form size can be part of the exec_com that initializes the driver.)

The following is an example of the command sequence to simulate form control of a preprinted form with dimensions 8 inches wide by 5 inches long. (Operator input is denoted by an exclamation point (!).)

```
    Enter command:
!   ctl_term  simulate
    Forms will have to be aligned.
    Enter command:
!   ctl_term page_length  30
    Enter command:
!   ctl_term modes 1170
    Enter command:
!   sample_form
      .
    [sample data is printed on the control terminal]
      .
    Enter command:
!   go
```

In this example, there are 30 lines to a 5-inch page and 70 characters to an 8-inch line (allowing for margins). The operator should give the sample_form command repeatedly until correct alignment of the form is achieved. Finally, he should give the go command to begin processing requests.

When the operator issues a quit signal to the driver (from any terminal) or when an unknown command is entered from the control terminal, the form alignment is assumed to be incorrect. Therefore, the driver demands that the operator give the sample_form command (or the ctl_term aligned command) before the next go command (or start command after a quit signal). This situation might look like:

```
    Enter command:
!   goo
    prta driver:  Invalid command for driver - goo
    Enter command:
```

```
! go
  Control forms not aligned.
  Enter command:
! sample_form
    .
  [sample form is printed on the control terminal]
    .
  Enter command:
! go
```

Now the driver can continue processing requests. Form alignment is ensured for all input and output on the control terminal as long as:

1. commands are entered only when requested
2. commands from the control terminal are entered correctly
3. no quit signals are issued
4. the control terminal maintains paper alignment on the platen

LIMITATIONS

With a PRT1200 or PRT1600, 20-lb paper should be used. (The use of lighter weight paper may prove problematic.)

OPERATION OF THE PRINTER DRIVER

Step-by-step procedures for operating the printer driver are available in the *Operator's Guide to Multics*, Order No. GB61.

Processing Requests

If the driver is not running in auto_print mode, the driver comes to the request command level after printing a request, rather than printing the next request automatically. It indicates this by printing:

```
Enter command(request):
```

This is not the normal driver command level. The driver is now ready to accept additional request control commands (plus a help command) to specify the starting page, to print a sample page, or to set the copy number of the current copy. These request control commands are described above under "Device Specific Driver Commands." At this point, the operator should verify that the correct paper stock is on the printer, aligned at the top-inside-page position. The operator may verify the alignment of the paper by printing a sample of the starting page (specified by the operator) before printing the file.

After the operator gives the "print" command, the driver prints a head banner and the text of the file from the starting page to the end of the file and completes the request as described above.

OPERATION OF THE PUNCH DRIVER

Step-by-step procedures for operating the punch driver are available in the *Operator's Guide to Multics*, Order No. GB61.

OPERATION OF THE READER DRIVER

Step-by-step procedures for operating the reader driver are available in the *Operator's Guide to Multics*, Order No. GB61.

Communicating with the Card Daemon

When the central site card daemon requires instructions from the operator, it types:

```
Enter command:
```

This occurs after initialization, after reading an end card, after a quit signal, or after some error condition is encountered. The following commands are understood by the daemon and may be typed on the daemon terminal:

help
: print a short description of available commands.

read_cards
: start reading cards from the card reader. The daemon assumes that the reader is ready (or waits for it after printing a message).

start
: continue the operation in progress after having received a quit signal.

logout
: logout the daemon.

reinit
: attempt to reinitialize the card daemon by detaching the card reader and reattaching it. This command may be used if the daemon appears to be in an inconsistent state.

clean_pool
: delete old card deck copies stored in the system storage areas. This command causes the daemon to ask for the age of segments to be deleted. (This command is normally used at the request of the system administrator or in the event of a record quota overflow.)

Error Conditions

The card daemon attempts to recover from most errors involving incorrectly punched control cards by forward spacing to the next card deck. That is, if an error occurs during the reading of a card deck, that deck is skipped and the reading continues with the next card deck. When the card daemon encounters a deck having an access class (as specified on the ++AIM card) that is different from its own access authorization, the card daemon stops the card reader and requires that the problem be corrected before continuing.

OPERATION OF THE SPOOL DRIVER

The Multics spool driver provides an alternative method for processing users' print requests when the service printer is either down or substantially backlogged. The spool driver obtains queued print requests from the coordinator and writes the requests out onto magnetic tape. The tape can then be processed immediately or at a later time in one of two ways: the spooling tape can be input to a Multics system using the print_spooling_tape command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) to write directly on the printer, or the spooling tape can be input to another system that has software capable of reading and printing the contents of the tape. The spool driver does not implement request command level.

Login and Initialization

The spool driver runs as a standard I/O driver process and can process printer requests. If the printer request type queues are to be used by the spool driver while the printer driver is logged in and working, the following situation arises: two drivers of the same request type share the processing of requests from the same queues in a round robin fashion, the first ready driver getting the next request in the queues. This scattering of print requests can result in the printing of two adjacent requests in the queue at significantly different times. To avoid this request scattering problem, the printer driver should be logged out (or placed in "hold") before bringing up the spool driver.

The spool driver is logged in like any other driver and its operation is selected by the operator when the driver requests:

```
Enter command or device/request_type:
```

The operator responds by typing the name of the spooling device, and either gives a request type or relies on the default request type (specified in the iod_tables).

Any special initialization messages associated with the request type are printed at this point. The line length, page length, and lines per inch to be used in printing the tape contents are printed for the operator.

Next, the operator must enter the tape data. At least one tape volume identifier (tape number) must be supplied, but additional data may optionally be supplied. This optional input includes a recording density and a number of requests

(files) or a number of lines to limit the spooling operations. The spool driver asks for this information by printing:

```
Enter volids and optional tape data or limits:
```

For example, if the operator wishes to spool 95 print requests to volume 070064 to be recorded at 800 bpi, he types:

```
-volid 070064 -density 800 -files 95
```

Spooling Parameters

The operator may make a selection from the following possible input parameters:

-volid STRs, -vol STRs
: where STR is a six-character volume identifier of a tape reel. Up to three volids (separated by spaces) may be specified at one time, and at least one volid must be specified.

-density N, -den N
: where N is either 800 or 1600. If the -density control argument is not specified, and the -interchange control argument is not specified, the default density is 1600 bpi. Density can only be given once during a spooling session or an error is indicated.

-interchange, -int
: specifies tape recording parameters that comply with the ANSI standard requirements for interchange. With this control argument, tape block size is set to 2048 characters and recording density is set to 800 bpi.

-files N, -fl N
: where N is a number between 1 and 999999, indicating the number of files (requests) to be written to tape before stopping. There is no default file limit. If this parameter is omitted, no limit is set on the number of spooling requests.

-lines N, -ln N
: where N is a number between 1 and 999999, indicating the number of printed lines to spool before stopping. There is no default line limit. If this parameter is omitted, no limit is set on the number of lines spooled.

When the coordinator accepts the spool driver as a driver and all the preliminaries of validating the input parameters have been completed, the spool driver prints:

```
Spool driver ready at 01/30/78  1452.8 edt Mon

Enter command:
```

The spool driver is now at normal command level and ready to start processing requests. At this point the operator can modify the paper printing parameters with the paper_info command if desired. All standard driver commands can be used as well as most device specific driver commands for printers. (The spool driver does not support a request command level.)

To begin processing requests, the operator must type the go command. Assuming that some outstanding print requests are queued, the spool driver starts processing requests at the go command. The first print request message is printed on the spool driver log, followed by a tape mount message; after the first tape reel has been mounted, requests continue to be processed and logged sequentially until either the queues become empty or one of the spooling limits has been reached. The spool driver output log looks something like the following:

```
Request 10001 printer q3: >udd>Demo>JSmith>test.
  from JSmith.Demo.a (for "heading" at "destination")
  Charge for request 10001.3:  $1.65 (1055 lines,  10 pages)

Mounting volume xxxxxx with a write ring.
xxxxxx mounted on tape_04.

Request 10002 printer q3: >udd>Demo>js>test1
Request 10003 printer q3: >udd>Demo>js>test2
  .
  .
  .
  .
```

When any spooling limits have been reached, i.e., either lines limit or files limit, the spool driver prints:

```
Reached specified spooling limits;

        Current file limit is xxx
        Current line limit is xxx

        Current file count is xxx
        Current line count is xxx

Enter new file and/or line limits, or "detach":
```

The current file count is a tally of the number of files spooled so far. The current line count is a tally of the number of lines spooled so far. Current line limit is the line limit stop last set. Current file limit is the file limit stop last set. If the limits are zero, then they are not currently set. Each copy that a user requests corresponds to one file spooled, but the limits are approximate, as a request is processed completely before the limits are checked.

At this time, the operator must choose to either enter new spooling limits and continue, or to terminate the spool driver. If new line limits are specified, the new limit is added to the current limit, and spooling continues until that new limit is reached. If only a line limit is specified, the files limit is set to zero and only reaching the line limit halts spooling; likewise, if only a file limit is specified, the line limit is set to zero and only reaching the file limit halts spooling. If both incremented limits are specified, they are both incremented and spooling continues until one of the two limits is reached, whichever one comes first.

To Continue Spooling

If the operator wishes to continue when the spooling limits have been reached, he must renew the limits by entering new -files and/or -lines parameters. The new values are added to the current spooling limits. For example, if the operator types:

```
-files 20 -lines 20000
```

he adds 20 to the current file limit and 20000 to the current line limit and spooling continues.

If the end of a volume is reached when only one volume identifier has been specified, the spool driver asks for additional volume names:

```
Reached end of spooling volume list;
Enter more volids or "detach":
```

Here the operator types in another volume identifier, -volid STR, to continue spooling or types detach to terminate spooling.

To Terminate Spooling

If the operator wishes to terminate spooling when spooling limits have been reached, he types "detach." The spool driver responds with a tally of files and lines processed and then logs out.

Spool Driver Messages

The spool driver automatically answers all questions asked by the tape_ansi_ I/O module. The operator should not have to type answers to any questions from tape_ansi_ that appear in the spool driver log. For example, should a given volume need initialization, the following sequence of lines might appear on the spool driver terminal:

```
tape_ansi_: Volume xxxxxx requires initialization, but
cannot read VOL1 label.
Do you want to initialize it?   yes
```

Spool Driver Commands

The special commands available to the spool driver are a subset of those listed for printers. The spool driver does not have a request command level. The commands available are:

```
banner_bars
paper_info
prt_control
sample_hs
single
```

OPERATION OF REMOTE DRIVERS

Step-by-step procedures for operating remote drivers are available in the *Operator's Guide to Multics*, Order No. GB61.

Processing Requests

If any printer minor devices specify an rqti segment that includes the auto_go driver attribute, those devices will automatically be readied as the driver initializes. The driver will then skip the request for an initial command and immediately look for requests to process for those minor devices.

The driver for a remote station provides several device-specific driver commands that control any reader, printers, or punches. These are issued from normal command level. After a request has been received from the coordinator, the driver can come to request command level. This is enabled by the ^autoprint mode for printers (see the prt_control command) and the ^autopunch mode for punches (see the pun_control command).

The driver can accept commands to alter the processing of the current request while at request command level. The commands are different for printers and punches.

At request command level, a printer device can be adjusted to a specific starting page or copy number and can print sample pages, as well as most other driver commands.

At request command level, a punch device can adjust its copy number. The basic use of request command level for a punch is to make the driver pause after printing the log message, to allow the remote device operator to clear the punch device or redirect the data to a specific file. This is very important for binary output since no separator cards are provided to identify the source, beginning, or termination of the data.

Sending a Quit Signal to a Remote Driver

Many remote terminals do not have "quit" buttons or special commands (for example, "CL" for G115/RCI protocol). Therefore, to stop the driver from printing, the remote station operator must press the STOP button (or equivalent) on the remote terminal. This disrupts the normal communications protocol and causes a quit to be signalled to the driver. This may cause one problem when using a 2780 bisync protocol; since the operator may have stopped the driver while it was printing, it cannot ask questions or print information for the operator.

Sending a quit may cause loss of locally buffered input or output (consult the manufacturer's documentation about the device for more information). The operator can still input commands. The following commands are useful after a quit signal:

cancel
: terminate and discard the current request

defer
: send the current request back to its queue

kill
: terminate the current request

logout
: log out the driver (and get ready to run a new remote station)

reinit
: reinitialize the driver

release
: return to normal command level (this may repeat the current request and may abort any current card input)

restart
: begin the current request over again (printers go to request command level, punches restart at the current copy)

save
save the current request for possible restarting

start
resume whatever the driver process was doing at the time of the quit

If no commands are entered within 60 seconds after the driver receives a quit signal, the driver will automatically execute a start command. For some remote stations, 60 seconds is too little time; you may use the auto_start_delay command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) to increase the delay time. The hold command aborts an automatic start.

I/O DAEMON ADMIN EXEC_COM FORMAT

An I/O daemon admin exec_com provides site-defined driver x command functions. The use of admin exec_coms is optional, but when missing, the driver x command will not work. See "Setting up a Driver to Driver Message Facility" later in this section for the application of the admin exec_com to the creation of a driver-to-driver message facility.

Each I/O daemon admin exec_com is located in the >ddd>idd directory and follows standard exec_com rules. There are two types of admin exec_coms: general and device specific. These differ only in segment name, to allow the site to separate x command functions by device name (station_id for remote stations). The iod_admin.ec segment is the general exec_com and will be used by any driver that cannot find a device-specific exec_com. A <device>_admin.ec segment is a device-specific exec_com for the given major device; for example, prta_admin.ec is specific to device prta. Added names can be used to group several devices under a single device-specific exec_com.

The Multics command iod_command may be used within an admin exec_com to execute arbitrary I/O daemon commands. For example:

```
iod_command defer_time 30
```

may be used in an admin exec_com to change the auto defer time limit for the current driver to 30 minutes. The iod_command command is described in detail in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

When writing an I/O daemon admin exec_com, you must remember that the process that executes it will, most likely, have full SysDaemon access and privileges to the system. Therefore, you must be careful when you choose what functions will be placed at the hands of a remote station operator or an inexperienced device operator.

What follows is a sample section of an admin exec_com. It includes examples of how some iod_val active function keys can be used to protect against operator errors. This sample is for illustration only; see the iod_admin.ec segment supplied in the release for working purposes.

```
& ------------------------------------------------------------
&
& iod_admin.ec    (to be found in >ddd>idd)
&
& This is the exec_com for the IO Daemon driver "x" command.
& The first argument to the "x" command is &1 in this exec_com.
& The standard action is to transfer control to a label
& which will implement the function of &1.
&
& Any arguments associated with an "x" command function begin
& with &2 in this exec_com.

&command_line off
&goto &1.command

&label help.command
&
& For "x help" print a list of x command functions.
&
&print cdr -user Pers.Proj <seg_ident>
&print car -user Pers.Proj <seg_ident>
&print pq {ldr_args}
&quit

&label cdr.command
&
& For "x cdr -user Pers.Proj <seg_ident>"
$ to cancel a dprint request for this driver
&
&if [not [exists argument &2]]
&then &goto missing_arg.error
cdr -rqt [iod_val request_type] &f2
&quit

&label car.command
&
& For "x car -user Pers.Proj <seg_ident>"
& to cancel an RJE job sent by this station
&
&if [not [exists argument &2]]
&then go to missing_arg.error
car -sender [iod_val station] &f2
&quit

&label pq.command
&
& For "x pq {ldr_args}"
& to list all requests that can be processed by this driver
&
&if [exists argument &2]
```

```
&then ldr -a &f2
&else ldr -a -admin -rqt ([iod_val rqt_string]) -tt
&quit

&label &1.command
&
& This is a catchall for any undefined command functions.
&
&print Undefined driver x command function.
&
ioa_ "received command: ^(^a ^)" &f1
&
&quit

&label missing_arg.error
&
&print Expected argument missing. Try again or type "x help".
&
&quit
```

GENERATING A DRIVER PROCESS IN TEST MODE

The following information describes how to generate a driver process in a test environment. This information should be used only as a guide, since it does not cover all circumstances and requirements.

The test environment allows you to test out changes to software and databases (ttt, rqti segments, iod_tables, etc.) normally used by the system coordinator and drivers, both remote and on-site. The test environment includes more detail in error messages, and special commands which control the test process. Full use of the Multics command language is provided, enabling you to set breakpoints using either the probe or the debug command.

Test Directory Structure

The test directory structure is similar to that of the >ddd>idd directory. Throughout this subsection, the directory's pathname is indicated by the term TEST_DIR. The test directory can be located anywhere where you have sma access. Also, some system databases can be shared with those of the system daemons.

If any request type is configured to use a request type info segment (rqti), you must create the rqt_info_segs directory in the test directory. This directory must contain all of the rqti segments you're going to use in the test session.

If you're going to perform card input, you must create the card_pool directory in the test directory. This directory has a different name than the cards directory used by a standard driver and has a different relative location in the drivers' directory structure. This directory must have sufficient quota assigned to it to handle whatever card input you're going to perform. Directories and quota are managed in the same manner as for the >daemon_dir_dir>cards directory.

The segments and directories created by the coordinator in the test directory are identical to those normally created in the >daemon_dir_dir>io_daemon_dir directory (described earlier in this section).

USER GENERATED DATABASES

In TEST_DIR, you must create the segment iod_tables.iodt (described earlier in this section). You must compile this segment by using the iod_tables_compiler command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) to create the iod_tables segment.

If you are going to run the driver from other than an IO.SysDaemon process, you must add the following to the iod_tables.iodt segment for each request type used:

```
driver_userid:      Person_id.Project_id;
accounting:         nothing;
```

where Person_id.Project_id identifies the testing process. The special name "nothing" disables any attempts by the daemon software to actually charge for any requests. is called instead of the charge_user_ subroutine so that actual charges are ignored. If you are testing an accounting routine, you should supply its name.

You are required to create message segment queues for the request types that will be used in the test session. You can do this automatically by using the create_daemon_queues command or manually by using the message segment commands (the create_daemon_queues command and the message segment commands are all described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64). When you use the create_daemon_queues command, you must give the -dr path control argument:

```
create_daemon_queues -dr TEST_DIR
```

If you are testing remote devices, your process must have the dialok attribute in the PDT and correct access to the access control segment for the communications channel or peripheral device.

You may optionally use a different terminal type table (TTT) than the system TTT. Refer to the *Multics Programmer's Reference Manual*, Order No. AG91, for a description of how to set up a TTT.

If you're going to use the x command, you must include an iod admin exec_com or device admin exec_com in TEST_DIR.

SHARED DATABASES

The test process can share some databases with the standard system daemon drivers.

The file PNT.pnt in the >system_control_1 directory is used by the test process to check station identifiers, passwords, and card input users. You may use a test version of PNT.pnt instead by issuing the command:

```
validate_card_input_$test TEST_DIR
```

You may create a test PNT.pnt by issuing the command:

```
create_pnt PNT.pnt
```

after invoking the command above.

The required access control segments for card input are also the same ones used by the system drivers. The testing process must have the same access to these segments as a regular driver process.

Manipulating Requests in the Test Queues

Since the test driver process will be using message segments in the test directory, the enter_output_request (eor), list_daemon_requests (ldr) and cancel_daemon_requests (cdr) commands must be made aware of the test environment. You can do this by calling special entries in each command procedure and indicating the test directory as follows:

```
dprint_$test TEST_DIR
ldr$test_ldr TEST_DIR
cdr$test_cdr TEST_DIR
```

Once this is done, the normal system printer/punch queues are no longer known to the test process. Issuing the new_proc command is one method of restoring access to the normal system queues; you could also issue the above commands with the pathname >ddd>idd.

The Test Process

A standard I/O daemon process operates either as a coordinator or as a driver, and a check is made so that only one coordinator is operating on the system at one time. In test mode, a single test process may perform the functions of both coordinator and driver; or, after one interactive test process has become a coordinator, another interactive process may become a driver. The second interactive process must use the same test directory as the first process. The test processes acting as coordinator and driver are unknown to the standard system I/O daemon processes.

Experimental software should exist in either bound or loose form in the test directory. If one component of a bound object segment is loose, then all components must be loose. You may want to initiate each object segment first.

TESTING A REMOTE STATION

You start the test process by invoking the test_io_daemon command:

```
test_io_daemon -dr TEST_DIR
```

or:

```
test_io_daemon TEST_DIR>test_tables
```

When you run the coordinator and the driver in a single test process, the dialog from this point on looks like the following (your responses are preceded by an exclamation point):

```
Enter command:  coordinator, driver, or return:

! coord

  I/O Coordinator Version: X.X
  I/O Coordinator initialized

! driver

  I/O Daemon Driver Version: X.X
  Driver running in test mode.
  Enter command or device/request type:
```

At this point the driver will accept a device name to run a printer, punch, or Type II remote device, or a listen command to initialize a Type I remote station.

Assuming this remote device can accept command input (i.e., is a Type I remote device), the dialog continues:

```
!  listen g115_1
   Attaching line "g115_1" on channel (b.h002).
```

Responses will be different for Type II devices, but operation is essentially the same.

The test process waits here until the line becomes dialed up, and does not respond to input from the terminal; the only way you can get the process's attention is to issue a quit signal. This will cause the process to print out the following message:

```
Enter command(early quit):
```

A limited set of commands is available at early quit command level, one of which is a help command which lists this set of commands. The process continues waiting for the dialup event from the FNP when you issue the start command. When the dialup event occurs, the following message is printed:

```
Requesting station identifier on line "g115_1".
```

At the same time the message "Enter station command:" is sent to the remote device. The station command must then be entered from the remote device.

After the station command has been given, you may run the process as a normal driver process. However, because the test entry was used, several other commands have been made available to you. One of these is the probe driver command. This simply calls the system probe command. From within the probe command, you may use all the probe command requests, including ".." to execute normal Multics commands.

Within the coordinator/driver test process there exist two pseudo processes stacked above the original interactive process: the coordinator in the middle, and the driver on top. Your terminal communicates with the driver process after you type in "driver" during initialization. If you issue the logout command, you log out only the driver part of the test process; the terminal is then communicating with the coordinator part of the test process. You may now start a new driver servicing the same or another device defined in the test directory's iod_tables. To terminate the test session, issue the return command again, and the coordinator part of the process logs out. You are now back to normal Multics interactive command level.

SETTING BREAKPOINTS

You may wish to set breaks in the software to investigate a problem. You must create a copy of the desired segment and initiate it in the test directory. If the segment normally exists in a bound object segment, you must create copies of all components and initiated them in the test directory. You may also copy the source into the test directory and recompile it with the map and table options. This allows you full use of either the probe or the debug command to investigate the problem.

When using the probe command to set breaks, you may enter probe, set the breaks, and optionally bring up the test driver from within probe. If the test driver is already initialized, you may use the probe command within test mode to enter probe and manipulate break points.

Some errors occurring before full driver initialization invoke probe automatically, while in test mode. You can examine the state of the process at this point. The probe quit request will perform the equivalent of a start command in this case.

COMMAND LEVEL MESSAGES

The standard command level message for the daemon coordinator/driver is:

```
Enter command:
```

Other possible levels are:

```
early quit
quit
request
iodd signal  (test mode only)
```

and are indicated parenthetically in the command level message. For example:

```
Enter command(quit):
```

Sample exec_com File

The following is a sample exec_com which sets up and runs a test environment. When creating your own exec_com, remember to replace TEST_DIR with the absolute pathname of the test directory.

```
&command_line off
&goto &ec_name

&label setup_environment
sa TEST_DIR>** sma [user name].[user project]
sa TEST_DIR>coord_dir>** rw [user name].[user project]
sa TEST_DIR>coord_lock rw
sa TEST_DIR>iodc_data rw
mssa TEST_DIR>([segs *.ms]) adros [user name].[user project]
& Initiate software in test directory at this point.
& set_ttt_path TEST_DIR>TTF.ttt
&quit

&label start_iod
&attach
test_io_daemon -dr TEST_DIR
coord
driver
&detach
&quit

&label use_test_queues
& Call the test entry of the daemon request commands.
dprint_$test TEST_DIR
ldr$test_ldr TEST_DIR
cdr$test_cdr TEST_DIR
&quit

&label use_system_queues
dprint_$test >ddd>idd
ldr$test_ldr >ddd>idd
cdr$test_cdr >ddd>idd
&quit

&label make_tables
& Compile the iod_tables and generate any missing message segments.
```

```
iodtc iod_tables
create_daemon_queues -dr TEST_DIR
&quit
```

Test Mode Commands

A detailed description of the test_io_daemon command, including its requests, is available in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

SETTING UP A DRIVER TO DRIVER MESSAGE FACILITY

In order for one user to send a message to another user, the first user must provide enough information to uniquely identify the second user's mailbox. With the standard send_message command this is accomplished by specifying the user_id of the person to whom the message is being sent. Because most standard drivers at a given site run simultaneously as the pseudo-user IO.SysDaemon, the user_id must be replaced by identification specific to individual drivers in order to enable driver to driver communication to take place.

The unique attribute of a standard driver is the major device it is using. For a remote driver, the station_id is unique. Therefore, to establish driver to driver communication, mailboxes of the form <device>.mbx must be created for each standard driver and <station>.mbx for each remote driver. The mailboxes are located in the directory >daemon_dir_dir>io_msg_dir. You can set up these mailboxes by typing:

```
change_wdir >daemon_dir_dir
create_dir io_msg_dir -access_class system_low
set_acl io_msg_dir s *
change_wdir io_msg_dir
mbx_create (device1 device2 station1 station2 ...)
mbx_set_acl * adrosw *.SysDaemon adrosw *.Driver_Projects
```

The message facility can be extended to all processes by adding the extended ACL term aosw *.*.* to each mailbox. This enables a user process to supply the device operator with a request_id and ask for that request to be run next.

At this point, drivers are able to send messages to specific devices by a command line of the form:

```
send_message -mbx >ddd>io_msg_dir><device> <message>
```

The next step in enabling the message facility is to define commands that allow drivers to communicate with each other. To do this, you edit either the default iod_admin.ec or the device or station specific exec_coms to produce the three new driver commands x am, x sm, and x pm.

To allow drivers to accept messages, add the following to the iod_admin or device/station specific exec_com:

```
&label am.command
&
& for:  x am   -no args needed-
&
am -mbx >ddd>io_msg_dir>[iod_val station_id] -print -call
   iod_driver_message
defer_messages -mbx >ddd>io_msg_dir>[iod_val station_id]
&quit
```

This initializes the mailbox; the driver can then receive messages. The iod_val active function returns the major device (or station_id for, remote device) that was established during driver initialization. Messages are deferred so that a remote site that relies on one printer for both listings and messages will not get messages in the middle of printing a request; for remote sites that do not direct messages to the printer (i.e., that use a separate console for slave or control output), it is also possible to remove the defer_messages command from the exec_com. The iod_driver_message program ensures that messages get to a slave if there is one active.

To allow one driver to send messages to another driver, add the following to the iod_admin or device/station specific exec_com:

```
&label sm.command
&
& for: x sm <station> <message>
&
&if [not [exists argument &2]]
&then &goto missing_arg.error
&if [not [exists argument &3]]
&then &goto conversational_sm
sm -mbx >ddd>io_msg_dir>&2 from driver [iod_val station_id]: &f3
&quit
&label conversational_sm
&print Enter your station_id as the first message line.
&print Type "." to exit send message.
send_message -mbx >ddd>io_msg_dir>&2
&quit
```

To allow a driver to print any pending messages (assuming that they are deferred as shown above), add the following to the iod_admin or device/station specific exec_com:

```
&label pm.command
&
& for: x pm   -no args needed-
&
pm -mbx >ddd>io_msg_dir>[iod_val station_id] -call
   iod_driver_message
&quit
```

The setup for the driver to driver message facility is now complete. Each remote station operator can check for pending messages between requests by giving the x pm command. If messages have not been deferred by the x am command, each message will appear as soon as it is received.

APPENDIX A

SUMMARY OF CONFIGURATION CARDS

The Multics configuration deck is described in detail in Section 7. This appendix presents a listing and brief description of the cards that can be used to make the requisite configuration deck (which can be input from tape or the bootload console). The general and labeled formats of each card are shown.

The various cards of a configuration deck describe five different categories of information:

1. The configuration of major hardware mainframe modules (identified in the list below as "hardware").

2. The configuration of peripheral controllers and devices (identified in the list below as "device").

3. The software parameters related to the configuration in which the Multics system must operate (identified in the list below as "software").

4. The parameters of the Multics storage system (identified in the list below as "system").

5. The cards that have specialized meanings in the Multics system (identified in the list below as "special").

chnl

Format:

```
chnl device_name iom1 chn1 nchan1 {...iom4 chn4 nchan4}
```

Labeled format:

```
chnl -subsys device_name -iom iom1 -chn chn1 -nchan nchan1 {... -iom
    iom4 -chn chn4 -nchan nchan4}
```

designates channels used to access a disk or tape subsystem through an IOM (device)

clok

Format:

```
clok delta zone boot_delta
```

Labeled format:

```
clok -delta delta -zone zone -boot_delta boot_delta
```

provides information to system software about how to interpret the readings of the calendar clock (software)

cpu

Format:

```
cpu tag port state {type} {model} {cache_size}
```

Labeled format:

```
cpu -tag tag -port port -state state {-type type -model model -cache
   cache_size}
```

identifies a central processor in the Multics system configuration (hardware)

dbmj

Format:

```
dbmj max_journals max_pages ast1 ast2 ast3 ast4
```

sets up dm_journal_seg_, and also sets various limits on synch-held pages.

intk

Format:

```
intk boot drive p1 p2 ... pN
```

specifies bootload conditions for bringing up the Multics system (special)

iom

Format:

```
iom tag port model state
```

Labeled format:

```
iom -tag tag -port port -model model -state state
```

describes an I/O mainframe as part of the Multics system configuration (hardware)

ipc

Format:

```
ipc type iom chn nchan
```

Labeled format:

```
ipc -type fips -iom iom -chn chn -nchan nchan
```

defines the channel and the IMU for an IPC-FIPS channel (device)

mem

Format:

```
mem port size state
```

Labeled format:

```
mem -port port -size size -state state
```

defines system controllers that are part of the Multics system configuration (hardware)

mpc

Format:

```
mpc ctlr_name ctlr_model iom1 chan1 nchan1 {... iom4 chan4 nchan4}
```

Labeled format:

```
mpc -ctlr ctlr_name -model ctlr_model -iom iom1
   -chn chn1 -nchan nchan1 {...-iom iom4 -chn chn4 -nchan nchan4}
```

defines channels and the IOM for an MPC in the Multics system configuration (device)

parm

Format:

```
parm parameters
```

defines software parameters (special)

part

Format:

```
part partname subsystem drive{sv}
```

Labeled Format:

```
part -part partname -subsys subsystem -drive drive{sv}
```

informs BCE and Multics of the location of special areas of disk used for various partitions (system)

prph

Format:

```
prph ccuN iom channel model
prph diaN iom channel model
prph dskN iom channel nchan model1 d1 {model2 d2...model5 d5}
prph fnpN iom channel model state
prph opcN iom channel model line_length state {option}
prph prtN iom channel model train line_length
prph punN iom channel model
prph rdrN iom channel model
prph tapN iom channel nchan model1 d1 {model2 d2...model5 d5}
```

Labeled format:

```
prph -device ccuN -iom iom -chn channel -model model
prph -device diaN -iom iom -chn channel -model model
prph -subsys dskN -iom iom -chn channel -nchan nchan -model model1
   -number d1 {-model model2 -number d2...-model model5 -number d5}
prph -device fnpN -iom iom -chn channel -model model -state state
prph -device opcN -iom iom -chn channel -model model -train train
   -ll line_length -state state {-option option}
prph -device prtN -iom iom -chn channel -model model -train
   train -ll line_length
prph -device punN -iom iom -chn channel -model model
prph -device rdrN -iom iom -chn channel -model model
prph -subsys tapN -iom iom -chn channel -nchan nchan -model model1
   -number d1 {-model model2 -number d2...-model model5 -number d5}
```

supplies all necessary data about a peripheral device in the Multics system configuration (device)

root

Format:

```
root subsystem1 drive1{sv} {...subsystemN driveN{sv}}
```

Labeled format:

```
root -subsys subsystem1 -drive drive1{sv} {...-subsys subsystemN
   -drive driveN{sv}}
```

specifies the location of the physical volumes of the RLV (system)

salv

Format:

```
salv keys
```

changes default options for all Multics system salvaging operations (system)

schd

Format:

```
schd wsf tefirst telast timax {mine {maxe {maxmaxe}}}
```

Labeled format:

```
schd -wsf wsf -tefirst tefirst -telast telast -timax timax {-mine mine {-maxe
   maxe {-maxmaxe maxmaxe}}}
```

sets the scheduling factors and parameters in the Multics system configuration (software)

sst

Format:

```
sst ast1 ast2 ast3 ast4
```

Labeled format:

```
sst -4k ast1 -16k ast2 -64k ast3 -256k ast4
```

describes the partitioning of the SST database in the Multics system configuration (software)

tbls

Format:

```
tbls name1 length1 {... name4 length4}
```

specifies the length of certain paged system tables in the Multics system configuration (special)

tcd

Format:

```
tcd apt itt
```

Labeled format:

```
tcd -apt apt -itt itt
```

describes the allocation of the databases that contain information needed by the traffic controller (software)

udsk

Format:

```
udsk subsystem nchan {drive1 count1...drive6 count6}
```

Labeled format:

```
udsk -subsys subsystem -nchan nchan {-drive drive1 -number count1 ... -drive
    drive6 -number count6}
```

specifies the number of channels available for user peripheral I/O on a disk subsystem (device)

APPENDIX B

DPU AND DMP/VIP OPERATING PROCEDURES

This appendix explains how to operate a Dynamic Maintenance Panel (DMP) via a Diagnostics Processor Unit (DPU) or a standard VIP terminal.

MULTICS DPU OPERATION

Powering on the DPU

Note: never power the DPU on or off with the diskettes in the drives.

1. Open the door in the front of the DPU, and find the DPU Maintenance Panel. Switch on the DPU and the diskette drives.

2. Switch on the VIP7205 terminal on top of the DPU.

Booting the DPU (Manual Boot)

1. At the DPU Maintenance Panel, switch the security key to the unlock position. You'll know the key is in the right position when you see the numbers on the panel light up. When this is done, the panel is enabled.

2. Depress the following keys in the order given:

```
STEP      (key marked with red "S")
CLEAR     (white key marked "CLR")
LOAD      (key marked "L")
EXECUTE   (red key marked "E")
```

   When this is done, the Quality Logic Tests (QLTs) of the DPU are invoked. Both the TRAFFIC and the CHECK lights will go on.

3. Watch to see that when the TRAFFIC light goes out, the CHECK light also goes out. If it doesn't, there's something wrong with the DPU, and you should call CSD.

4. Check to see that the D1 register contains "0400" by using the following procedure:

   a. If the WRITE and READ lights are on, turn them off by depressing the key marked with a red "S" below the STEP light.

   b. If the CHANGE light is on, turn it off by depressing the key marked "S" below the CHANGE light.

c. Depress the key marked "D", then the key marked "1", on the data entry pad on the right side of the panel.

d. "D1" should be displayed in the left hand set of lights (under LOCATION), and "0400" should be displayed in the right hand set of lights (under CONTENTS). If they aren't, you should call CSD.

5. Check to see that the E0 register contains "0002" by using the following procedure:

a. If the WRITE and READ lights are on, turn them off by depressing the key marked with a red "S" below the STEP light.

b. If the CHANGE light is on, turn it off by depressing the key marked "S" below the CHANGE light.

c. Depress the key marked "E", then the key marked "0", on the data entry pad on the right side on the panel.

d. "E0" should be displayed in the left hand set of lights (under LOCATION), and "0002" should be displayed in the right hand set of lights (under CONTENTS). If they aren't, you should call CSD.

6. Mount the DPU system diskette in drive 0 (the one on the left). (Push the button to make the door slide up. Slide the door down to close it). Depress the EXECUTE key (red key marked "E"). The red light on the drive will light up.

7. After about three minutes, the bootload will be complete. Diskette activity will halt, (i.e., lights will stop flashing) and the following message will be displayed on the terminal (blank lines are not shown here):

```
***DIAGNOSTICS PROCESSOR UNIT A.X (DL6.XX)***
RMI ACTIVE  <remote maintenance interface status>
C?          <ready prompt>
```

You are done with the DPU Maintenance Panel now.

Booting the DPU (Alternate Boot)

If the primary diskette reader (drive 0) doesn't seem to be functioning, you can do an alternate boot of the DPU by performing the following steps:

1. Perform steps 1-3 given above.

2. Change the D1 register to "0480". (Consult with CSD for the proper procedure.)

3. Mount the DPU system diskette in the other drive (the one on the right). Complete steps 6-7 given above.

You should note that when a diskette reader doesn't seem to be functioning, the problem may lie with the diskette itself, not the reader.

DPU Typing Conventions

All input to the DPU must be in upper case. The easiest way to accomplish this is to depress the CAPS LOCK key and leave it depressed for the remainder of your DPU session.

All input to the DPU, including the use of special controls such as the DEL key and the function keys, must be terminated by pressing the RETURN key. This is true no matter what mode the DPU is in.

WARNING: pressing the RETURN key without typing a new command causes the previous command to be reexecuted. In other words, when you've executed a command on the DPU by typing its name and hitting the RETURN key, if you hit the RETURN key again later on, the command will be executed again. This is true only in VIP mode (with the "UNT CMD" prompt), not in general (not with the "OFL?" and "C?" prompts).

Exceptions to the above rules are the character and line kill keys. They are not typed in upper case, and are not followed by a carriage return. To delete an input character, type "@". To delete an input line, type CTL-X. (This is done by depressing the key marked CTL and keeping it depressed while you type an X.)

Installing the Site Configuration

Once you've received a ready prompt on the terminal, the first thing you should do is install the site dependent configuration that is cabled to the DPU.

Each unit-type (or device, e.g., CPU, SCU, IOM) that is cabled to the DPU has an assigned keyname that is used by the DPU to uniquely identify that unit-type. The keynames are as follows:

```
CPMxx  DPS 8 Processor
SCUxx  DPS 8 SCU
IOMxx  DPS 8 IOM
```

where xx is a two-digit number from 00 to 99 that is used to uniquely identify a specific unit-type in a multi-unit configuration.

To install the site configuration, the following steps must be taken:

1. Verify the DPU cabling from the DPU communication channels to the DMP connections in each unit-type. Line 0 is reserved for the RMI modem connection, lines 1-7 for seven unit connections, and lines 8-15 for eight more optional unit connections. Record the channel number, the unit-type connected to it, and the keyname of the unit-type.

2. Use the CLST command (type "CLST") to display the currently active DPU configuration. It will print a listing like the following:

```
Chnl    Unit      Keyname    Comments

1000    RMI       REMOT      Reserved by DPU operating system.
                             No current requirement to use or
                             implement this connection.
0500    VIP       LOCAL

1080    CPU-A     CPM00      Can be arranged in any order
1100    CPU-B     CPM01      to suit a site's needs.
1180    SCU-A     SCU00
1200    SCU-B     SCU01
1280    IOM-A     IOM00
```

3. Compare the currently active configuration with the information you recorded in step 2. If there are differences, perform steps 4 and 5. If not, installing the configuration is complete.

4. Use the CBLD command (type "CBLD") to make any changes to the DPU configuration that may be necessary. The CBLD command supports the following options:

   BUILD -
   provides a "fresh start" capability by clearing the previous configuration.

   ADD -
   provides an append capability by preserving the previous configuration and adding to it with the prompted-for information.

   CHANGE -
   allows the current configuration to be changed by accessing the current configuration and allowing alteration of either the keyname or channel number.

   LIST -
   lists the contents of the modified configuration file.

   DONE -
   signifies normal termination of the CBLD session and lists the new configuration file.

   ABORT -
   terminates the CBLD session. The current configuration may have been modified, so an automatic list of the configuration is displayed.

5. If you change the configuration, you must reboot the DPU to get it to use the new or modified version. You can do this by repeating steps 1-6 under "Booting the DPU (Manual Boot)" or by using the BOOT command, (type "BOOT"). Either of these methods will make the changes made to the configuration by the CBLD command become effective. Note that the CLST command will not display any changes made by the CBLD command until after a reboot is performed.

Displaying Configuration Panels

Once the configuration is installed correctly, you are free to perform whatever tasks you need to accomplish. To display a unit's configuration panel, the following steps must be taken:

1. Use the OFL command to indicate which unit-type you want to work with. Type:

   ```
   OFL <unit>xx
   ```

   where <unit>xx is the keyname of the unit-type, as defined in the configuration. For example:

   ```
   OFL CPM00
   ```

2. After initializing, OFL will display the following message:

   ```
   RD CMD FILE
   ```

   This indicates that it is reading in the command file for the unit-type you specified. After loading the command file, OFL will display the following message:

   ```
   OFL?
   ```

   This is the transparent (TM) mode input prompt. When you see this, type "VIP".

3. VIP will display the following message:

   ```
   <unit> CMD
   ```

   where <unit> is the unit keyname (CPM, SCU or IOM). This is the VIP mode input prompt. At this point, your actions depend on which unit you're working with.

   If you want to display the configuration panel of the CPU or the SCU, depress function key 1 (the key marked "F1") or type "CF".

   If you want to display the configuration panel of the IOM, type "CFG".

4. Displays which exceed a full screen are automatically stopped at full screen intervals. To make a display continue, type a space. To make a display stop, depress the DEL key.

   If you try to interrupt a display before it finishes filling a screen, indeterminate conditions may occur.

   You should use continuous, repeated, multiple commands with caution, to avoid skewed or distorted displays.

5. Return to TM mode by typing "TM", or depressing the BRK key.

6. To exit from TM mode, type "QUIT" or depress the BRK key. You must exit from TM mode to display another unit's panel, then repeat steps 1–6.

Performing System Recovery

Step–by–step procedures for executing fault and executing switches on a DPS 8 system with both a DPU and a DMP/VIP are available in the *Operator's Guide to Multics*, Order No. GB61.

Displaying the SCU History Registers

If a SCU has its HRD light on after a system crash, or at any other time when it may be helpful to see the contents of the history registers, you should perform the following steps:

1. If the DPU is already connected to the SCU with the HRD light on, verify that the current unit is the desired unit with this procedure:

   a. Depress the BRK key.

   b. When the OFL prompt is displayed, type ".U".

   c. When the unit identifier is displayed, check to see that it is that of the desired SCU. If it is, continue with step d. If it isn't, continue with steps e–f below.

   d. When the OFL prompt is displayed, type "VIP".

   If the DPU is not connected to the SCU with the HRD light on, switch the current unit to the desired unit with this procedure:

   e. Depress the BRK key.

   f. When the OFL prompt is displayed, type "QUIT".

   g. When the C prompt is displayed, type "OFL SCUxx", where xx is the unit number in the DPU configuration of the desired SCU.

   h. When the OFL prompt is displayed, type "VIP".

2. Having typed "VIP", you will receive the SCU CMD prompt. When you see this, depress function key 3 (the key marked F3).

3. The contents of the history registers will be displayed on your screen. Follow the rules for dealing with full screen displays given in step 4 of "Displaying Configuration Panels".

DPU Command Summary

The following summary of commands is not complete. If you need more information, you should contact the Customer Services Division (CSD) or your Customer Services Representative (CSR).

DPU COMMANDS (C? prompt)

CLST
: displays the currently active DPU configuration. It will not display any changes made by the CBLD command before a software reboot is performed.

CBLD
: alters the DPU configuration.

BOOT
: causes the DPU to perform a software reboot. This is usually done to make changes made to the configuration by the CBLD command effective.

OFL
: enters TM mode, an environment where you can use keyboard commands to display configuration panels and perform certain recovery procedures. This environment is a result of the DPU/DMP interface.

?
: gives you help with DPU commands.

TM MODE COMMANDS (OFL? prompt)

.U
: displays the current unit identifier (e.g., CPM00).

VIP
: enters VIP mode

SUSP
: exits TM mode, retaining the step condition. This is required if you are putting more than one processor in step at a time.

QUIT
: exits TM mode. This is required if you want to display another unit's configuration panel.

<BRK>
: also exits TM mode.

VIP MODE COMMANDS (<unit> CMD prompt)

CPU DMP Commands

<F1>
: displays the configuration switches of the CPU.

CF
: displays the configuration switches of the CPU.

ST CU
: places a processor in step.

BCE 24000
causes an EXECUTE SWITCHES function to occur, and automatically takes the bootload processor out of step and restarts it.

BCE 24002
forces an ESD.

SCU DMP Commands

<F1>
displays the configuration switches of the SCU.

CF
displays the configuration switches of the SCU.

<F3>
displays the contents of the history registers of the SCU. This is useful in determining why the HRD light is lit on the maintenance panel.

IOM DMP Commands

CFG
displays the configuration switches of the IOM.

MULTICS DMP/VIP OPERATION

Getting the VIP Connected to the DMP

1. If each unit (CPU, IOM, SCU) has its own terminal, you can just type on the one connected to the unit you want to work with. Otherwise, you must get your terminal hooked up to the desired unit before you can begin typing.

2. Enter several carriage returns. This will enable the DMP to determine the baud rate of your terminal. Once the DMP has done this, you will receive the UNT CMD prompt, indicating that you are connected to the DMP and in VIP mode.

Using the DMP/VIP

Being in VIP mode on the DMP/VIP is the same as being in VIP mode on the DPU. The difference is that you don't have to install the site configuration and go through the DPU and TM modes first.

All of the commands listed under "VIP Mode Commands" above are available to you. There's only one difference between using these commands on the DPU and using them on the DMP/VIP. On the VIP, you don't have to type a carriage return after you depress a function key.

You can use the DMP/VIP to display the configuration panels of the CPU, SCU, and IOM, to perform system recovery, and to display the contents of SCU history registers.

APPENDIX C

STARTUP CHECKLISTS OF SWITCH SETTINGS

The lists that follow summarize switch settings for all major module panels on the Honeywell Multics System prior to Multics startup. (The major module panels are described in Section 3.) All switch settings should be checked before the Multics system is brought up.

SYSTEM CONTROLLER UNIT CONFIGURATION PANEL SWITCHES (6000 SC)

Switch	Position
MAINTENANCE PANEL MODE	NORMAL
STORE A	ON LINE (if used)
STORE B	ON LINE (if used)
SIZE	Set to size available for Store A and B
EXECUTE INTERRUPT MASK ASSIGNMENT	One switch set to the port for each CPU that can be configured. Unused switches are OFF.
LOWER STORE	Whatever is applicable
INTERLACE	If both stores equal, then ON; if stores unequal, then OFF.
CYCLE PORT PRIORITY	Use if applicable. All switches between CPU ports and between IOM ports UP, others DOWN.
PORT CONTROL	ENABLE ports being used. Ports for processors other than the bootload CPU should be in the PROG CONT position.

All other switches on the SCU configuration panel should be left in the OFF or down position.

SYSTEM CONTROLLER UNIT CONFIGURATION PANEL SWITCHES (4MW SCU)

Switch	Position
MAINTENANCE PANEL MODE	PROGRAM
STORE A	ON LINE (if used)
STORE A1	ON LINE (if used)
STORE B	ON LINE (if used)
STORE B1	ON LINE (if used)
LWR STORE SIZE	Set to size available for Store A and Store A1.
INTERLACE	ON if both stores equal in size; otherwise OFF.
LWR STORE	Whatever is desired.
MASK/PORT ASSIGNMENT	Mask A set to port for bootload CPU. Mask B set to OFF.
PORT ENABLE	Ports for all IOMs and the bootload CPU ON; other ports OFF.
ALARM	ENABLE
MODE	PROGRAM

All other switches on the SCU configuration panel should be left in the OFF or down position.

CENTRAL PROCESSING UNIT CONFIGURATION PANEL SWITCHES

Switch	Position
PORT ENABLE	(Low-order memory) ON
INITIALIZE ENABLE	(Ports used) ON
ASSIGNMENT	(Low-order memory) 000 (↓↓↓); others as appropriate.
ADDRESS RANGE on L68; not present on DPS8	FULL or HALF, as appropriate.
ALARM	NORMAL on L68; ENABLE on DPS8.
MAINTENANCE PANEL MODE	NORMAL on L68; PROCESSOR on DPS8 (in TEST position enables the EXECUTE CONTROL portion of the processor maintenance panel).
FAULT CONTROL on L68; PROCESSOR FAULT BASE ADDRESS on DPS8	100 (octal); switch 11 UP.
PROCESSOR NUMBER	000 (↓ ↓ ↓) = CPU A (PRO-0) 001 (↓ ↓ ↑) = CPU B (PRO-1) 010 (↓ ↑ ↓) = CPU C (PRO-2) 011 (↓ ↑ ↑) = CPU D (PRO-3) 100 (↑ ↓ ↓) = CPU E (PRO-4) 101 (↑ ↓ ↑) = CPU F (PRO-5) 110 (↑ ↑ ↓) = CPU G (PRO-6) 111 (↑ ↑ ↑) = CPU H (PRO-7)
OPERATING MODE	MULTICS on L68; VMS on DPS8.

All other switches on the processor configuration panel should be left in the OFF or down position.

CENTRAL PROCESSING UNIT MAINTENANCE PANEL SWITCHES

Switch	Position
ENABLE MATCH (two switches)	ON (up)
DATA SWITCHES	Set to XED - Location 24000 (024000 717200)
INITIALIZE & CLEAR/INITIALIZE CONTROL	INITIALIZE CONTROL
AUTO (Step control section, center of panel)	OFF
CYCLE (Step control section)	OFF
EXECUTE PB/SCOPE REPEAT (Execute control section)	EXECUTE PB (UP)
EXECUTE SWITCHES/EXECUTE FAULT	EXECUTE FAULT (DOWN)

All other switches on the processor maintenance panel should be left in the OFF or down position. On DPS8 systems, this panel has been replaced by a display.

IOM CONFIGURATION PANEL SWITCHES

Switch	Position
PORT ENABLE	(Ports used) ON
INITIALIZE ENABLE	(Ports used) ON
ASSIGNMENT	(Low-order memory) 000 (↓↓↓); others as appropriate.
ADDRESS RANGE on L68; not present on DPS8	FULL or HALF, as appropriate
ALARM	NORMAL on L68; ENABLE on DPS8
MAINTENANCE PANEL MODE on L68; IOM on DPS8	NORMAL
IOM BASE ADDRESS	1400 (octal) IOM A 2000 IOM B 2400 IOM C 3000 IOM D
INTERRUPT BASE ADDRESS	1200 (octal)
IOM NUMBER	00 (↓↓) = IOM A (IOM-0) 01 (↓↑) = IOM B (IOM-1) 10 (↑↓) = IOM C (IOM-2) 11 (↑↑) = IOM D (IOM-3)
SOURCE	TAPE
CHANNEL NUMBER CODE on L68; CHANNEL SELECT on DPS8	Tape channel number.
ZERO BASE S.C. PORT NO. on L68; SCU PORT NUMBER on DPS8	Set switches to reflect the SCU port to which the IOM is cabled
OPERATING MODE	PAGED

All other switches on the IOM configuration panel should be left in the OFF or down position.

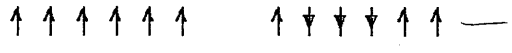
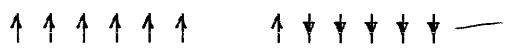
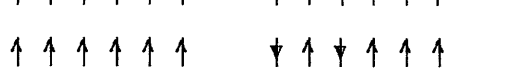
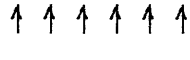
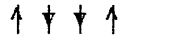
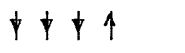
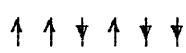
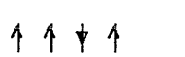
IMU CONFIGURATION

MCA configuration files perform the same function for the IMU that switch settings perform for the IOM. The values placed in the MCA configuration files are similar to the values set in the IOM switches. These values are placed in the MCA configuration files via the MCA config command. If you're using a configuration file other than the assigned default, you must give the name of the file with the MCA iboot and init commands. MCA commands are described in the *Information Multiplexer Unit Hardware Operations Manual*, Order No. 58010010. An example of an IMU config file looks like

```
CONFIG.          SOURCE   INTERRUPT   MAILBOX
PATHNAME         DEVICE   BASE ADDR   BASE ADDR
.CMULTIC          :SI:     001200Q     002000Q

IPC              DSK HW FW FIRMWARE TOT PRI.
PAN DESC STATE   TAB ID ID PATHNAME BUS CHAN. VARIANTS:
00  PSI4  150    003 15 -- -------- 0   20-23 RSO REQ=N,
01  NDIC  150    003 09 -- -------- 0   18    FNP=DATANET 6670
02  NDIC  150    003 09 -- -------- 0   19    FNP=DATANET 6670
03  CONS  150    003 03 -- -------- 1   14    MAS=Y,R.MAINT=N,POLL 01 TO 01
04  PSI2  150    003 01 -- -------- 0   16    RSO REQ=Y,
05  PSI2  150    003 01 -- -------- 0   28-31 RSO REQ=N,
06  FIPD  150    003 06 09 IFDK00C0 1   40-43
07  FIPT  150    003 06 08 IFTP00B0 1   48-49
08  PSI4  150    003 15 -- -------- 0   24-27 RSO REQ=N,
09  NDIC  150    003 09 -- -------- 0   17    FNP=DATANET 6670
10  NDIC  150    003 09 -- -------- 0   15    FNP=DATANET 6670
11  PDSI  150    003 02 04 IPR540A0 2   08-09
12  DAIC  150    003 04 02 ICRD00A0 2   10-11
13  PSI2  150    003 01 -- -------- 0   32-35 RSO REQ=N,
14  PSI2  150    003 01 -- -------- 0   36-39 RSO REQ=N,
15  FIPD  150    003 06 09 IFDK00C0 1   44-47
17  MP-B  150    003                          04096 - 08191K,INIT=N
18  MP-C  150    003                          08192 - 12287K,INIT=N
19  MP-D  150    003                          00000 - 04095K,INIT=N
20  IMU1  150    003    15 IIMUC0G0
21  PS    150    003
BOOTHOST  150 OSTYPE=MULTICS, PORT=1, CHAN=16, DEV=00, SRC=2(TAPE)
              PTW WRITE PERMIT CHECK IS DISABLED.
23  MCA       003 TRACE (FAULT=Y, BOOT=N, DEBUG=N)
                  FAULT CONDITION (CLASS B=Y, CLASS D=N)
NOTE 1: * = ALTERS WILL BE APPLIED TO FW PATHNAME.
***END CONFIGURATION LIST***
```

FNP DIA SWITCHES (DN6670)

6000	MAILBOX	
	FNP A	
	FNP B	
	FNP C	
	FNP D	
	FNP E	
	FNP F	
	FNP G	
	FNP H	
6000	TERMINATE	
6000	EMERGENCY	
HNP	MAILBOX	
HNP	TERMINATE	
HNP	SPECIAL INTERRUPT	

Each FNP may be configured with one or two DIA boards. Each board must be configured on a separate FNP port. The FNP ports on which DIA boards may be configured are 3,. 4, 5, and 14. Each board's FNP port must be cabled to an IOM channel. The FNP DIA connections are identified in the Multics config deck by the IOM channel to which the FNP port is cabled. The Multics software determines which FNP port to use in accessing the FNP by references to the IOM channel cabled to the active FNP port. The FNP port number is not recorded in the FNP core image, nor in Multics supervisor databases, nor on any Multics config card.

A FNP with two DIA boards can be cabled to two different IOMs on a single Multics system, or to an IOM on each of two different systems. However, only one of the DIA boards may be used at a time. The prph fnp config card for the IOM channel cabled to the active DIA must have a state of on; the card for the IOM channel cabled to the inactive DIA must have a state of off.

Cabling a FNP to two different IOMs on a single Multics system offers a measure of improved reliability. If the IOM attached to the active DIA board breaks down, the IOM and its attached FNP can be deleted from the system, and the FNP can then be added to the system using the other IOM channel. However, users of the FNP at the time of the IOM failure will have to login again. If their processes had the save_on_disconnect attribute, they will be able to reconnect to their processes and continue the work which was interrupted when the IOM failed. If their processes did not have the save_on_disconnect attribute, work in progress when the IOM failed will be lost.

Cabling a FNP to the IOMs of two different systems allows the FNP to be shifted easily from one system to the other.

Multics requires that each FNP use a paging mechanism to access FNP memory beyond the first 32K words of memory. The paging mechanism on the FNP pager board can be disabled for testing purposes, but Multics requires that it be enabled during normal operations. Contact your CSD representative if your FNP will not operate. Ask him to insure that the paging mechanism is fully operative.

APPENDIX D

NAMES OF COMMUNICATIONS CHANNELS

The name of a communications channel is an encoding of the information describing the physical connection between the system and a remote I/O device. Every such name is a string of 6 to 32 characters. The name is divided into components separated by "." characters; each component represents a level of multiplexing.

The first two components have a standard form, and describe a physical channel on an FNP. Multiplexed channels (i.e., subchannels of a concentrator whereby multiple terminals are supported on a single FNP channel) have additional components identifying the individual subchannels. The form of each component depends on the type of multiplexer involved.

The general form of the name of a physical channel is:

```
F.ANSS
```

where:

F
: is an FNP identifer. It may be a, b, c, or d.

A
: must be h, to indicate a channel of a high-speed line adapter (HSLA).

N
: is the number of the HSLA on the specifed FNP. It is in the range 0 to 2.

SS
: is a 2-digit decimal number identifying a subchannel of the specifed HSLA.

Here are some examples:

```
F ANSS
a.h003      FNP a, HSLA 0, subchannel 03
b.h219      FNP b, HSLA 2, subchannel 19
c.h111      FNP c, HSLA 1, subchannel 11
```

In the following examples, the physical channel b.h108 (i.e., FNP b, HSLA 1, subchannel 8) is assumed to be a concentrator whose subchannels are numbered sequentially from 0 to 15:

```
b.h108.00      subchannel 0  (first subchannel)
b.h108.03      subchannel 3
b.h108.15      subchannel 15 (last subchannel)
```

APPENDIX E

CONTINUOUS OPERATION EXEC COMS

This appendix describes the BCE exec_coms supplied with the system to implement automatic recovery after system crashes. The operator usually types only the two command lines:

ec auto star
: to initiate system bootload, with automatic restart if a crash occurs, and the system is in unattended mode.

ec go
: to restart automatic operation after a manual return to BCE.

FLAG USAGE

Several flags and indicators coordinate the BCE and Multics modes of operation. The BCE and Multics get_flagbox and set_flagbox commands (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) are used to examine and set, respectively, flags in the toehold. Four flags have preassigned meanings and are known by keywords in these commands:

Flag 1:
: auto_reboot
TRUE if the system is to attempt to reboot itself after it has crashed.

Flag 2:
: booting
TRUE during bootload. It is turned off at the end of part 3 of system_start_up.ec, when bootload is over. This flag prevents the system from looping attempting to reboot if it crashes before it comes up.

Flag 4:
: rebooted
TRUE if the system has rebooted as a result of automatic operation.

Flag 5:
: unattended
TRUE if the system is not attended by an operator.

In addition, the "call_bce" and "shut" flags may be examined to determine the mode of BCE entry. The "ssenb" flag may also be tested to see if the storage system has been enabled.

Auto.ec

This exec_com starts automatic operation.

```
&command_line off
&-
&- automatic reboot ec for bce
&-
&if [equal [bce_state] "early"] &then &goto cant_boot_early
&if [equal [bce_state] "crash"] &then &goto cant_boot_crash
&print Begin auto boot.
set_flagbox bce_command ""
set_flagbox auto_reboot true
set_flagbox booting true
&input_line off
&attach
config_edit
gp/^cpu/
gp/^mem/
q
&detach
set_flagbox bce_command "exec_com rtb"
boot &rf1
&quit
&label cant_boot_early
&print The system cannot be booted from the "early" state.
&print First use "bce" to get to the "boot" state.
&quit
&label cant_boot_crash
&print The system cannot be booted from the "crash" state.
&print First use "reinitialize" to get to the "boot" state.
&quit
```

Dump.ec

This exec_com performs a standard dump.

```
&command_line off
&-
&- ec so site can set dump defaults
&- Honeywell supplied defaults are built into the dump command.
&-
dump -standard &rf1
&quit
```

Go.ec

This exec_com restarts automatic operation after a manual return to bce.

```
&command_line off
&-
&- restart auto operation after manual bce entry
&-
set_flagbox auto_reboot true
set_flagbox rebooted false
set_flagbox booting false
set_flagobx bce_command "exec_com rtb"
go
&quit
```

Rtb.ec

This exec_com determines what operations to perform upon a return to BCE.

```
&command_line off
&-
&- ec to handle returning to bce
&-
&if [not [get_flagbox call_bce]] &then &goto non_call_entry
&-
&print bce invoked via hphcs_$call_bce.
&-
&if [not [query "Should normal recovery procedures be used?"]]
&then &goto abort_auto_mode
&-
&label non_call_entry
&-
&- look at the state of things
&-
&if [not [get_flagbox ssenb]] &then &goto ss_not_enabled
&-
&- storage system enabled; take a dump and esd
&-
exec_com dump
&-
&if [nequal [severity dump] 0] &then &goto dump_okay
&-
&print Dump failed.
&goto abort_auto_mode
&-
&label dump_okay
&-
emergency_shutdown
&- return from above is back at rtb
&-
&label ss_not_enabled
&-
&- Is everything okay?
```

```
&-
&if [nequal [shutdown_state] 4] &then &goto okay_shutdown
&-
&if [nequal [shutdown_state] 3] &then &print Shutdown with locks set.
&else &print Error during shutdown.
&goto abort_auto_mode
&-
&label okay_shutdown
&-
&- normal shutdown - see if we should reboot
&-
&if [not [get_flagbox unattended]] &then &goto abort_auto_mode
&if [not [get_flagbox auto_reboot]] &then &goto abort_auto_mode
&if [get_flagbox booting] &then &goto system_cant_boot
&-
set_flagbox rebooted true
&-
&-inform a.s. that we are doing an automatic reboot
&-
exec_com auto star
&quit
&-
&label system_cant_boot
&-
&print System crashed during boot.
&-
&label abort_auto_mode
&-
set_flagbox bce_command ""
set_flagbox auto_reboot false
set_flagbox rebooted false
&quit
```

APPENDIX F

SAMPLE SYSTEM STARTUP

```
&  ********************************************************
&  *                                                      *
&  * Copyright, (C) Honeywell Information Systems, Inc.,*
&  * 1984                                                 *
&  *                                                      *
&  * Copyright (c) 1972 by Massachusetts Institute of     *
&  * Technology and Honeywell Information Systems, Inc.   *
&  *                                                      *
&  ********************************************************
&
& SYSTEM_START_UP.EC - Installation-dependent commands at system startup time.
&
& This exec_com is invoked by system_control_ three times:
& 1. Before answering service startup, in response to "startup" or
&    "multics" command.
& 2. After answering service startup, in response to "startup" or "go" command.
& 3. After channel attachment, in response to "startup" or "go" command.
&
& a ten-second pause is made between step 2 and channel attachment to allow the
& message coordinator to get output from "login" commands and such out.
&
& To log in a Data Management daemon, issue the command line:
&   login Data_Management Daemon <message_coordinator_channel_id>
& ------------------------------------------------------------------------------
&
&command_line off
&goto &1
&
&label part1
&  must create and set acls for ".message" segments used by non-SysDaemon daemons:
&
&if [exists segment mc.message] &then &else create mc.message; set_acl mc.message
&   rw *.Daemon.*
&if [exists segment reader.message] &then &else create reader.message; set_acl
&   reader.message rw *.Daemon.*
&if [exists segment vinc.message] &then &else create vinc.message; set_acl
&   vinc.message rw *.Daemon.*
&if [exists segment vcons.message] &then &else create vcons.message; set_acl
&   vcons.message rw *.Daemon.*
&if [exists segment vcomp.message] &then &else create vcomp.message; set_acl
&   vcomp.message rw *.Daemon.*
&quit
&
&label part2
&
```

```
&         a.h000 is an example of an installation-dependent channel number
&         of a terminal in an input/output area remote from the main computer
&         room.  The lines referring to a.h000 and ioc2d are commented out,
&         and are present to show how a remote i/o terminal can be set up
&         using the message coordinator
&
&         EXAMPLE OF REMOTE I/O TERMINAL
& sc_command accept a.h000
sc_command define alarm tty otw_
sc_command define scc tty otw_
sc_command define asc tty otw_
sc_command define ioc tty otw_
sc_command define bkc tty otw_
&         EXAMPLE OF REMOTE I/O TERMINAL
& sc_command define ioc2d tty a.h000
&
sc_command define iolog log iolog
sc_command reroute as severity1 default_vcons asc
sc_command reroute as severity2 default_vcons *asc
sc_command reroute as severity3 default_vcons *asc
sc_command route as severity3 *alarm
sc_command route (io1 io2 cord prta prtb) user_i/o ioc
&         EXAMPLE OF REMOTE I/O TERMINAL
& sc_command route (io1 prtb cord) user_i/o ioc2d
sc_command route (io1 io2 cord prta prtb) error_i/o *ioc
&         EXAMPLE OF REMOTE I/O TERMINAL
& sc_command route (io1 prtb cord) error_i/o *ioc2d
sc_command route (prta prtb reader io1) log_i/o iolog
sc_command route (prta prtb reader io1) log_i/o ioc
&         EXAMPLE OF REMOTE I/O TERMINAL
& sc_command route (prtb io1) log_i/o ioc2d
sc_command route (bk cd1 cd2 rt vinc vcons vcomp) user_i/o bkc
sc_command route (bk cd1 cd2 rt vinc vcons vcomp) error_i/o *bkc
&
& CHANGE and uncomment the following line to name the volumes that should
&   be used for process directories.
& sc_command set_pdir_volumes public
sc_command login IO.SysDaemon cord
sc_command login Backup.SysDaemon bk
sc_command login IO.SysDaemon prta
sc_command login Utility.SysDaemon ut
sc_command login Volume_Dumper.Daemon vinc
&
& if system rebooted itself after a crash, while unattended (flagbox 5 is "unattended")
&if [and [get_flagbox 5] [get_flagbox rebooted]] &then &else &goto not_unattended_reboot
& delete the tape drives
sc_command reconfigure delete device  tape_(01 02 03 04 05 06 07 08)
& turn off automatic rebooting, to avoid a crash loop
set_flagbox auto_reboot false
&label not_unattended_reboot
&
&quit
&
&label part3
& set_timax 1
```

```
initialize_peek_limits >system_library_1>ring_zero_meter_limits_ASCII_
set_flagbox booting false
hpsa >system_library_1>system_privilege_ re *.Daemon.*
hpsa >system_library_1>rcp_priv_ re *.HFED.*
hpsa >system_library_1>phcs_ re *.HFED.*
hpsa >system_library_1>tandd_ re *.HFED.*
set_acl >sl1>syserr_log.** [list_iacl_seg >sc1>syserr_log]
save_history_registers off -priv
& The following will log in a daemon to scavenge all mounted physical
& volumes with inconsistencies.
ec admin scav -all -auto -nopt
&quit
&
&label &1
&print ERROR &1
& end
```

APPENDIX G

VOLUME MANAGEMENT

Volumes (e.g., tape reels and disk packs) at a Multics site are usually stored in racks that have a specific location for each volume. These locations, or slots, are identified by a slot name, which is usually identical to (or derived from) the name of the resource that occupies the slot.

When a user requests the mounting of a volume he supplies its name, which can then be used to determine the slot. Operators are responsible for locating the correct volume and mounting it for the user; and for returning volumes to their proper slots when they are no longer needed.

It is necessary that the user's access to the requested volume be verified before he is allowed to mount it. At sites running with RCP Resource Management, the system performs all the necessary access checking before allowing the mount request to appear on the bootload console. In this case, no additional manual access checking need be performed. If the site is not using Resource Management, manual access checking must be performed as determined by site policy. In this case, there is usually a sticker on the volume telling who may mount it.

If Resource Management is not being used, operators must make sure to restrict access to system volumes containing confidential information, such as Backup tapes and disk packs containing saves. These should never be mounted for a nonsystem user, since this will bypass Multics security.

When a mount request is made, the system checks the label recorded on the volume to ensure that the correct volume has been mounted. Almost all volumes contain a valid label in some format that can be recognized by the Multics label authentication software. Occasionally, a volume may be requested that was created at a different site where it had a different name, or a blank or specially-formatted volume is used; and the system either cannot determine the label, or finds a mismatch. In this case, the operator is asked to authenticate the mount request. This is done by issuing the "x auth" command with the authentication code appearing on the volume. The authentication code is a three-letter hash function derived from the volume name.

The make_volume_labels command (described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64) is used to produce stickers for volumes, giving the volume name and authentication code.

APPENDIX H

ALTERNATE PROCEDURES FOR DISK VOLUME RECOVERY

Section 10 discusses different kinds of disk failures and how to recover from them. In its "Disk Volume Recovery Procedures" subsection, it recommends the use of volume reloading. This appendix describes a variation of volume reloading: a BCE restore operation followed by a volume reload operation. This procedure is almost never needed, and for that reason, its description has been placed in this appendix, rather than in Section 10.

This appendix also discusses an alternate procedure for complete disk volume recovery: a BCE restore operation followed by a hierarchy reload operation. While BCE restore/hierarchy reloading is not generally recommended for reloading complete volumes, your site may decide to use this procedure if problems are encountered (e.g., many unreadable tapes) during the volume reloading procedures described in Section 10, or if your site does not use the Volume Backup facility.

DISK VOLUME RECOVERY VIA BCE RESTORE/VOLUME RELOADING

Recovery via BCE restore followed by volume reloading involves replacing the damaged disk volume with a spare volume, restoring the most recent BCE save tapes for the damaged volume using the BCE restore command, and then reloading the consolidated and incremental volume dumper tapes created after the BCE save operation was performed. The -save control argument of the reload_volume command indicates that the date-contents-modified field of each entry being reloaded should be compared with the date-unmounted field of the volume label. Since a volume must be unmounted before a BCE save operation can be performed, the date-unmounted value placed in the volume label by the BCE restore operation is a good indicator of the date on which the BCE save operation was performed. If the entry from the volume backup tape is newer than the date-unmounted field from the disk label, then the tape entry is reloaded.

Recovery of the RPV with BCE Restore/Volume Reloading

If a disk volume failure occurs for the RPV, the following procedure can be used to recover the contents of the RPV from a combination of BCE save tapes and volume backup tapes. See Section 9 for general information and more details on volume backup and volume reloading. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described in Section 10 under "Recovering From Disk Failures". If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering From Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the original RPV volume, or to recover its data onto a spare disk volume, you * will need to boot BCE and Multics on a temporary RPV. This temporary RPV may be obtained in any of the following ways:

- If your site has prepared a one- or two-volume "test system" for hardware and software checkout purposes, you can boot this test system for use in testing and reloading the original RPV

- You can restore the BCE save tapes for the original RPV onto a spare disk volume for use as the temporary RPV. The actual data on the temporary RPV is not important since it will not become part of the production hierarchy; an older set of save tapes can be used, as long as the saved RPV is for the Multics release you are currently running.

You will have to boot BCE on the temporary RPV, and specify "cold" to the "Enter rpv data:" prompt to allow the temporary RPV to be properly initialized. After restoring the RPV, remember to update the root and part configuration cards to describe only the temporary RPV.

Spare disk volumes should be properly formatted and tested as described in Section 10 under "Preformatted Disk Volumes."

3. Boot BCE on the temporary RPV, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original RPV disk volume, attempt to read it using the BCE test_disk command, as described in Section 10 under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original RPV, follow the procedures described in Section 10 under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original RPV is only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described in Section 10 under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload RPV information from BCE save and volume backup tapes onto a spare disk volume. These steps assume that the original RPV volume is totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that the original RPV is physically damaged (i.e., scratched or warped), then replace the RPV with a spare volume which has already been formatted and tested, as described in Section 10 under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original RPV.

7. Mount the disk volume to be reloaded on any available drive.

8. Create a restore control file that will identify the new RPV, then use the BCE restore command to load information from the BCE save tapes onto the new RPV. For example:

```
qx
a
td tapa_01
td tapa_02
ts ROOT
pv rpv dska_01
part rpv dska_01 -all
\f
w rpv_restore
q
restore rpv_restore
```

9. Once the BCE save tapes have been restored, boot Multics on the temporary RPV, coming up to Multics ring 1 command level, as described in the *Operator's Guide to Multics*, Order No. GB61.

10. Convert the disk drive on which the new RPV is mounted to an I/O drive, using the set_drive_usage command. For example:

```
sdu dska_04 io
```

11. Recover the volume log for the RPV using the recover_volume_log command with the -wd control argument. For example:

```
recover_volume_log rpv -wd
```

Mount the last volume backup tape for the volume backup group which includes the RPV. The volume name of the last tape should be recorded in the tape log, as described in Section 10 under "Backup Tape Logs." If volume backup operations were ongoing at the time of disk failure, you should mount the tape which was being written at the time of failure.

12. Reload the new RPV using the volume reloader, by issuing the reload_volume command with the -pvname, -operator, -save, and -wd control arguments. For example:

```
reload_volume -pvname rpv -operator Jones -wd -save
```

Mount tapes as requested by the reload_volume command. When all tapes have been reloaded, continue with the next step.

13. Shutdown Multics on the temporary RPV.

14. If the RPV was reloaded onto a spare volume and the original RPV is partially readable, you may want to try to copy the contents of the CONF, FILE, DUMP, and LOG partitions onto the new RPV, as described in Section 10 under "Recovery of Partitions after RLV Volume Recovery."

15. If the newly reloaded RPV is not mounted on the proper disk drive for normal operation, move the new RPV to the proper disk drive.

16. Boot BCE on the newly reloaded RPV, according to normal site procedures. If reloading was performed on a spare disk volume rather than on the original RPV, then the contents of the CONF, BCE, and FILE partitions have been lost. In BCE, you will have to reload the config deck from a config file read off the BCE tape, using the BCE "config <deckname>" command. Make adjustments to the configuration file as necessary, to reflect the current hardware configuration and disk volume locations.

17. Boot Multics according to normal site procedures.

18. Perform the procedures for salvaging, quota adjustment, and connection failure detection described in Section 10 under "Disk Volume Post-Recovery Procedures." This completes recovery of the RPV.

Recovery of a Non-RPV Root Volume with BCE Restore/Volume Reloading

If a disk volume failure occurs on a volume which is part of the Root Logical Volume (RLV), but is not the RPV, the following procedure can be used to recover the contents of that volume from BCE save tapes and volume backup tapes. See Section 9 for general information and more details on volume backup and volume reloading. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described in Section 10 under "Recovering From Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering From Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the original root volume, or to recover its data onto a spare disk volume, you will need to boot BCE and Multics on the RPV.

3. Boot BCE on the RPV, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original root disk volume, attempt to read it using the BCE test_disk command, as described in Section 10 under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original root volume, follow the procedures described in Section 10 under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original root volume is only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described in Section 10 under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload root volume information from volume backup tapes onto a spare disk volume. These steps assume that the original root volume is totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that the original root volume is physically damaged (i.e., scratched or warped), then replace it with a spare volume which has already been formatted and tested, as described in Section 10 under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original root volume.

7. Mount the disk volume to be reloaded on any available drive.

8. Create a restore control file that will identify the physical volume, then use the BCE restore command to load information from the BCE save tapes onto the volume. For example:

```
qx
a
td tapa_01
td tapa_02
ts ROOT
pv root2 dska_02
\f
w root2_restore
q
restore root2_restore
```

9. Remove all disk volumes from the root config card, except for the RPV. If any part config cards identify the damaged disk volume, remove those part cards from the config deck.

10. Boot Multics on the RPV, coming up to Multics ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61.

11. Convert the disk drive on which the new root volume is mounted to an I/O drive, using the set_drive_usage command. For example:

```
sdu dska_05 io
```

12. Recover the volume log for the root volume using the recover_volume_log command with the -wd control argument. For example:

```
recover_volume_log root2 -wd
```

Mount the last volume backup tape for the volume backup group which includes the RLV. The volume name of the last tape should be recorded in the tape log, as described in Section 10 under "Backup Tape Logs." If volume backup operations were ongoing at the time of disk failure, you should mount the tape which was being written at the time of failure.

13. Reload the new root volume using the volume reloader by issuing the reload_volume command with the -pvname, -operator, -wd and -save control arguments. For example:

```
reload_volume -pvname root2 -operator Jones -wd -save
```

Mount tapes as requested by the reload_volume command. When all tapes have been reloaded, continue with the next step.

14. Shutdown the Multics running on the RPV.

15. Restore the root and part config cards to their normal values, either by retyping the changed cards or by issuing the BCE "config <deckname>" command to load a new copy of the config deck from a BCE file.

* 16. If the root volume was reloaded onto a spare volume and the original volume is partially readable, you may want to try to copy the contents of the DUMP partition onto the new root volume, if this partition was on the damaged root volume. Follow the procedure described in Section 10 under "Recovery of Partitions after RLV Volume Recovery." This can only be done if the location of partitions was not changed on the new root.

17. If the newly reloaded root volume is not mounted on the proper disk drive for normal operation, move the volume to the proper disk drive.

* 18. Boot BCE on the RPV, according to normal site procedures. Make adjustments to the configuration file as necessary, to reflect the current hardware configuration and disk volume locations.

19. Boot Multics according to normal site procedures.

20. Perform the procedures for salvaging, quota adjustment, and connection failure detection described in Section 10 under "Disk Volume Post-Recovery Procedures." This completes recovery of the root volume.

Recovery of a Non-Root Volume with BCE Restore/Volume Reloading

If a disk volume failure occurs on a volume which is not part of the Root Logical Volume (RLV), the following procedure can be used to recover the contents of that volume from BCE save and volume backup tapes. See Section 9 for general information and more details on volume backup and volume reloading. All of the commands used in this procedure are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described in Section 10 under "Recovering From Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering From Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the original volume, or to recover its data onto a spare disk volume, you will need to boot BCE and Multics on the RLV. *

3. Boot BCE, as described in the *Operators' Guide to Multics,* Order No. GB61. |

4. If your Customer Service Representative believes there has been no physical damage to the original disk volume, attempt to read it using the BCE test_disk command, as described in Section 10 under "Extent of Disk Volume Failure." |

5. If only transient errors are encountered when reading the original volume, follow the procedures described in Section 10 under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original volume is only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described in Section 10 under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload information from volume backup tapes onto a spare disk volume. These steps assume that the original volume is totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that the original volume is physically damaged (i.e., scratched or warped), then replace it with a spare volume which has already been formatted and tested, as described in Section 10 under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original disk volume.

7. Mount the disk volume to be reloaded on any available drive.

8. Create a restore control file that will identify the physical volume, then use the BCE restore command to load information from the BCE save tapes onto the volume. For example: |

```
qx
a
td tapa_01
td tapa_02
ts Xpublic
pv xpub02 dska_06
\f
w xpub_restore
q
restore xpub_restore
```

9. Boot Multics on the RLV, coming up to Multics ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61. *

10. To complete the boot, delete the logical volume which contains the damaged physical volume, using the del_lv command. For example:

```
del_lv Xpublic
```

11. Issue the standard command to move to ring 4:

```
standard
```

12. If the system can run reasonably without the deleted logical volume, warn users (via a message_of_the_day, or with a login warning set by the word command) that the logical volume has been deleted for repair operations. For example:

```
word login Xpublic volume is offline for repairs.
```

If the system cannot run reasonably without the deleted logical volume, put the system into a special session, using the multics and go commands. This will prevent users from logging in:

```
multics
go
```

13. Convert the disk drive on which the new volume is mounted to an I/O drive, using the set_drive_usage command. For example:

```
sdu dska_06 io
```

14. Login the volume reloader and issue a reload_volume command with the -operator, -pvname, and -save control arguments. For example:

```
login Volume_Reloader.Daemon vrld
r vrld reload_volume -pvname xpub02 -operator Jones -save
```

Mount tapes as the reloader asks for them; it will indicate when all necessary tapes have been reloaded.

If the reloader indicates that the volume log is unavailable, recover the volume log for the volume using the recover_volume_log command. For example:

```
r vrld recover_volume_log xpub02
```

Mount the last volume backup tape for the volume backup group which includes the failing volume. The volume name of the last tape should be recorded in the tape log, as described in Section 10 under "Backup Tape Logs." If volume backup operations were ongoing at the time of disk failure, you should mount the tape which was being written at the time of failure. After the volume log has been recovered, then reissue the reload_volume command, as shown above.

15. After volume reloading is complete, issue a set_drive_usage command to convert the drive back into storage system usage. For example:

```
sdu dska_06 ss
```

16. Issue the add_vol command to inform the system of the new location for the reloaded disk volume. For example:

```
add_vol xpub02 dska_06
```

17. Issue the add_lv command to add the logical volume containing the reloaded disk volume. For example:

```
add_lv Xpublic
```

18. If the system is in special session, return it to normal session:

```
word login
maxu auto
abs start
abs maxu auto
```

19. Perform the procedures for salvaging, quota adjustment, and connection failure detection described in Section 10 under "Disk Volume Post-Recovery Procedures." This completes recovery of the volume.

DISK VOLUME RECOVERY VIA BCE RESTORE/HIERARCHY RELOADING

The BCE restore/hierarchy reloading strategy can be used to reload a volume which is not part of the Root Logical Volume (single volume reload), to reload the entire Root Logical Volume (RLV reload), or to reload the entire hierarchy (complete reload). BCE restore/hierarchy reloading cannot be used to recover only a single root volume (either the RPV or an RLV volume). A complete or RLV reload must be performed to recover single RLV volumes.

The BCE restore/hierarchy reload strategy involves replacing physically damaged volumes with spare disk volumes, initializing these volumes, and then reloading complete, consolidated and incremental dump tapes onto them in chronological order (the order in which they were written).

Hierarchy Reload of RLV versus Reload of All Volumes

The loss of a part of the Root Logical Volume (RLV) is always very serious. The recovery operation when reloading hierarchy dump tapes is more complex than when reloading volume dump tapes. When reloading hierarchy dump tapes, the entire RLV must be reloaded rather than just the damaged root volume.

The need to reload the entire RLV stems from the way the hierarchy reloader works. If a directory being reloaded does not already exist, the hierarchy reloader uses the next available VTOCE to hold the directory, rather than placing the directory in the same VTOCE from which it was dumped. Because directories are being reloaded into different locations, superior directories can lose track of the new location, causing connection failures. The only method of avoiding such connection failures is to reload the entire RLV.

Another factor adding to the complexity of single volume and RLV hierarchy reloads is the requirement of the hierarchy reloader that it operate on a consistent copy of the hierarchy. After a BCE restore of one or several volumes is complete, directory salvaging and physical volume connection failure detection operations must be performed to restore the consistency of the hierarchy before the hierarchy reload is performed. Directory salvage operations are needed to delete branches for entries which were deleted after the BCE save tapes were made. Reverse connection failure detection is needed to recover VTOCEs for segments which were deleted after the BCE save tapes were made (either by adopting these segments or by garbage collecting their VTOCEs). The considerable amount of time required to perform these operations must be weighed against the simpler, but sometimes longer procedure of doing a complete reload of the entire system.

Recovery of All Volumes with BCE Restore/Hierarchy Reloading

If a disk volume failure occurs on several different disk volumes (either on volumes of the RLV or on non-root volumes), the following procedure can be used to recover the contents of all volumes on the system from BCE save and hierarchy backup tapes. This procedure is often referred to as a "complete restore/reload" of the hierarchy.

Note that it is possible to recover just the volumes of the RLV, or just a single non-root volume. Procedures for such recovery operations are described later in this appendix under "Recovery of the Root Logical Volume with BCE Restore/Hierarchy Reloading" and "Recovery of a Non-Root Volume with BCE Restore/Hierarchy Reloading." However, these recovery operations are more complex than a complete restore/reload operation, and they may be more time-consuming as well. You should consider the steps involved in each type of BCE restore/hierarchy reloading procedure carefully, and choose the best procedure for your particular circumstances.

See Section 9 for general information and more details on hierarchy backup and hierarchy reloading. All of the commands used the procedure below are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described in Section 10 under "Recovering From Disk Failures". If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering From Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the damaged disk volumes, or to recover their data onto spare disk volumes, you will need to boot BCE and Multics on an RPV. The RPV to be used for testing can be obtained in any of the following ways:

- If the RPV of the production Multics system is not one of the damaged disk volumes, you can boot BCE on the original RPV for testing and reloading the other disk volumes

- If your site has prepared a one- or two-volume "test system" for hardware and software checkout purposes, you can boot this test system for use in testing and reloading the original RPV

- You can restore the BCE save tapes for your RPV onto a spare disk volume for use as the temporary RPV. The actual data on the temporary RPV is not important since it will not become part of the production hierarchy; an older set of save tapes can be used, as long as the saved RPV is for the Multics release you are currently running.

You will have to boot BCE on the temporary RPV, and specify "cold" to the "Enter rpv data:" prompt to allow the temporary RPV to be properly initialized. After restoring the RPV, remember to update the root and part configuration cards to describe only the temporary RPV.

3. Boot BCE on the chosen RPV, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original disk volumes, attempt to read them using the BCE test_disk command, as described in Section 10 under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original volumes, follow the procedures described in Section 10 under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original volumes are only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described in Section 10 under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload information from BCE save and hierarchy backup tapes onto spare disk volumes. These steps assume that the original volumes are totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that one or more of the original volumes are physically damaged (i.e., scratched or warped), then they must be replaced with spare volumes which have already been formatted and tested, as described in Section 10 under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original disk volumes.

7. Mount the disk volumes to be reloaded on any available drive. You can use the original disk drives if the Customer Service Representative says they are in good working condition.

8. If the original RPV was physically damaged, then you must reboot BCE on the spare volume which will become the new RPV. The spare disk volume should be properly formatted and tested as described in Section 10 under "Preformatted Disk Volumes." You will have to boot BCE on the temporary RPV, and answer "cold" to the "Enter rpv data:" prompt to specify that the RPV is to be initialized.

   Similarly, if you are running on a temporary RPV or on a test system and the original RPV is not physically damaged, then you must reboot BCE on the original RPV.

9. If the RPV was reloaded onto a spare volume and the original RPV is partially readable, you may want to try to copy the contents of the CONF, FILE, DUMP, and LOG partitions onto the new RPV, as described in Section 10 under "Recovery of Partitions after RLV Volume Recovery."

10. Now you must either create restore control files that will define the volumes to restore, or use the control files that were created when the original BCE save was done. You can restore either one or multiple volume sets. For example:

```
restore -set tape_devs_1 root_lv -set tape_devs_2 public_lv
```

11. Once the BCE save tapes have been restored, boot Multics on the newly reloaded RPV, coming up to ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61.

12. Attach all logical volumes by typing:

```
add_lv -all
```

13. Use the reload command to read, in forward chronological order, all hierarchy consolidated and incremental dump tapes made since the BCE save tapes were created:

```
reload -nomap
```

When all tapes have been reloaded, continue with the next step.

14. Boot Multics according to normal site procedures.

15. Perform the procedures for salvaging, quota adjustment, and connection failure detection described in Section 10 under "Disk Volume Post-Recovery Procedures." This completes recovery of the volume.

Recovery of the Root Logical Volume with BCE Restore/Hierarchy Reloading

If a disk volume failure occurs on one or more disk volumes of the RLV, the following procedure can be used to recover the contents of all volumes of the RLV from BCE save and hierarchy backup tapes. This procedure is often referred to as an "RLV restore/reload". It it sometimes better than a complete restore/reload because it can preserve later copies of non-root segments than those appearing on the backup tapes.

See Section 9 for general information and more details on hierarchy backup and hierarchy reloading. All of the commands used the procedure below are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described in Section 10 under "Recovering From Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering From Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the damaged disk volumes, or to recover their data onto spare disk volumes, you will need to boot BCE and Multics on an RPV. The RPV to be used for testing* can be obtained in any of the following ways:

- If the RPV of the production Multics system is not one of the damaged disk volumes, you can boot BCE on the original RPV for testing and reloading the other disk volumes

- If your site has prepared a one- or two-volume "test system" for hardware and software checkout purposes, you can boot this test system for use in testing and reloading the original RPV

- You can restore the BCE save tapes for your RPV onto a spare disk volume for use as the temporary RPV. The actual data on the temporary RPV is not important since it will not become part of the production hierarchy; an older set of SAVE tapes can be used, as long as the saved RPV is for the Multics release you are currently running.

You will have to boot BCE on the temporary RPV, and specify "cold" to the "Enter rpv data:" prompt to allow the temporary RPV to be properly initialized. After restoring the RPV, remember to update the root and part configuration cards to describe only the temporary RPV.

3. Boot BCE on the chosen RPV, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original disk volumes, attempt to read them using the BCE test_disk command, as described in Section 10 under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original volumes, follow the procedures described in Section 10 under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original volumes are only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described in Section 10 under "Recovering from Partial Disk Volume Failure," and skip the rest of these steps.

The steps below attempt to reload information from BCE save and hierarchy backup tapes onto spare disk volumes. These steps assume that the original volumes are totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that one or more of the original volumes are physically damaged (i.e., scratched or warped), then they must be replaced with spare volumes which have already been formatted and tested, as described in Section 10 under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original disk volumes.

7. Mount the disk volumes to be reloaded on any available drive. You can use the original disk drives if the Customer Service Representative says they are in good working condition.

8. If the original RPV was physically damaged, then you must reboot BCE on the spare volume which will become the new RPV. The spare disk volume should be properly formatted and tested as described in Section 10 under "Preformatted Disk Volumes." You will have to boot BCE on the temporary RPV, and specify "cold" to the "Enter rpv data:" prompt to specify that the RPV is to be initialized.

   Similarly, if you are running on a temporary RPV or on a test system and the original RPV is not physically damaged, then you must reboot BCE on the original RPV.

9. If the RPV was reloaded onto a spare volume and the original RPV is partially readable, you may want to try to copy the contents of the CONF, FILE, DUMP, and LOG partitions onto the new RPV, as described in Section 10 under "Recovery of Partitions after RLV Volume Recovery."

10. Now you must either create restore control files that will define the volumes to restore, or use the control files that were created when the original BCE save was done. For example:

```
restore -set tape_devs_1 root_lv
```

11. Once the BCE save tapes have been restored, boot Multics on the newly reloaded RPV, coming up to ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61.

12. Attach all logical volumes by typing:

```
add_lv -all
```

13. Salvage the Multics hierarchy by typing:

```
salvage_dirs -check_vtoce -delete_connection_failure
```

    to delete directory branches for entries that were present when the BCE save was performed, but have since been deleted.

14. At this point, you must decide whether or not to try performing segment adoption (to create new directory branches to preserve the VTOCEs and segment contents for segments created since the BCE save tapes were written). If you're going to attempt segment adoption, you must do it now, before copies of segments created since the BCE save get reloaded from the backup tapes.

    You must also decide whether there is enough space on non-root volumes to receive copies of segments created since the BCE save tapes were written. If any non-root logical volumes do not have sufficient space to hold new copies of all segments created since the save, you will have to make space on these logical volumes. This can be done by "garbage collection:" looking for reverse connection failures (VTOCEs that have no directory branch), and deleting these VTOCEs.

    If you decide to perform either of these functions, continue with step 15. Otherwise, continue with step 19.

15. Issue the standard command to move to ring 4:

```
standard
```

16. Enter admin mode, using the admin command.

17. Use the sweep_pv command as described in Section 12 under "Segment Adoption" and "How to Perform VTOC Garbage Collection on a Pack."

18. After performing either of these functions, you must leave admin mode, shutdown Multics (to BCE level), reboot Multics to ring 1 command level, and add all logical volumes:

```
ame
shut
boot
add_lv -all
```

19. Use the reload command to read, in forward chronological order, all hierarchy consolidated and incremental dump tapes made since the BCE save tapes were created:

```
reload -nomap
```

If reload error files get created, stop the reload process (at the end of a tape). Cross out to ring 4 and enter admin mode:

```
standard
admin
```

Print the error files. If the errors are occurring because one or more logical volumes are full, you must perform VTOC garbage collection via sweep_pv, as described in Section 12. Then you must leave admin mode, shutdown Multics (to BCE level), reboot Multics to ring 1 command level, and add all logical volumes:

```
ame
shut
boot
add_lv -all
```

Finally, you must start the reload process again with the first tape for which an error file was created.

20. When all tapes have been reloaded, shutdown Multics:

```
shut
```

21. Boot Multics according to normal site procedures.

22. Perform the procedures for salvaging, quota adjustment, and connection failure detection described in Section 10 under "Disk Volume Post-Recovery Procedures." This completes recovery of the volume.

Recovery of a Non-Root Volume with BCE Restore/Hierarchy Reloading

If a disk volume failure occurs on one or more non-root disk volumes, the following procedure can be used to recover the contents of the damaged volumes from BCE save and hierarchy backup tapes. This procedure is often referred to as a "single volume restore/reload".

See Section 9 for general information and more details on hierarchy backup and hierarchy reloading. All of the commands used the procedure below are described in the *Multics Administration, Maintenance and Operations Commands* manual, Order No. GB64.

1. If the system has not already crashed, attempt to recover from the failure by following the procedures described in Section 10 under "Recovering From Disk Failures." If that corrects the problem, then skip the remaining steps. Otherwise, use the last procedure under "Recovering From Disk Failures" to shut down or crash the system.

2. Consult with your Customer Service Representative to correct any hardware failure that is occurring. Have him repair or replace any damaged hardware.

To test the damaged disk volumes, or to recover their data onto spare disk volumes, you will need to boot BCE and Multics on an RPV. The RPV to be used for testing can be the RPV of the production Multics system.

3. Boot BCE, as described in the *Operators' Guide to Multics,* Order No. GB61.

4. If your Customer Service Representative believes there has been no physical damage to the original disk volumes, attempt to read them using the BCE test_disk command, as described in Section 10 under "Extent of Disk Volume Failure."

5. If only transient errors are encountered when reading the original volumes, follow the procedures described in Section 10 under "Recovering from Transient Disk Volume Failure," and skip the rest of these steps.

6. If the original volumes are only partially damaged and you decide that loss of the unreadable records is acceptable, follow the procedures described in Section 10 under "Recovering from Partial Disk Volume Failures," and skip the rest of these steps.

The steps below attempt to reload information from BCE save and hierarchy backup tapes onto spare disk volumes. These steps assume that the original volumes are totally unreadable, or that the amount of lost data caused by unreadable records is unacceptably high. If your Customer Service Representative believes that one or more of the original volumes are physically damaged (i.e., scratched or warped), then they must be replaced with spare volumes which have already been formatted and tested, as described in Section 10 under "Preformatted Disk Volumes." Otherwise, you can reload data onto the original disk volumes.

7. Mount the disk volumes to be reloaded on any available drive. You can use the original disk drives if the Customer Service Representative says they are in good working condition.

8. Now you must either create restore control files that will define the volumes to restore, or use the control files that were created when the original BCE save was done. You can restore either one or multiple volume sets. For example:

```
restore -set tape_devs_1 public_lv -set tape_devs_2 xpublic_lv
```

9. Once the BCE save tapes have been restored, boot Multics on the newly reloaded RPV, coming up to ring 1 command level, as described in the *Operators' Guide to Multics,* Order No. GB61.

10. Attach all logical volumes by typing:

```
add_lv -all
```

11. Issue the standard command to move to ring 4:

```
standard
```

12. Enter admin mode, using the admin command.

13. Perform "garbage collection" on the volumes being reloaded, looking for reverse connection failures (VTOCEs that have no directory branch), and deleting these VTOCEs. Such segments have been moved or deleted since the BCE save tapes were written. Use the sweep_pv command as described in Section 12 under "How to Perform VTOC Garbage Collection on a Pack."

14. Leave admin mode, shutdown Multics (to BCE level), reboot Multics to ring 1 command level, and add all logical volumes:

```
ame
shut
boot
add_lv -all
```

15. Use the reload command to read, in forward chronological order, all hierarchy consolidated and incremental dump tapes made since the BCE save tapes were created:

```
reload -nomap -error_on
```

Do not use the -pvname control argument. The reload command will only reload segments from the tape whose date-contents-modified is later than that of the existing segment on disk, or for which there is no existing disk segment.

If reload error files get created, stop the reload process (at the end of a tape). Cross out to ring 4 and enter admin mode:

```
standard
admin
```

Print the error files. If the errors are occurring because one or more logical volumes are full, you must perform VTOC garbage collection via sweep_pv, as described in Section 12. Then you must leave admin mode, shutdown Multics (to BCE level), reboot Multics to ring 1 command level, and add all logical volumes:

```
ame
shut
boot
add_lv -all
```

Finally, you must start the reload process again with the first tape for which an error file was created.

16. When all tapes have been reloaded, shutdown Multics:

```
shut
```

17. Boot Multics according to normal site procedures.

18. Perform the procedures for salvaging, quota adjustment, and connection failure detection described in Section 10 under "Disk Volume Post-Recovery Procedures." This completes recovery of the volumes.

APPENDIX I

MULTICS HEALS

DESCRIPTION OF HEALS

HEALS (Honeywell Error Analysis and Logging System) assists Customer Services and operations personnel in monitoring the performance of the hardware. It provides a record of hardware operation for diagnosing transient malfunctions, tracking performance of hardware modules, and predicting scheduled maintenance.

HEALS IMPLEMENTATION

The functions of an error analysis and logging system are:

1. Capturing and logging hardware data.

2. Sorting and analyzing the data.

3. Presenting the analyzed data in a series of reports.

The logging function is performed by the syserr mechanism to the syserr log. The other functions are performed by the facilities described in this appendix.

The syserr log contains a number of entries not needed for the HEALS reports, and the time interval of syserr log data is normally not as large as may be desired for HEALS error data analysis. Therefore, the syserr log entries of interest to HEALS are extracted from the syserr log and written to an independent segment named >system_control_1>heals_dir>heals_log (hereafter referred to as the HEALS log).

The update_heals_log, truncate_heals_log, and print_heals_message commands are provided to manage the HEALS log.

The heals_report command creates a report for the specified time intervals and appends it to the output file, which is created if none exists. The default pathname of the output file is heals_reports in the working directory. The HEALS log is not updated or otherwise changed by the heals_report command. If the latest syserr log entries are wanted in the reports, the heals_report command must be preceded by the update_heals_log command.

The segment heals_log and a control data segment (heals_log_info) are contained in the directory >system_control_1>heals_dir. Management of the HEALS log is expected to be done by Customer Services personnel.

HEALS INSTALLATION REQUIREMENTS

The directory >system_control_1>heals_dir, created by acct_start_up.ec (the system accounting startup), must exist.

The heals_log segment is created by the first invocation of the update_heals_log command.

HEALS USAGE

HEALS is for both routine reporting of hardware errors and for specific reports on demand.

All HEALS reports should be generated on a daily basis following a HEALS log update to maintain a continuous record of hardware errors and malfunctions. This HEALS activity should be triggered by a scheduled absentee process such as the administrative "crank."

Any time that specific reports are wanted for monitoring or diagnostic purposes, the heals_report command can be invoked at the terminal with the name of the specific report desired (e.g., heals_report io_error). Similarly, update_heals_log can be invoked by a privileged user of HEALS.

HEALS REPORTS

HEALS reports are initiated by the heals_report command (described later in this section). The names of desired reports, the time period of the reports, and the pathname of the report file are specified by arguments to the command. The reports are:

io_error_report
: contains all I/O errors logged in the syserr log by the ioi_, disk_control, dn355, and bulk_store_control subroutines. The entries are in syserr log time sequence and contain the full octal status return word.

sorted_io_error_report
: contains the I/O errors of the io_error report ordered by day and by device address (IOM number, channel number, and device number). The errors are grouped for the convenience of maintenance personnel. Within a device address, entries are further ordered by power off, major status, sub status, initiate/terminate interrupt, device command, IOM status, and record count residue. The octal status word is replaced (to keep the format width to 72 columns) by additional details of tape and disk errors.

cpu_error_report
contains history register data and other pertinent data for op_not_complete, parity, command, startup, and shutdown faults.

mos_edac_error_report
contains the MOS EDAC error entries in syserr log.

media_io_error_report
is similar in content to the sorted_io_error report except that the primary sort key is media volume name (e.g., tape reel number).

EXAMPLES OF REPORTS

Examples of the HEALS reports that result from invocation of the heals_report command are shown on the following pages. An exception to this is the media_io_error report, which is not shown because its format and content are similar to the sorted_io_error report. If a problem is detected in processing an entry for the io_error report or the sorted_io_error report, the problem is reported with a comment line in place of data in the report entry. If the system is reconfigured between the time an error is logged and the time a HEALS run is executed, reassigned channels or device names different from those obtained from the configuration table are not known to the report generators. These are reported as "ch_unkn" or "dv_unkn". The configuration known to HEALS is printed preceding an io_error or sorted_io_error report. If a device address cannot be determined, it is assigned IOM number 0 and channel number 0 so that the entries are grouped at the beginning of the sorted_io_error report. The numbers assigned 0,0 flag the entries as having invalid addresses.

Each entry of a report contains the syserr log sequence number and log time so that entries can be cross-referenced to the original syserr log and the HEALS log, and between the io_error and sorted_io_error reports.

Channel Assignment Table

The configuration known to HEALS that is printed out prior to an io_error or sorted_io_error report is shown below.

CHANNEL ASSIGNMENT TABLE AT TIME OF HEALS RUN
RUN DATE: 08/15/84 RUN TIME: 1620.4,
SYSTEM_ID: MR11.0 SITE_ID: Honeywell

IOM NUM	CHNL NUM	DEVICE NAME	MODEL NUMBER
1	08	prtd	1600
1	09	prta	1200
1	10	rdra	301
1	11	puna	300
1	12	prtc	901
1	14	rdrb	201
1	15	punb	201
1	16	opca	6601
1	17	fnpa	6670
1	18	tape	500
1	24	dska	451
1	25	dska	451
1	26	dska	451
1	27	dska	451
1	28	dskb	451
1	29	dskb	451
1	30	dskb	451
1	31	dskb	451

I/O Error Report

```
IO_ERROR_REPORT: 08/14/84  1619.8 TO 08/15/84  1619.8            PAGE  1

SYSERR     LOG         DEVICE         STATUS  TLY     TAPE_NO  STATUS_RETURN
------------    ---------------   ------
TIME    NUMBER    NAME I-CC-DD CM    MJ-SB-I          DISK_AD

-------------------------------------------------------------------------------
DATE: 08/14/84                                                  DATE: 08/14/84

1725.4   34421    rdra 1-10-01 01    02-01-t      5    N/A         420140000000
1809.8   34427    prtd 1-08-01 34    03-10-t      2    N/A         431000000000
1809.9   34429    prtd 1-08-01 34    02-01-i      1    N/A         420102000000
1822.4   34440    prtd 1-08-01 34    03-04-t      1    N/A         430400000000
1834.6   34441    prtd 1-08-01 34    02-01-i      1    N/A         420102000000
1917.5   34447    tape 1-18-01 15    13-22-t      5    .           532200000000
1926.5   34457    tape 1-18-03 15    13-22-t      1    mc019       532200000000
1939.5   34458    tape 1-18-03 15    13-22-t      1    mc019       532200000000
1955.4   34482    tape 1-18-04 15    13-22-t      1    mc020       532200000000
2000.8   34490    tape 1-18-02 15    13-22-t      1    mc021       532200000000
2006.7   34491    tape 1-18-02 15    13-22-t      1    mc021       532200000000
2012.3   34499    tape 1-18-01 15    13-22-t      1    mc022       532200000000
2017.7   34504    tape 1-18-03 05    12-10-t      1    m2088       521000000000
2017.9   34505    tape 1-18-03 05    12-10-t      1    m2088       521000000000
2018.4   34508    tape 1-18-01 15    13-22-t      2    mc022       532200000000
2023.2   34516    tape 1-18-04 15    13-22-t      1    mc023       532200000000
2034.1   34519    tape 1-18-04 15    13-22-t      7    mc023       532200000000
2045.2   34527    tape 1-18-02 15    13-22-t      1    mc024       532200000000
2047.9   34528    tape 1-18-02 15    13-22-t      1    mc024       532200000000
2053.7   34536    tape 1-18-03 15    13-22-t      1    mc025       532200000000
2103.8   34549    tape 1-18-03 15    13-22-t      3    mc025       532200000000
2116.8   34557    tape 1-18-04 15    13-22-t      1    mc026       532200000000
2120.8   34571    tape 1-18-01 15    13-22-t      1    mb025       532200000000
2208.2   34582    tape 1-18-03 15    03-10-t      1    m2068       431000000000
2357.9   34586    tape 1-18-01 15    13-22-t      2    mb025       532200000000

-------------------------------------------------------------------------------
DATE: 08/15/84                                                  DATE: 08/15/84

0700.3   34610    dska 1-26-02 31    02-20-t      1                422000000100
0700.3   34612    dska 1-26-02 31                      422456
0700.3   34614    dska 1-26-02 31    extended: (40 00 00 00 82 00 00 00 00)
0714.0   34617    tape 1-18-01 15    13-22-t      2    mb026       532200000000
0728.2   34626    tape 1-18-02 15    13-22-t      1    mb027       532200000000

END: IO_ERROR_REPORT
```

Sorted I/O Error Report

```
SORTED_IO_ERROR_REPORT: 08/14/84  1619.8 to 08/15/84  1619.8     PAGE  1

DEVICE  STATUS  TLY   TAPE_NO DENS RING TR<    SYSERR LOG
------  ------  ---   ------  ----
I-CC-DD NAME CM   MJ-SB-I         DISK_AD  CYL HEAD SEC    TIME   NUMBER

-------------------------------------------------------------------------
DATE: 08/14/84                                              DATE: 08/14/84

1-08-01 prtd 34   02-01-i    1    N/A                          1809.9  34429
1-08-01 prtd 34   02-01-i    1    N/A                          1834.6  34441
1-08-01 prtd 34   03-04-t    1    N/A                          1822.4  34440
1-08-01 prtd 34   03-10-t    2    N/A                          1809.8  34427
end: prtd errors

1-10-01 rdra 01   02-01-t    5    N/A                          1725.4  34421
end: rdra errors

1-18-01 tape 15   13-22-t    5    .        .      .   .       1917.5  34447
1-18-01 tape 15   13-22-t    1    mc022    1600   ys  df      2012.3  34499
1-18-01 tape 15   13-22-t    2    mc022    1600   ys  df      2018.4  34508
1-18-01 tape 15   13-22-t    1    mb025    1600   ys  df      2120.8  34571
1-18-01 tape 15   13-22-t    2    mb025    1600   ys  df      2357.9  34586
1-18-02 tape 15   13-22-t    1    mc021    1600   ys  df      2000.8  34490
1-18-02 tape 15   13-22-t    1    mc021    1600   ys  df      2006.7  34491
1-18-02 tape 15   13-22-t    1    mc024    1600   ys  df      2045.2  34527
1-18-02 tape 15   13-22-t    1    mc024    1600   ys  df      2047.9  34528
1-18-03 tape 15   03-10-t    1    m2068    800    ys  df      2208.2  34582
1-18-03 tape 05   12-10-t    1    m2088    800    ys   9      2017.7  34504
1-18-03 tape 05   12-10-t    1    m2088    800    ys   9      2017.9  34505
1-18-03 tape 15   13-22-t    1    mc019    1600   ys  df      1926.5  34457
1-18-03 tape 15   13-22-t    1    mc019    1600   ys  df      1939.5  34458
1-18-03 tape 15   13-22-t    1    mc025    1600   ys  df      2053.7  34536
1-18-03 tape 15   13-22-t    3    mc025    1600   ys  df      2103.8  34549
1-18-04 tape 15   13-22-t    1    mc020    1600   ys  df      1955.4  34482
1-18-04 tape 15   13-22-t    1    mc023    1600   ys  df      2023.2  34516
1-18-04 tape 15   13-22-t    7    mc023    1600   ys  df      2034.1  34519
1-18-04 tape 15   13-22-t    1    mc026    1600   ys  df      2116.8  34557
end: tape errors


-------------------------------------------------------------------------
DATE: 08/15/84                                              DATE: 08/15/84

1-18-01 tape 15   13-22-t    2    mb026    1600   ys  df      0714.0  34617
1-18-02 tape 15   13-22-t    1    mb027    1600   ys  df      0728.2  34626
end: tape errors

1-26-02 dska 31   02-20-t    1                                0700.3  34610
1-26-02 dska 31                  422456  555    16  16       0700.3  34612
1-26-02 dska 31   extended: (40 00 00 00 82 00 00 00 00)      0700.3  34614
```

SORTED_IO_ERROR_REPORT (cont)

DEVICE	STATUS	TLY	TAPE_NO	DENS	RING	TRK	SYSERR	LOG
I-CC-DD NAME CM	MJ-SB-I		DISK_AD	CYL	HEAD	SEC	TIME	NUMBER
DATE: 08/13/84							DATE: 08/13/84	
1-17-07 tapa 00	02-04-i	1					0902.7	34690
1-17-07 tapa 00	02-04-i	1					1013.6	34929
1-17-07 tapa 00	02-04-i	1					1046.3	35044
1-16-05 tapa 00	03-10-t	6	.	.	.	.	0842.3	34641
1-16-05 tapa 00	03-10-t	1	.	.	.	.	0842.6	34643
1-16-05 tapa 00	03-10-t	3	.	.	.	.	0844.6	34645
1-16-05 tapa 00	03-40-t	3	.	.	.	.	0847.7	34647
1-16-05 tapa 00	03-40-t	1	.	.	.	.	0847.8	34650
1-16-05 tapa 00	03-40-t	1	.	.	.	.	0847.8	34652
1-17-05 tapa 00	03-10-t	1	.	.	.	.	0842.3	34642
1-17-05 tapa 00	03-10-t	1	.	.	.	.	0843.5	34644
1-17-05 tapa 00	03-40-t	1	.	.	.	.	0847.7	34649
1-17-05 tapa 00	03-40-t	1	.	.	.	.	0847.8	34651
1-26-01 dska 00	00-03-t	1	0001496	1	18	16	1126.7	35133
1-26-01 dska 00	00-03-t	1	0001507	1	18	27	1129.3	35145
1-28-11 dskb 34	00-01-t	1	0081464	107	3	24	1332.3	35465
1-26-07 dska 00	00-20-t	1	0120512	158	10	32	0925.9	34736
1-26-07 dska 00	00-02-t	1	0121272	159	10	32	0926.0	34737
1-24-07 dska 35	00-03-t	1	0402248	529	5	08	0954.2	34843
1-26-07 dska 00	00-20-t	1	0404640	532	8	00	0954.1	34841
1-26-07 dska 35	00-20-t	1	0405400	533	8	00	0954.2	34842
1-20-01 dskc 34	00-20-t	1	0444384	584	13	24	1053.1	35075
1-20-01 dskc 00	00-20-t	1	0445144	585	13	24	1053.0	35074
1-28-11 dskb 00	00-20-t	1	0592040	779	0	00	1331.8	35464
1-16-04 tapa 00	03-10-t	1	dp012	dflt	ys	df	1355.3	35516
1-16-04 tapa 00	03-10-t	5	dp012	dflt	ys	df	1358.5	35522
1-16-01 tapa 00	03-40-t	11	dp126	dflt	ys	df	0838.7	34630
1-16-03 tapa 00	03-10-t	1	dp127	dflt	ys	df	0839.6	34640
1-16-03 tapa 00	03-10-t	2	dp127	dflt	ys	df	0849.6	34660
1-16-03 tapa 00	03-40-t	2	dp127	dflt	ys	df	0849.6	34663

END: SORTED_IO_ERROR_REPORT

CPU Error Report

```
CPU_ERROR_REPORT:        from 08/12/84  1081.7       to 08/12/84  1300.0
HEALS RUN OF 08/19/84   1102.0 ON SYSTEM MR11.0


____________CU Legend_____________
cy = cycle type (d = direct operand)
(i=instr..fetch,o=operand,F=fault)
(n=indirect,x=xec,*=nop,e=EIS)
mc = memory command
(00=rrs,sp; 04=rrs,dp; 10=rcl,sp)
(12=rmsk,sp; 16=rmsk,dp; 20=cwr,sp)
(24=cwr,dp; 32=smsk,sp; 36=smsk,dp)
(40=rd/lck; 54=rgr; 56=sgr)
(60=wrt/ulck; 62=con; 66=xec; 72=sxc)
>>>flags<<<
-y = memory address invalid
br = BAR mode
cl = control unit load
cs = control unit store
dr = direct operand
fa = prepare fault address
ic = IC value is odd
in = inhibited instruction
ol = operations unit load
os = operations unit store
pa = prepare operand address
pb = port busy or data from cache
pi = prepare instruction address
pl = port select logic not busy
pn = prepare final indirect address
pt = prepare operand tally
ra = request alter word
ri = request indirect word
rp = executing repeat
sa = store alter word
si = store indirect word
tr = transfer condition met
wi = request instruction fetch
xa = prepare execute interrupt address
xe = execute double from even ICT
xi = execute interrupt present
xo = execute double from odd ICT

____________OU Legend_____________
>>flags<<<
9b = 9-bit byte (IT modifier only)
ar = A-register in use
d1 = first divide cycle
d2 = second divide cycle
dl = direct lower operand
du = direct upper operand
in = first ou cycle
it = IT character modifier
oa = mantissa alignment cycle
oe = exponent compare cycle
of = final OU cycle
om = general OU cycle
on = normalize cycle
os = second cycle of multiple ops
qr = Q-register in use
rb = opcode buffer loaded
rp = primary register loaded
rs = secondary register loaded
sd = store data available
-d = data not available
x0 = index 0 in use
x1 = index 1 in use
x2 = index 2 in use
x3 = index 3 in use
x4 = index 4 in use
x5 = index 5 in use
x6 = index 6 in use
x7 = index 7 in use
```

```
_____DU Legend_____
mc = data mode (b,4,6,9,w)
offset = descriptor counter
>>>flags<<<
()a = prepare alignment count for
      numeric operand (1,2)
a() = load alpha operand (1,2)
al = adjust length
as = alpha store
bd = binary-decimal execution

bg = blanking gate
c0 = force stc0
cg = character operation
d() = descriptor active (1,2,3)
da = data available
db = decimal-binary execution
dd = decimal unit idle
di = decimal unit interrupted
dl = decimal unit load
ds = decimal unit store
ei = mid-instruction interrupt enabled
en = end instruction
es = end sequence
ff = floating result
fl = first data buffer load
fp = first pointer preparation
fs = end sequence
l() = load descriptor (1,2,3)
ld = length = direct
lf = end first pointer preparation
lv = level < word size
lx = length exhaust
l< = length < 128
mp = executing MOPs
n() = load numeric operand (1,2)
nd = need descriptor
ns = numeric store
op = operand available
pc = alpha packing cycle
pl = prepare operand length
pp = prepare operand pointer
r() = load rewrite register (1,2)
re = write-back partial word
rf = rounding
rl = rewrite register 1 loaded
```

```
_____APU Legend_____
seg# = SDWAMR and PTWAMR numbers if
corresponding MATCH bits are set.
offset = final store address
mc = ring number (TSR.TRR)

>>>flags<<<
an = final address, nonpaged
ap = final address, paged
f  = access violation or directed
     fault
fd = fetch descriptor segment PTW
fh = fault waiting
fs = fetch SDW
md = modify descriptor segment PTW
mp = modify PTW
p1 = fetch PTW
p2 = fetch PTW+1
pm = MATCH in PTWAM
sm = MATCH in SDWAM
```

CPU_ERROR_REPORT (cont)

```
rw = du=rd+wt control interlock
sa = select address register
sg = shift procedure
xg = exponent network
xm = extended al,ql modifier
+g = add-subtract execution
*g = multiply-divide execution


syserr sequence #33228, at 08/12/84 1238.7;
syserr_log text: op_not_complete fault on CPU B by
                 Initializer.SysDaemon.z.

scu_data:            000033570041 000000000027 400326000120 000000000000
                     000230000200 342000000005 000006757120 000006757120

pointer registers:   61|5070  61|5120  33|446  61|4720
                     15|1374  15|1374  61|4720  61|0

index registers:     003126 005070 001260 000000
                     000002 000030 000241 000200


a: 000000002000 q: 000446000000  exp: 000 timer: 000331342 ring_alarm: 0

eis_info:            000400000000 000400000000 004620252000 771077777707
                     000000002000 000077777670 004576002004 000077777734

fault register:      010400000000

NUM       OU registers                CU registers
1         627000627100 137767003101   200107764000 000033050020
2         213000213100 123777013505   201037710100 000447050200
3         450000450300 177777013522   201137710000 000224050200
4         736000236340 113777003107   300007235120 005072050020
5         736000736100 133777003110   200007235000 004145042011
6         621000621100 136777003122   600137735000 000226042201
7         431210431100 123777003123   200127735000 000007050015
10        275210275500 127777010221   300007035120 005074050020
11        757000757300 177777010222   200007035000 000050042011
12        740000740300 175777010223   600137735000 000230050201
13        213000213100 123777012172   200127735000 000003050015
14        735000235340 107777000224   300007413120 001464050021
15        735000735100 127777000225   200007413005 002000550010
16        035000035500 127777000226   700137757120 000232044201
17        735000735100 127777000227   300127757120 005076050021
20        413000735240 023777000230   300125757120 005076050002
```

CPU_ERROR_REPORT (cont)

```
NUM       DU registers                          AU registers
1         777757037717 744243410017             000614006144 023321450775
2         737757037737 744243410017             000336012000 001145460775
3         737757037737 744243410017             000336001020 000001470775
4         777757037737 744243410017             000612006144 023331740775
5         737757037717 744243410017             000144006450 005621500775
6         777757037737 744243410017             000336012000 001145500775
7         737757037737 744243410017             000336001020 000001430775
10        777757037737 744243410017             000153000000 001775740775
11        777757037717 744243410017             000152201000 023521720775
12        737757037737 744243410017             000152011000 000403640775
13        777757037737 744243410017             000715000000 001775740775
14        777757037737 744243410017             000714201100 023521620775
15        737757037717 744243410017             040040040040 040040040040
16        777757037737 744243410017             040040040040 040040040040
17        737757037737 744243410017             000224400043 006440000000
20        737757037737 744243410017             077777400043 000001000000

 HR                          c
id##   IC_____ opcd__ tag_ y seg#_ offset__ mc flags

CU 1           lprp4       o             33  4 pa ic -y cl
CU 2           tra         i            447  4 pa tr wi it
CU 3       447 tra         i            224  4 pa tr ic wi
CU 4       224 lda    n*   n           5072  4 pa ri -y it cl
CU 5             "         o           4145  4 pa -y ol pb
OU14                                         rp rs in of ar
CU 6       225 als         i    61      226  4 pi pa ic wi pb
AU 1                         1  2  2332145  0 ap sm pm
CU 7             "         d    33        7  4 pa ic wi -y ol dr pb
OU15                                         rs of -d ar
CU10       226 adla   n*   n    33     5074  4 pa ri -y it cl
CU11             "         o    33       50  4 pa -y ol pb
OU16                                         rb rs of -d ar
CU12       227 als         i    33      230  4 pi pa ic wi pb
AU 2                        0       114546  0 an sm
CU13             "         d    33        3  4 pa ic wi -y ol dr pb
OU17                                         rs of -d ar
CU14       230 rscr   n*   n    33     1464  4 pa ri -y it cl pb
CU15             "    al   o    33     2000 54 pa -y ol
CU16       231 staq   n*   i    33      232  4 pi pa ri ic wi it pb
AU 3                                     147  0
AU 4                         1  2  2333174  0 ap sm pm
CU17             "    n*   n    14     5076  4 pa ri ic wi -y it cl pb
CU20             "    n*   F    14     5076  4 pa ri ic wi -y fa it pl
OU20                                         rp in -d ar qr

END: CPU_ERROR_REPORT
```

MOS EDAC Error Report

```
MOS_EDAC_ERROR_REPORT:      from 08/01/84  1059.5      to 08/07/84  1059.5
HEALS RUN OF 8/19/84  1059.7 ON SYSTEM MR11.0

       LAST ERROR              TALLY     ERROR     SYSTEM CONTROLLER REGISTER
LOG_NUM DATE        TIME                 RATE
                                         /MIN

  21019 08/01/84  1435.7          1      5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  21589 08/02/84  1049.8          1      5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  21649 08/03/84  1709.8          1      5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  22193 08/04/84  1146.8          2      5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  22273 08/04/84  1256.8          2      5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22274 08/04/84  1256.8          1      5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  22428 08/04/84  1441.8          4      5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22549 08/04/84  1646.8          4      5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22661 08/04/84  1951.8          2      5.00      000000000000 140737400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board Q, chip A67

  22730 08/05/84  0001.9          1      5.00      000000000000 542177400001
     EDAC error on mem b store b. MOS, 4k chip, Error: board M, chip A77

  23343 08/05/84  1606.2          1      5.00      000000000000 340077400001
     EDAC error on mem b store a. MOS, 4k chip, Error: board R, chip A78

  23573 08/06/84  0412.3          1      5.00      000000000000 000137400001
     EDAC error on mem c store a. MOS, 4k chip, Error: board Q, chip A17


END: MOS_EDAC_ERROR_REPORT
```

HEALS COMMANDS

The commands that can be invoked to produce the HEALS reports are described in the remainder of this section. Command descriptions are presented in alphabetical order.

Name: heals_report

SYNTAX AS A COMMAND

```
heals_report {report_names} {-control_args}
```

FUNCTION

produces reports of interest to site-support and Customer Services personnel. The reports are appended to a report file specified in the -output_file control argument or by default to the heals_reports segment in the working directory. The ASCII report segment can be displayed on the terminal or printed on a high-speed line printer.

ARGUMENTS

report_names
can be one or more names from the following list (see -all below):

io_error
selects the I/O error report.

sorted_io_error
selects the sorted I/O error report.

media_io_error
selects the media I/O error report.

cpu_error
selects the CPU error report.

mos_edac_error
selects the MOS EDAC error report.

CONTROL ARGUMENTS

-all, -a
specifies that all reports are to be generated. This argument can be used instead of listing all report names.

-from DT, -fm DT
specifies the date and time after which errors are reported. If this argument is not given, the default value is the value of -to time minus 24 hours.

-output_file path, -of path
puts the report file in the file specified by path.

-to DT
specifies the date and time up to which errors are reported. If this argument is not given, the default value is the current date and time.

ACCESS REQUIRED

You must have r access on >system_control_1>heals_dir>heals_log to use this command.

NOTES

The dates specified after the -from and -to control arguments must be acceptable to convert_date_to_binary_ (described in the *Multics Subroutines and I/O Modules* manual, Order No. AG93).

EXAMPLES

If the command line:

```
heals_report io_error -from 03/01/84 -to 03/02/84
```

is issued at 2:00 PM, an ASCII report segment named heals_reports suitable for printing is created in the current working directory, containing the I/O error report for the period from 2:00 PM, March 1, 1984 to 2:00 PM, March 2, 1984.

Name: print_heals_message

SYNTAX AS A COMMAND

```
print_heals_message {-control_args}
```

FUNCTION

is a tool to be used by administrators for the maintenance of the HEALS log (the segment named >system_control_1>heals_dir>heals_log). It allows the printing of all or selected messages currently in the log. It can also be used to delete bad records from the log as well as to print out parts of each logged record.

CONTROL ARGUMENTS

-match STR
selects messages with text containing the match string.

-time DT
selects all messages that occurred after the specified time. If omitted, a value of 0 is assumed.

-update
allows you to delete selected messages from the HEALS log if you have the appropriate access. (See "Notes" below.)

ACCESS REQUIRED

You must have rw access on >system_control_1>heals_dir>heals_log for the update function; otherwise, r access is sufficient.

NOTES

The date/time following the -time control argument must be of a form acceptable to convert_date_to_binary_ (described in the *Multics Subroutines and I/O Modules* manual, Order No. AG93).

The print_heals_message command opens the heals_log segment with a mode of keyed_sequential_update to allow messages to be deleted. If a message is selected by using either the -time or the -match control argument, you can issue the following requests:

quit, q
discontinues message processing and returns to command level.

next
selects the next message that meets the specified selection requirements.

delete
deletes the current record.

data
prints the octal data contained in the current record.

EXAMPLES

The command line:

```
print_heals_message  -time 01/01/84  -match ioi_interrupt
```

sends to the user_output I/O switch all messages that were received after 01/01/84 whose ASCII text contains the string "ioi_interrupt".

Name: truncate_heals_log

SYNTAX AS A COMMAND

```
truncate_heals_log N
    or
truncate_heals_log {-control_args}
```

FUNCTION

deletes records from >system_control_1>heals_dir>heals_log. It is used with the update_heals_log command.

ARGUMENTS

N
is the number of days, counted back from the current time, for which messages are to remain in the HEALS log.

CONTROL ARGUMENTS

-from DT, -fm DT
starts deleting messages from the specified date/time. If this control argument is omitted, a clock value of 0 is assumed; that is, the truncate_heals_log command starts deleting messages from the beginning of the log.

-to DT
stops deleting messages from the specified date/time. If omitted, a clock value equal to the current time is assumed.

ACCESS REQUIRED

You must have rw access to the heals_log and heals_log_info segments, both located in >system_control_1>heals_dir, in order to delete messages from the HEALS log.

NOTES

The date/times following the control arguments must be in a form acceptable to convert_date_to_binary_ (described in the *Multics Subroutines and I/O Modules* manual, Order No. AG93).

update_heals_log

Name: update_heals_log

SYNTAX AS A COMMAND

```
update_heals_log
```

FUNCTION

copies messages of interest to HEALS from the syserr log file into the HEALS log. The messages copied are those new messages added to the syserr log since the last invocation of the update_heals_log command by any process.

ACCESS REQUIRED

In order to update the log, the directory >system_control_1>heals_dir must already exist, and you must have access to system files as follows:

```
re to audit_gate and to phcs_
r  to system_control_1>perm_syserr_log
rw to system_control_1>heals_dir>heals_log
rw to system_control_1>heals_dir>heals_log_info
```

If either the segment >system_control_1>heals_dir>heals_log or the segment >system_control_1>heals_dir>heals_log_info does not exist, it is created; in this case, you need sma access on >system_control_1>heals_dir. The heals_log_info segment contains information about the current heals_log segment.

APPENDIX J

MULTICS DISK MANAGEMENT

This appendix deals generally with Multics system performance, how to monitor it via metering, and how to improve it via tuning. It deals specifically with system performance as it applies to the virtual memory system (physical memory and the disk systems), and even more specifically with performance as it applies to the Multics disk DIM, which is a significant part of the virtual memory system.

The Multics disk DIM (device interface module) controls disk drives. This appendix explains how Multics manages disk systems via the disk DIM. It outlines both the hardware and the software basis for the current disk management methods. It describes the features of the disk DIM, and provides an understanding of the mechanisms of disk control, and the way disk control can affect system operation. Finally, it shows you how to improve system performance, especially in high load situations (i.e., when there's a high volume of disk I/O), in terms of metering and tuning the virtual memory system.

Throughout this appendix, various meters and values are presented in terms of the mechanisms and features which produce them, in an attempt to provide a better understanding of how to improve system performance.

TUNING

Tuning may be defined as the art of determining the inefficiencies in the operation of a Multics system and correcting them. Tuning involves experimentation, theorization, skill, and luck.

A major component of any tuning effort is the ability to set goals which are reasonable and measurable, and to then go through the orderly procedure of substantiating what the problems are and what results are attainable. To pursue a concerted tuning effort without measurement is difficult, if not impossible. To pursue a tuning effort for unrealistic and unattainable goals is a waste of time.

The following questions should be considered as part of any measurement and tuning effort.

1. *Is there a problem?*
   This is a simple question, but one which is usually overlooked. It is not sufficient to simply believe that you are not getting enough out of your system. You need to determine what you are getting and what you are missing. A ballpark figure can be attained by determining what percent of the system is delivered as *virtual CPU time* to the users.

2. *What is the source of the problem?*
Most sites with efficiency problems find that these problems are the result of distinct and correctable situations, many of which are caused by the way in which the system is used. Generally, such problems are solved without tuning those parameters dealing with hardware, but rather by tuning those parameters dealing with parallel system loading.

3. *What are the characteristics of the problem?*
Certain types of problems only become apparent when a system becomes loaded. They are typically manifestations of bottleneck areas of system performance. A number of these areas exist in the virtual memory system of Multics and in its management of disk drives and memory. Recent disk DIM enhancements are aimed specifically at these kinds of situations.

4. *What is the scale of the problem?*
The scale of the problem is an indication of the amount of effort which is reasonable to expend on its solution. In addition, you need to determine future trends for system utilization. If you have 10% of the system left at a reasonable level of delivery and you are expanding 15% a year in use, then simple tuning methods will not provide a long term solution.

5. *What methods can be used to resolve the problem?*
As noted above, many problems can be resolved by changing the parallel loading characteristics of the system, rather than by changing the system's hardware tuning characteristics. Some other problems require altering a combination of tuning both hardware and loading, to varying levels of satisfaction.

The result of a tuning effort will be a reduction in the severity of a problem situation within the cost and policy limits imposed by the site. Under certain constraints, it will be impossible to resolve a problem; in other situations, problems will only be partially resolved. It may well be that the only, or best, solution to a problem is the addition of more hardware.

In order to determine when there are problems in system operation, you should have an understanding of the mechanisms which manage system resources and the available meters, so you can interpret the meters in light of the mechanisms in operation.

SYSTEM MECHANISMS

In order to pick an effective tuning strategy, you must have an understanding of the mechanisms by which the system functions. This knowledge, along with the values produced by the meters, will permit you to conceptualize where areas of inefficiency may exist, and whether they are amenable to tuning.

Segment Control

Multics relies on the management of gross storage entities as segments of memory. These segments are controlled through the *active segment table (AST)*, which groups all the known segments of the system into four size categories or pools: 0-4 pages, 5-16 pages, 17-64 pages and 65-256 pages.

Pools are used to minimize the overhead of memory dedicated to holding the *AST entries (ASTEs)*, each of which contains the page table for the pages of its segments. An ASTE takes 12 words for its header, and a word per page for its page table entries. Thus, in the space of a 256K AST entry (268 words), you can fit 16 4K entries, 9 16K entries and 3 64K entries. A system is typically configured with a large number of 4K entries, a smaller number of 16K entries, and with the 256K pool having the smallest size. This mirrors segment usage and size on a typical system, where the average segment size is roughly 5 pages. Thus, the vast majority of segments fit in 4K pool entries.

If a segment is mapped by an ASTE, it is *activated*; if it is not mapped by an ASTE, it is *deactivated*. Segments must be active to be accessed. If all entries in a pool are active and a new segment must be made active, then an old segment within the pool is deactivated to make room for the new segment. Deactivation requires that all of the segment's pages which are still in memory and have been modified must be written back to disk, and the contents of the ASTE written back to the VTOC entry on disk, before deactivation is complete. Typically, the set of AST entries more than maps the extent of physical memory on the system, and a large number of the entries have no modified pages in memory. Though the deactivation algorithm preferentially chooses segments which have the minimum deactivation cost associated with them, there is still the necessary VTOC update required to complete deactivation. Therefore, any activation or deactivation sponsors some disk I/O. The maximum disk I/O is sponsored if modified pages exist in the segment to be deactivated.

If there are insufficient AST entries in a pool, a system can experience *AST thrashing*. When this happens, demand deactivation removes an entry from the AST pool prior to its re-use. This will be seen in the meters as a very low AST pool *lap time* and will probably be experienced by users as a very sluggish system. (Lap time for an AST pool is the time it takes the segment replacement mechanism to look at all AST entries in the pool once.) The figure for AST lap time should usually exceed 200 seconds or so at peak system loading.

Page Frame Control -- The Clock

Memory is mapped in terms of page frames, each of which is 1024 words in length. This mapping utilizes a data structure called the *core map*, which is a circularly linked list of *core map entries (CMEs)*, one per page frame. Each CME references an ASTE page table word for a segment page which is in memory; each CME also points to the ASTE header. The core map entries are utilized by the page replacement algorithm to determine which page frames are free for use and which page frames must be written back to disk because they have been modified.

The *clock algorithm* is the Multics page replacement algorithm and sets the characteristics of virtual memory page replacement. It uses two pointers into the core map circular list. The first is termed the *replacer* (sst.usedp) and points to the current head of the list.

Each time a new page frame is needed, the replacer pointer is used to scan the core map looking for a page frame which is unused, not modified, and has a zero *pin count*. (Page pinning is a method of giving working set preference to pages faulted by special processes. A pin count associated with the process is copied into the CME for a page to be read from disk. Each time the purification process (described below) sees a CME with a nonzero pin count for an unmodified page, it decrements the pin count. The replacement process (described below) decrements the pin count for modified pages. Until a page's pin count has decremented to zero, it is not considered for replacement.) The contents of such a page frame can be discarded and the page table word altered to indicate the location on disk of the page's contents. The page frame is then used as the target for the demanded page to be read from disk or to be created if *zero*. (Zero pages are specially recognized. A page of zeros is not typically written to disk, but is instead flagged in the ASTE and VTOCE. A reference to such a page causes the paging system to create a page of zeros. Zero pages do not count against quota when they are flagged in this manner and are not in memory.)

The second pointer is termed the *purifier* (sst.wusedp) and points to a core map entry already passed by the replacer because it is a modified page whose contents no longer correspond to the page contents still on disk. The purifier pointer indicates a page which must be written to disk before the page frame contents can be discarded and the page frame made available for a new page. When the page write has been completed, this completion is *posted* back to the purification process, which marks the page as free and threads it into the core map ahead of the replacer, thus making it the next candidate for replacement. (Posting entails notifying page control of the completion of the I/O to either clear the page frame for a write or to notify a blocked process of the completion of a read.)

Replacement and *purification* are the two processes which produce the *least recently used (LRU)* page replacement algorithm. Page faults occur which require available page frames as targets for pages read from disk. This requires the replacement process to use the replacer pointer to scan the core map looking for a good candidate.

The replacement process runs the purification process if replacement has either skipped a *pre_seek_limit* number of modified pages (currently 15) while seeking a free page frame, or has completely circled the core map without finding a free frame while there exist one or more modified pages.

When run, the purification process initiates all necessary page writes between the current purifier and replacer pointers. A page is a candidate for writing if it has been modified since it was last read from disk, but has not been used since the last time the LRU algorithm scanned it. The replacer algorithm turns off the used flags for pages which have not been modified; the purifier algorithm turns off the used flags for pages which have been modified. Thus, a page will be written only if it has not been referenced at all since the last time it was seen by the purification process. Typically, this may take up to twice the current *lap time* of memory. (Lap time for page frames is the time it takes the clock algorithm to scan all memory frames once.)

The purification mechanism is important because of the I/O burst characteristics it gives to page writes. Since page writing is only initiated if a single page frame search for a free frame skips 15 modified pages, or the entire core map has been circled, the average situation is that more than 15 modified pages will typically exist between the purifier and the replacer pointers. It is likely that page writes will occur in large but moderately infrequent bursts, while page reads will occur with a rather even distribution.

This means that disk management must have the ability to handle deep bursts of activity within its queuing resources to handle these page write situations.

DISK MANAGEMENT MECHANISMS -- HARDWARE AND SOFTWARE

This subsection outlines the hardware and software mechanisms of disk management. It begins with the hardware, since Multics hardware has some characteristics for channel and drive access which form the basic structure of Multics disk management. This section is a more detailed explanation of mechanisms than that above, since a discussion of the Multics disk DIM is the prime purpose of this appendix, and you must understand the concepts of disk control to utilize the metering and tuning features of the Multics disk DIM.

There are two basic classes of hardware disk subsystems. The first consists of Honeywell-supplied devices in the following styles: 400, 451, 500, and 501. This class of disks is supported on both types of I/O mainframes, the IOM and the IMU. These devices have microprogrammed controllers (MPCs), disk controllers that interface with the system and disk drives. An MPC is a stored program computer that can be loaded with firmware by the system, and can execute various commands to control disk operations. Each command is a program within the MPC memory that ties up the MPC until command completion.

The other class consists of devices in the following styles: 3380 and 3381. This class of disks is only supported on the IMU I/O mainframe, and conforms to the Federal Information Peripheral Standard (FIPS). The disk controller functions for the FIPS devices are done by a combination of three hardware components: the head-of-string device, the storage directory, and the IPC-FIPS IMU channel. This IPC-FIPS channel in the IMU allows the system to interface with the FIPS disk subsystem in the same manner as the MPC subsystem. The firmware, however, is not loadable by the system.

Disk Controllers

Each disk controller can be connected to 32 disk drives and supports the ability to have all drives under its control simultaneously moving their head assemblies *(seeking)*. It does this by separating a combined I/O function such as a read or write into the component command functions of seek initiation and I/O. Since the seek operation is typically the largest physical delay in a disk read or write operation, this *seek overlap* capability provides noticable increases in efficiency.

Disk drives may be connected to two separate disk controllers to provide what is termed *dual porting*. The controllers support this mode of operation; there is no delay in switching control of a drive between controllers. This provides dual paths to drives and increases the hardware reliability and fallback capabilities of the system. Multics utilizes the possible multiple physical disk controllers and paths to distribute physical path loading as much as possible.

Physical Channels for MPCs

Disk controllers are connected to the Multics system through *physical data channels*. Physical data channels are the physical cable and board assemblies connecting disk MPCs to I/O mainframes. MSP0451, MSP0601, and MSP0603 disk controllers have two physical channels which can be in operation simultaneously. These controllers permit two simultaneous physical I/O operations to occur (read or write). MSP0607, MSP0609, MSP0611, and MSP0612 disk controllers are meant for faster transfer drives and only support a single physical data channel per controller. They can only perform a single physical I/O operation at a time. Certain of these faster controllers may appear to have two physical channels, but this is essentially packaging rather than function. The MSP0609 is two physical MPCs in two separate cabinets; the MSP0612 is two physical MPCs in a single cabinet. Each MPC is capable of only a single physical I/O transfer at a time over its single physical data channel.

IPC-FIPS Physical Channel

This channel reacts similarly to the physical channel for the MPC. It connects to a port on the storage director. There are two storage directors in a cabinet. Each storage director can be connected to the same head-of-string device cabinets. This allows dual porting to the FIPS devices. Each physical channel can only perform a single I/O data transfer at a time.

Logical Channels

Within the Multics I/O mainframe each physical data channel is logically divided into a number of addressable entities called *logical channels.* Each logical channel acts as the registers of a controller and can hold an I/O operation. This permits a single disk controller to receive a number of I/O requests simultaneously through the multiple logical channels configured on a single physical data channel. The disk controller will initiate all necessary seeks for these I/O requests in parallel and then wait for *on-cylinder* indications from the drives. When one or more drives are on-cylinder, the disk controller determines which of the possible physical data transfers to initiate, utilizing *rotational position sensing* to determine a best candidate. When the physical data transfer is initiated, the disk controller is tied up until the transfer completes, due to the program running in the disk controller. Thus, a disk controller permits seek overlap, but is limited to single data transfers per physical data channel.

You can see that there will be a one-to-one correspondence between the number of available logical channels configured to a disk subsystem and the number of seeks which can be simultaneously overlapped. This can effect system performance in situations where a high degree of seek overlap is essential for efficient disk service. On busy systems, for example one which runs 32 451 drives on a single string with 16 logical channels, it is not uncommon to see an average of eight drives simultaneously seeking. Serializing such I/O, by having fewer disk channels, would degrade disk system responsiveness.

Disk Subsystems

Multics disk drives are each configured with a physical drive number, which is utilized by the MPCs to address individual drives. Each MPC is limited to connecting up to no more than 32 drives, thus breaking the set of disk drives and disk controllers into a number of subsets, due to their physical connectability. These subsets are termed *disk subsystems.* All Multics disk drives are managed in terms of disk subsystems.

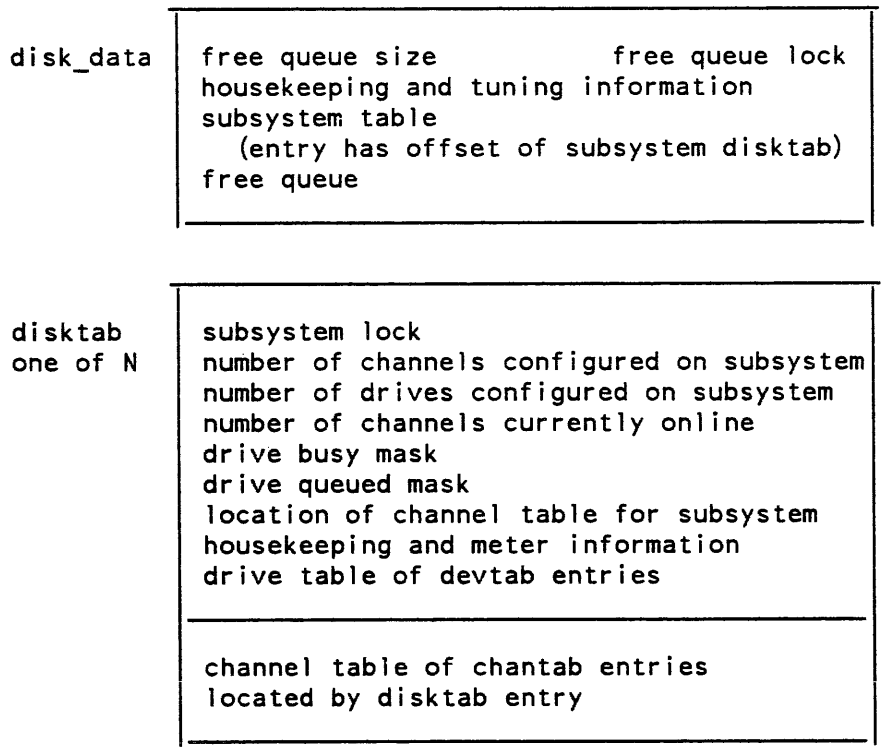
A disk subsystem is a direct expression of the physical connectability of its disk controllers and disk drives. One characteristic of a subsystem is that all of its drives can be accessed through any of the channels assigned to the subsystem. This characteristic greatly simplifies the task of channel selection for drive control and thus improves basic system efficiency. Depending on the types of MPCs involved, a subsystem will have two to four physical data paths divided into up to 32 logical channels.

Though a subsystem may have up to 32 drives physically connected to its MPCs, it is common for sites to limit the number of drives per controller to decrease data path contention. Thus, most sites have subsystems smaller than 32 drives each.

Disk Data Structures

Disk drives are controlled by a *disk device interface module (disk DIM)* which uses the databases contained within *disk_seg*. The first structure in disk_seg is *disk_data*, used for system-wide information and to locate all other disk control structures. The disk_data structure contains the *free queue* and *free queue lock*, the *subsystem table*, and system wide tuning information.

Individual subsystems are found by an offset from the base of disk_seg stored in the subsystem table. Each subsystem is represented by a *disktab* structure which holds all drive and drive tuning information, error counters, and meters, and has an offset from the beginning of disk_seg to locate the channel table for the subsystem. The following diagram illustrates the relationships contained within disk_seg.



Queues

Queues are used within disk management to store disk requests for drives and, in the case of the free queue, to establish a pool of entries available for queuing use. Queues all have the same structure.

All queues are found and controlled by a *queue head/tail control block (QHT)*, which contains a disk_seg offset to the head of the queue and an offset to the tail of the queue. The control block contains metering information such as the current number of elements in the queue *(depth)*, the number of allocations done to the queue *(count)*, the high water mark of the depth *(max_depth)* and the sum of the depths at the time allocations were done *(sum)*. The sum and the count permit a metering tool to take two readings and, by dividing the difference in the two sums by the difference in the two counts, to determine the *average queue depth* for the time period. Average queue depth is a request-averaged, rather than time-averaged, value. It is precisely the average of the number of requests which were in the disk queue at the time a request was queued. If four requests were queued into an empty queue, the average depth would be: (0+1+2+3)/4 or 1.5.

Queues are forward and backward linked for ease of insertion and deletion. One characteristic of all queues is that the request at the head of the queue is the oldest and the request at the tail is the most recent. This characteristic is exploited in determining request stagnation, as you will see later.

Queue entries contain all the information needed to process an I/O request. They hold a forward and backward link, an indication as to whether a request completion interrupt is needed *(posting)*, an entry in use flag, the I/O type, the core address of the I/O, the PVT index, and the primary device index and logical device index of the drive in the subsystem. They also indicate the cylinder and sector for the I/O and the number of sectors to be transferred. The final element of a queue entry is the full clock time when the entry was last allocated from the free queue.

The Free Queue

The free queue is the pool of available *quentry* elements set up when disk_seg is initialized. The number of elements in the free queue is a site tunable value and is controlled through the *config deck* with the *parm card's dskq* parameter. It can be specified in the range of 5 through 200 queue elements per configured physical volume. If the dskq parameter is not specified, the default is 20 free queue elements per configured physical volume.

The queue control block for the free queue is exactly the same as for any queue, with the exception that the free queue insertion and deletion routines use the depth counter to indicate the number of elements from the free queue currently in use, rather than the number of elements in the queue. This recognizes that queue elements are allocated from, rather than to, the free queue. Thus, the average depth for the free queue is a measure of the typical number of I/O requests simultaneously, though not continuously, loading the disk system.

Drive Queues

The Multics disk DIM utilizes a single request queue per drive which holds both high and low priority requests for the drive. As you will see later, this provides efficiency and ease of request optimization.

Note: FIPS devices are divided into subvolumes. When the disk DIM is called, the subvolume record address is converted to a device record address. This allows the disk DIM to manage and meter the device as one entity; it need not track each subvolume.

Disk Channels

The *channel table* for each subsystem is dynamically allocated according to the number of channels found in the config deck for the subsystem, and permits up to 32 disk channels to be configured. This is essentially the hardware limit for the maximum MPC configuration possible for a single subsystem of eight logical channels per physical data path.

The channel initialization algorithm of the Multics disk DIM does not impose any limits on the configuration of channels.

(Note: since the disktab structure has *devtab* as a dynamic sized array, the *chantab* array of the pre-MR11.0 disk DIM has become *disk_channel_table* and is located through the *devtab.channels* offset.)

Disk Software Modules

The software for disk management utilizes both a PL/1 disk DIM *(disk_control)* and an ALM disk DIM *(dctl)*. The PL/1 disk_control module is a complete and functional disk management system with error recovery and reporting, *emergency shutdown (ESD)* reinitialization, queuing, and interrupt management. The ALM dctl module contains a subset of this functionality and is only capable of queuing and simple interrupt management. The ALM routine is the normal and preferred path for disk management, since it is roughly three times faster in operation, but it defers complex interrupt management, disk running, error reporting and recovery to the PL/1 routine.

DISK MANAGEMENT

Disk management is broken into two operational parts, the *call side* and the *interrupt side*. The call side handles all requests to read or write a sector or page on disk. It allocates and initializes a disk queue element for the request and immediately initiates the I/O if the drive is free and a channel is available. If a channel is not available or the drive is currently busy, disk management queues the request for the drive.

The interrupt side handles all interrupts and errors and is called through the *fim* to manage the completion of an I/O operation. It validates the completion, updates meters, frees the channel, and returns the queue element to the free queue. If a drive has a queue of requests and is not busy, the interrupt side will initiate I/O for it using the channel just freed.

Allocation Locks

If the disk DIM is given a request and the free queue resource has been totally consumed, the DIM enters an *allocation lock* situation. In this situation, it goes through a loop of ensuring that all possible disk drives are busy and waiting for an I/O completion which will free a queue element. In very busy systems, allocation locks can account for better than 20% of the overhead of the system.

In the Multics disk DIM, the ability to share the free queue resource among all subsystem drives makes good use of the committed memory, given the load statistics for drive and subsystem activity. Since the size of the free queue is tunable through the config deck, and defaults to a reasonable number of queue elements per configured drive, a site should be able to configure an adequate free queue size, even for high loading situations, to virtually eliminate allocation lock overheads.

The Masked Environment -- Running

When the disk DIM is called to make an I/O request, or interrupted to handle an MPC or drive action, it runs in a masked environment and cannot detect interrupts signalling I/O completions. The DIM utilizes a *polling* mechanism to detect I/O completions when it is in this state. This polling is called *running the disks* and entails having the DIM check the status of every active channel to determine if an I/O has completed. If an I/O has completed, the queue element is returned to the free queue and a new I/O request may be initiated. Disks are run whenever an allocation lock occurs, since the DIM must wait for an I/O completion to recover the queue element. Disks are also run every 15 seconds from a timer event in page control to ensure no I/Os are missed. The ALM disk DIM calls the PL/1 disk DIM to run disks if an allocation lock situation occurs.

Disk running occurs on the drives of all possible subsystems, since the DIM may need to recover a queue element from a different subsystem than the subsystem encountering the allocation lock. This recovery is done by running the current subsystem, and then running all other subsystems which are immediately lockable. This prevents a deadly embrace situation waiting for a locked lock. It is valid because if a subsystem is currently locked, it is currently doing I/O, and running is not necessary.

Running all subsystems has the additional advantage of completing I/O for other subsystems while masked which would not otherwise be detected until control had returned from the paging system and the processor became unmasked.

Blocking vs. Non-blocking I/O

The Multics virtual memory system controls the execution of a process within the system. A process which has all its pages in memory executes at the full speed of the processor. If the process tries to access data on a page which is not in memory, it must wait until the page is available. The process is then *blocked for execution.* When a process is blocked, some other process must be found to utilize the processor(s), otherwise the system will be *idle* and not be used at maximum efficiency.

A process can be blocked while waiting for a page to be read from disk to memory. If the segment containing the page is not resident in the AST (activated), then the process must wait for the VTOCE I/O necessary to establish the ASTE page table needed to find the page. Thus, it may take up to two disk I/Os to permit a process to continue.

Processes are typically not blocked by pages being written to disk, since this is a background task of the virtual memory system, and not directly tied to any normal process function. As such, VTOCE write and page write operations are *non-blocked*.

Multiprogramming

As seen above, if a process is blocked for execution, there must be another process available for execution if the processor is to be properly utilized and not go idle. *Multiprogramming* is the term used to describe this action, and the *level of multiprogramming* expresses the number of processes which are typically available for execution at the same time.

In Multics, the level of multiprogramming is controlled by the *maximum eligible* tuning value. This indicates the maximum number of processes which may be in the *eligible queue* at a time, and which are then candidates for execution. Most Multics systems run in the range of 10 to 20 or more processes in the eligible queue. This provides quite a broad base of processes, at least some of which will probably be available for execution.

Figure J-1 provides an indication of the wait percentage and the required levels of multiprogramming for various system wait times. The vertical axis indicates the level of multiprogramming (i.e., the number of processes); the horizontal axis indicates the individual process wait percentage. The table elements indicate the resultant system wait times. Table information is taken from the book *Operating Systems* by Madnick and Donovan (MIT Press).

MP Level	Wait Percentage Average per Process									
	10.0%	20.0%	30.0%	40.0%	50.0%	60.0%	70.0%	80.0%	90.0%	100.0%
1	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.0	100.0
2	1.0	4.0	9.0	16.0	25.0	36.0	49.0	64.0	81.0	100.0
3	0.1	0.8	2.7	6.4	12.5	21.6	34.3	51.2	72.9	100.0
4	.....	0.2	0.8	2.6	6.3	13.0	24.0	41.0	65.6	100.0
5	.....		0.2	1.0	3.1	7.8	16.8	32.8	59.0	100.0
6	.....			0.4	1.6	4.7	11.8	26.2	53.1	100.0
7	.....			0.2	0.8	2.8	8.2	21.0	47.8	100.0
8	.....				0.4	1.7	5.8	16.8	43.0	100.0
9	.....				0.2	1.0	4.0	13.4	38.7	100.0
10	.....					0.6	2.8	10.7	34.9	100.0
11	.....					0.4	2.0	8.6	31.4	100.0
12	.....					0.2	1.4	6.9	28.2	100.0
13	.....					0.1	1.0	5.5	25.4	100.0
14	.....						0.7	4.4	22.9	100.0
15	.....						0.5	3.5	20.6	100.0
16	.....						0.3	2.8	18.5	100.0
17	.....						0.2	2.3	16.7	100.0
18	.....						0.2	1.8	15.0	100.0
19	.....						0.1	1.4	13.5	100.0
20	.....							1.2	12.2	100.0

..... Lines indicate 100% processor use

Figure J-1. Multiprocessing Table of Wait Percentages vs. Multiprogramming

Request Optimization

It is important to choose an optimization strategy for scheduling disk I/O requests which provides maximum system efficiency. Such a strategy is not one which simply provides maximum disk throughput, since you must consider the effect of a disk optimization strategy on the system as a whole, not just on the disk system.

Since a blocking I/O causes a process to wait and possibly idles a processor, it is important for blocking I/O operations to be completed as quickly as possible, to maximize system throughput and response.

However, it is possible for the queuing resource, which holds both blocking and non-blocking requests, to become saturated. At this point, non-blocking requests become blocking requests waiting for queue resources to become available, and in turn block the queuing of normal blocking requests. This is termed an allocation lock and was described earlier. An allocation lock stops system paging until a physical I/O completes, but does not change optimization strategy. As a result, the system runs in a de-optimized mode and suffers high paging overheads due to allocation locks.

As disk loading increases in a system, the simple optimization of high and low priority I/O requests leads to situations where a drive is completely occupied in servicing high priority requests, and never services low priority requests. Thus, the low priority requests *stagnate*. At this point, a significant queue builds for a busy drive and may result in an allocation lock situation. Even if the free queue is made large enough to prevent an allocation lock situation, the system will become sluggish and unresponsive. Further, there will still be no attempt to service the low priority requests and the queue will not shrink.

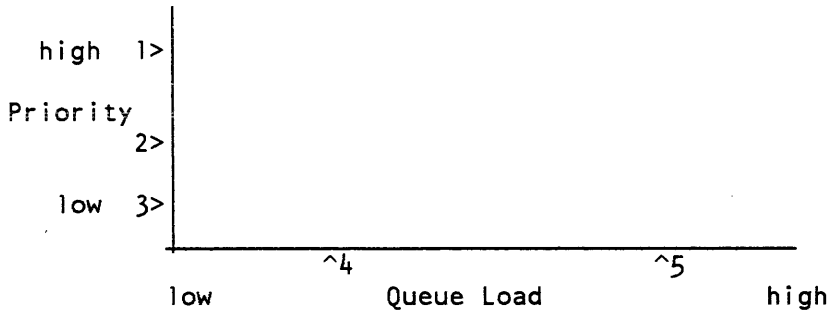
The Multics disk DIM has features aimed at optimizing such situations to attempt to reduce or eliminate allocation lock overheads and to preserve smooth and responsive system action. This is done through an optimization strategy, *load adaptive disk optimization*, which recognizes the problems caused by queue resource saturation and the different service priorities needed for different types of I/O.

Load Adaptive Disk Optimization

Load adaptive disk optimization recognizes that each type of disk I/O, VTOC read, VTOC write, page read and page write may require a different scheduling priority as far as efficient system operation is concerned. It also recognizes that this priority may depend on the queue load that the I/O type imposes on the system such that when heavily loaded, the priority should be increased, and when lightly loaded, the priority should be decreased.

For example, if the system is very lightly loaded by I/O requests, there is a lot of queue resource left to buffer non-blocking I/O requests. In this situation, it is desirable to provide the best throughput and I/O response to blocking I/O requests. However, as the I/O queues become more heavily loaded and approach queue resource saturation, it is desirable to increase the priority of non-blocking requests to lighten their queue loading and avoid allocation locks.

These two limits are termed the *response point* and the *load point*. The response point is defined by a queue loading of a single request, and the I/O priority desired for the drive for best system wide response and throughput. The load point is defined by the maximum queue loading desirable for that I/O type for the drive and receives the maximum I/O priority. The following diagram illustrates these concepts.




```
2   Defines the read type's  response point (Qload=1)
3   Defines the write type's response point (Qload=1)
1,4 Define the read type's  load point
1,5 Define the write type's load point
```

At any point in time, each disk drive will be operating at some point between these two limits. This scheduling priority adjustment should be implemented on a per-drive basis to degrade blocking I/O response, system wide, as little as possible. This is due to the need to avoid resource saturation and I/O stagnation for a single drive.

This is precisely what the Multics load adaptive disk optimization does. For each drive, it maintains an *optimization table* which has an entry for each type of disk I/O. Currently defined types are:

Page read
: is used for reading pages from disk to memory. It is a blocking I/O type.

Page write
: is used for writing pages from memory back to disk. It is a non-blocking I/O type.

VTOC read
: is used for reading VTOC sectors from disk to memory. It is a blocking I/O type.

VTOC write
: is used for writing VTOC sectors from memory to disk. Such an operation is non-blocking but is buffered in the very small VTOC buffers. As such, it can very easily become a blocking I/O when the VTOC buffer resource saturates.

Test
: does not actually initiate I/O at all. This type of I/O is never queued, though it does use a queue element. As such, it cannot cause queue resource saturation, though it can be delayed by saturation.

Bootload read
: only occurs within BCE during initialization of the system. This I/O operation has no effect on normal system operation.

Bootload write
: only occurs within BCE during initialization of the system. This I/O operation has no effect on normal system operation.

Each drive optimization table entry holds a number of values used to determine the optimization priority of requests of its I/O type. Each time an I/O request is queued or dequeued for a drive, its I/O type is used to determine which optimization table entry should be used. Then the current queue depth of that I/O type for the drive is determined and used to set the optimization priority. That priority stays in effect for that I/O type on that drive until another request is queued, or a request is finished and dequeued, and the priority is recalculated.

The Multics disk DIM has been implemented to make it easy to define and optimize new I/O types, as they become identified and their characteristic optimization and resource loading is determined.

Implementation of Prioritization

There are two basic methods of optimizing disk operations: *disk combing* and *nearest seek*. Disk combing, sometimes called windshield wiping, involves moving the read/write head assembly continuously in one direction servicing requests until no further requests can be serviced in that direction. Then head direction is reversed and servicing continues. It causes the heads to comb across the surface of the disk like the windshield wipers of a car. It has the advantage of ensuring that all disk requests will be serviced, but has the disadvantage of not optimizing the ordering of service to any particular system advantage. Since requests can wait for a long time if they are behind the current head position, it produces indeterminate system wait characteristics for a system such as Multics.

The other basic method, nearest seek, involves determining the best seek from the current position to be the shortest seek possible of all candidates in the queue. This ensures minimum seek times, but does not ensure that all requests will be serviced. If the arrival rate of requests is higher than the drive processing rate, then some requests will remain unprocessed and will stagnate. This is particularly true if there are distinct bands of I/O activity on the surface of the drives, since long seeks will be required between bands with much shorter, and therefore preferential, seeks within a band.

Nearest Logical Seek

The Multics disk DIM modifies the nearest seek algorithm to become a *nearest logical seek* algorithm. It does this by determining a logical seek according to the priority of the I/O type and the physical seek length of an I/O request from the current head position.

A logical seek is the physical seek length multiplied by an optimization factor for the I/O type in question. A high priority I/O has a low optimization factor; a low priority I/O has a high optimization factor. The result is that for the same physical seek length, a low priority I/O will have a longer logical seek length than a high priority I/O, and will be preferentially chosen.

This results in the following definitions for the response point and the load point:

response point (a seek multiplier in units of cylinders)
The response point occurs at a queue loading of one request and specifies the optimization factor to be applied to the physical seek length which will be appropriate for this I/O type in terms of deriving maximum system responsiveness.

load point (a queue loading in units of requests)
The load point occurs at a specified queue loading where maximum I/O throughput of the I/O type is desired. This occurs with an optimization factor of one, meaning that physical seek length = logical seek length. Above the load point, the optimization factor continues to be one.

Algorithm Implementation

It is essential that the load adaptive disk optimization algorithm be efficient to implement and comprehensive to use. This subsection details the implementation and action of the algorithm.

The nearest seek algorithm lends itself well to the needs of adaptive optimization. As seen above, you can utilize a nearest seek to optimize logical seeks just as the nearest seek algorithm normally optimizes physical seeks. An efficient conversion of physical seeks to logical seeks can be done by simply multiplying the physical seek length with a factor appropriate for conversion of physical seeks to logical seeks.

The nearest seek algorithm normally does a simple comparison between seek lengths of all possible requests and the current head position, and then selects the shortest seek. The nearest logical seek algorithm does the same, but multiplies the physical seek length by the optimization factor for the I/O type of the seek before doing the comparison. Thus, only a multiplication is added to the innermost loop of the nearest seek algorithm.

The response point and load point can be considered the endpoints of a line, called the *optimization line*, defined with an X axis of *queue loading* and a Y axis of *optimization factor*. As seen above, the optimization factor is a multiplier of physical seek length from which the logical seek length is derived. The line can be characterized by the formula:

```
Y = b - a*X
```

where Y is the optimization factor and X is the current queue loading of the I/O type. Using the response and load points:

```
a = (Response Point-1) / (Load Point-1)
b = Response Point - a
```

(In the preceeding straight line formula, the uncommon use of a subtraction of the slope from the intercept is done to produce positive slope and intercept values.)

The optimizing table entry holds the slope and intercept values for the optimizing line specified for that I/O type for that drive, the current queue load, and the current optimizing factor. Each time a request of that I/O type is queued, the queue load value is incremented and the optimization factor is calculated from:

```
factor = max (1.0, b - a*X)
```

(The use of the MAX function prevents queue loading beyond the load point from deoptimizing seeks beyond the maximum load.)

When the nearest logical seek algorithm runs, it need only refer to the already calculated optimization factor. The only complex calculations needed to update the factor occur once for a queue insertion and once for a queue deletion.

Nearest Logical Seek Examples

The examples which follow indicate the possibilities of this method. They utilize two I/O types, R and W, which have the following cylinder requests in the queue:

```
Type R: 1, 20, 30, 100
Type W: 10, 50, 55, 60, 80

Current head position on cylinder 70.
```

To mirror the pre-MR11.0 queue separation, simply set an optimization factor for low priority I/Os greater than the number of cylinders on the drive, and an optimization factor for high priority I/Os of one. Then any seek for a low priority I/O (W) will look longer than any possible high priority I/O (R). You then arrive at logical seek lengths of:

```
R Factor: 1,   W Factor: 101

R Logical seeks: 69, 50, 40, 30
W Logical seeks: 6060, 2020, 1515, 1010, 1010
```

The obvious best seek is the R seek of 30 logical cylinders. And from there all R seeks will be done before any W seeks are done.

If you lower the W factor, as would occur with increased queue loading, you can see the kind of interactions possible:

```
R Factor: 1,   W Factor: 3

R Logical seeks: 69, 50, 40, 30
W Logical seeks: 180, 60, 45, 30, 30
```

At this point, the last R seek and two of the W seeks look equally good. If you take the first of the W seeks, you then get into a close band of W I/Os and finish them off.

Since the optimization factors are being adjusted each time that requests are added to a queue or removed from a queue, the relative priorities constantly change to reflect the immediate loads of the drive. Thus, the optimization relationships are constantly changing to follow the dictates of the established optimization policies and the drive loads.

These examples also demonstrate the tendancy of load adaptive disk optimization to cluster I/O requests. Low priority I/Os tend to accumulate in the queue until their optimization factor has become small enough to make a low priority I/O become preferential. The accumulation of requests tends to make request clusters, in which there is very little difference in cylinder positions.

Once a cluster is entered, it becomes preferential to stay within it, servicing I/Os with a short seek length, until I/O loading drops sufficiently to make the much longer higher priority I/Os look "good" enough to make leaving the cluster worthwhile. This can produce bursts of activity with very low seek lengths as appropriate for the prevailing load/request characteristics. It will tend to recover drive throughput without significantly losing drive responsiveness.

Optimization Policies

The Multics disk DIM permits each type of I/O on a drive to be given a specific optimization policy, to dictate its optimization. This policy is stated in terms of the response point and the load point, and establishes the optimization line for the I/O type. The relationships between the optimization lines for all the I/O types for a drive establish the drive policy.

You should choose policies to meet the individual I/O requirements, remembering that the optimization factor is a multiplier used to convert physical seek length to logical seek length for a nearest logical seek comparison.

For example:

VTOC reads
: are essential to establish the conditions to permit page I/O to occur. Thus, VTOC reads are *top* priority.

VTOC writes
: are essential to clear the VTOCE buffers quickly. If any number of them build up, they should occur quickly.

Page reads
: are essential to unblock processor execution, but VTOC reads are more important. Typically, VTOC I/O appears to be about 100 cylinders average from where page I/O occurs.

Page writes
: are essentially background and shouldn't interfere. There are 200 free queue elements.

With these broad statements in mind, you could establish the following response points:

VTOC read
: response optimization factor is one for maximum throughput. Load point is one request (since the factor is one, this really doesn't matter).

VTOC write
: response optimization factor is ten, to leave some headroom for optimization of other I/O types at high loads. Load point is three requests, which is roughly 10% of the VTOC buffer resource.

Page read
: response optimization factor is 100, to leave optimization headroom and make VTOC I/O more attractive. Load point is six outstanding requests for adequate multiprogramming levels.

Page write
: response optimization factor is 80000 to completely move write seeks outside of the longest logical page read seek (presumes 799 cylinders on the drive). Load point is 150 outstanding requests, to leave extra headroom in the queue to handle large bursts before allocation locks occur.

Such a policy ensures that VTOC and page read operations get good optimization and speeds system responsiveness and throughput. Any on-cylinder requests will be optimized. If you get very large page write queue buildups, you will start to optimize page writes preferentially to page reads until queue loading drops again.

Optimization Dynamics

The optimization of I/O types is very dynamic in operation, since it is a combination of queue loadings and physical seek lengths. The specific optimization relationships change from I/O to I/O, as the queue lengths change. Optimization relationships also are completely different for different drives experiencing different loadings. Thus, the majority of drives will experience optimization relationships which optimize VTOC I/O and page reads, while only a few heavily loaded drives will decrease blocking response to accentuate non-blocking throughput to decrease queue lengths. These relationships and drive loadings will be constantly changing.

One of the expected characteristics of a loaded system using load adaptive disk optimization is that the service ordering of requests will be altered to provide needed response and throughput on a drive-by-drive basis. This may make it difficult to do precise benchmarking between disk optimization systems, since completion times of benchmark processes will vary as a function of drive optimization and request clustering.

Systemic Optimization

So far, load adaptive disk optimization has been dealt with as it applies to single disk drives and their required responsiveness, and the limiting of queue loadings. However, the queue resource is a system wide resource, rather than a disk subsystem or disk drive-related resource. As such, we must also manage the loading of this resource system wide must also be managed, to prevent allocation locks and degraded system response.

This is handled through a *system optimization table*, residing in disk_data, which tracks the system wide queue loading of I/O types. This table has one entry for each I/O type which contains:

depth
: the current system wide load of this I/O type.

max_depth
: the maximum permissible system wide load of this I/O type.

fraction
: a fraction expressing the relation:

```
fraction = max (0.0, (depth-max_depth)/max_depth)
```

This fraction is used in the nearest logical seek algorithm to adjust the drive optimization factor as:

```
factor = max (1.0, fraction * factor)
```

depth_map
: indicates which system optimization entry is to be used for counting I/Os of this type. It permits you to use a single system wide depth counter to combine queue loadings of more than one type.

The system wide fraction is used to condition the logical seek length and to increase the priority of an I/O type on a drive when the system is approaching a predetermined system wide queue loading limit. This limit does not have to be the same as that for any drive. Thus, if each individual drive is permitted to use an appreciable fraction of the system wide queue resources before the optimization algorithm produces a low optimization factor for the drive, the possible over-commitment of parallel loading of a number of drives is managed by the system wide optimization fraction.

This recognizes the uneven burst loading characteristics of the system, by permitting relatively large individual drive queue limits, without imposing the dangers of large distributed bursts causing queue resource exhaustion.

The system load counters utilize a *counter mapping* technique to permit more than one I/O type to contribute to a combined system loading. For example, VTOC operations could map VTOC read and VTOC write to the same counter (VTOC read) to permit a single VTOC load figure to be derived for the system. This figure could be used to accelerate VTOC optimization of both VTOC read and VTOC write if loading of either increases. (*Note*: system load counters are not permitted to become negative, which could happen through remapping "on the fly." After a counter remapping is done, it is recommended that you reset the depth counters using the tune_disk command (described later) to let them find their own level again. Otherwise, a depth counter might be left artificially high.)

Stagnation Management

Throughout this discussion of the load adaptive disk optimizer, the dangers of request stagnation have been seen, where the arrival rate and cylinder characteristics of requests prevent some of them from ever being serviced by the nearest seek algorithm.

The Multics disk DIM manages stagnation characteristics by changing its optimization policies when stagnation situations occur. This is all part of the load adaptive disk optimization.

Stagnation management utilizes the guaranteed service characteristics of the disk combing technique and the queue age characteristics mentioned earlier (i.e., that the oldest request in every queue is at the head of the queue and the youngest request is at the tail.) Prior to scanning a drive's queue to determine the nearest logical seek, a check is done of the oldest request in the drive queue. If it is older than a site tunable *stagnation time*, currently defaulting to five seconds, then a disk combing algorithm is used, rather than a nearest logical seek.

The disk combing continues motion of the head in the current direction of travel and finds the nearest request of any type in that direction. Each time the drive is ready for another I/O this time comparison is done, and unless the oldest request becomes younger than the stagnation time, disk combing continues. When no more requests in the current direction exist, the direction is reversed.

It is important to note that though the oldest request is older than the stagnation time, it is not necessarily the request which is picked for servicing. Thus, a number of disk I/Os may be sponsored under disk combing until the oldest request gets serviced. And by this time some other request may be old enough to continue disk combing.

The end result is that the load adaptive disk optimization does provide as responsive a system as possible for prevailing conditions, and guarantees servicing of all requests. (It is possible to create a secondary stagnation situation under extremely high disk loading. Secondary stagnation is thrashing in which the instruction page is paged out before the data page is referenced. This requires the instruction page to be brought in before execution can continue. To date this has only been produced in load tests.)

Use of Adaptive Optimization

Load adaptive disk optimization is a Multics disk DIM feature aimed at continued system responsiveness and throughput under high disk load situations. It has no effect under light load situations and little effect under moderate load situations.

With it, a site can expect to see a change in the request servicing characteristics of disk drives as these drives become busy. Visible head motion becomes smoother and smoother as drive load increases, rather than simply becoming a more frequent head move. As drive loading increases to the point where stagnation periods for requests approach the stagnation limit, the true disk combing motion becomes apparent.

Experience with load testing indicates that this optimization technique has broad tuning response. This makes it easy to use and robust in action. Unless the optimization priority relationships are reversed, the system action is acceptable through a broad range of load conditions. If optimization priorities are reversed, to give highest priority to non-blocking I/O, system responsiveness and total system throughput is noticably degraded.

METERING

Multics is a system rich in meters, but only a few of them are really essential in determining problem areas. Some important metering values are:

Virtual CPU Time

This is the bottom line. It is an indication of how much of the CPU hardware on the floor is being delivered to the users of the system. It is a meter value produced by the *total_time_meters (ttm)* command and is stated in terms of absolute processor percentage and non-idle processor percentage. It is the summation of all the virtual CPU time delivered to individual processes.

Idle Time

Idle time is the extent to which a processor is not doing any work, and represents a possible area of CPU recovery through tuning. There are five idle times of interest, all produced by the *ttm* command:

Zero Idle

Zero idle is the time used by the system idle process when no other processes are running, none are ready, none are waiting, and no processes hold the page table lock. It indicates that there is simply no work to be done. No amount of tuning will recover zero idle, but changes to process scheduling limits may allow better system use, by permitting otherwise-held processes to be executed.

NMP Idle

Non-multiprogramming idle is the time used by the system idle process when there are no processes eligible to run, but the system is not in a zero idle situation. It indicates that there are not enough processes available to provide sufficient system load. It indicates a situation in which disk tuning will probably not recover idle processor time, but changes to process scheduling, for example absentee limits, may provide better system use.

Loading Idle

Loading idle is the time used by the system idle process when there are processes eligible to run, but the last eligible process is not yet loaded. (A process is loaded when page 0 of its pds and dseg are wired in memory.) It indicates a situation where there is sufficient paging for the PDS and DSEG pages of a process to get paged out between the time when the process becomes ineligible and the time when it becomes eligible again. This can also be due to long inter-eligibility time delays in relation to the current memory lap time, causing thrashing.

MP Idle

Multiprogramming idle is the time used by the system idle process when the maximum permissible number of processes eligible to run has been reached, or no processes missed becoming eligible because a work class maximum has been reached or exceeded. It indicates that there are processes to execute, but that one or more processes are blocked from execution, typically by I/O in progress, and not all processors are busy. It may indicate a situation in which disk tuning will have a payback.

Work Class Idle

Work class idle is the time used by the system idle process when processes miss becoming eligible because a work class maximum has been reached or exceeded and the maximum number of eligible processes has not been reached. It indicates a situation where changes in usage limits might recover more of the processor resource.

Overheads

There are a number of areas in which overheads are metered. Overheads can be due to the amount of work to be performed and be essentially unrecoverable by tuning or they can be due to resource saturation which causes a processor or the system to *spin* in a hard loop until the resource becomes available. Resource saturation overheads are usually amenable to tuning recovery. Some of the important overheads shown by the *ttm* command are:

Interrupts

This is an indication of the overhead required to service interrupts. Most interrupts are due to communications and disk management. Interrupt overhead is typically a direct function of the amount of work which must be done. Recovery of disk interrupts will usually only occur as a secondary result of tuning, which causes less thrashing to occur, and hence less disk I/O.

Page Faults

This is an indication of the overhead required to run the paging system. It can involve overhead required for a number of spin-locks in page control, and can be an indication of saturation of queuing resources. Since the queue resource is a tunable value, a site should easily be able to set a free queue size which eliminates allocation lock situations.

The mechanisms of page control and the file system are metered through the *file_system_meters (fsm)* command. Of primary interest are the meter values pertaining to segment activation and deactivation, the use of the AST pool and AST locking, and system paging activity and its origins.

Segment Activation and Deactivation

Segment activation and deactivation is one of the mechanisms found within storage management. It makes segment contents accessible to processes, or removes them from memory. It is the way in which the AST pool resource is managed. Several important meter values are available:

Activations

Activations make a segment known to Multics processes and put the segment and page table information into an ASTE. Segments can be activated due to segment faults, due to an explicit command to make them known, for the backup system, or to make directories known and their contents available. The various meter values are: *segfault, makeknown, backup,* and *directories.*

Deactivations

Deactivations are used to remove entries from the AST, either explicitly to make them unknown or to recover their space to activate another segment. Recovering space is done by *demand deactivation* of the existing entry and does not constitute the majority of most deactivations for a system with sufficient AST pool sizes.

AST Locking and Searching

AST management is done through the four pools of the AST. The *fsm* command provides quite a bit of information to determine what is occurring:

AST Lock Information

The AST is a system wide database which is protected by a locking mechanism to ensure serialization of updates. The *AST locked* value indicates the average locked time for the lock and the percentage of time the lock is locked. The *AST lock waiting* average and percent value indicate the time lost by processors waiting for the lock. The amount of time consumed by processors waiting for the lock may be tunable by increasing the number of AST entries. See the discussion of *lap times* under "AST Pool Usage Information" next.

AST Pool Usage Information

The AST pool information for the 4K, 16K, 64K, and 256K AST pools indicates the usage of the AST entries. The average step size and lap time of the pool can indicate whether the AST pool size is appropriate for the typical demand. If lap times drop below 200 seconds or so, an unreasonably large number of demand deactivations are required within the pool to find entries for new segment activations.

Paging Activity

Paging activity is the final section of the *fsm* output. It provides a number of interesting values:

Needc

This value indicates the total number of page frames required in the metering period and indicates the average time between page faults. It is an indication of total system paging activity. Various page fault types are shown in a number of meter values as percentages of total paging activity. Since each meter value is considered separately, and categories may overlap, it is possible for the percentages to add up to a sum greater than 100%.

Ring 0 faults

This is the percentage of paging activity for page faults taken while the ring of execution is ring 0.

PDIR faults

This is the percentage of paging activity for page faults taken on segments with the *per_process* flag set. It is an indication of the amount of total activity which is for temporary work space pages.

Level 2 faults

This is the percentage of page faults taken on segments one directory level below the root. These include segments within project directories, and also segments in many system directories such as ">sc1". The latter would typically be system tables and installed software in directories such as ">sss", ">unb", etc..

DIR faults

This is an indication of the page faulting required to bring directory pages into memory.

New Pages

This is an indication of the page faulting activity due to the creation of new pages. It indicates activity which generates new information rather than that which brings existing information from disk. Page creations do not require data I/O and are done by zeroing the contents of a page frame.

Zero Pages

This indicates page writing activity in which a page which has been modified and is supposed to be written to disk is determined to contain zero words only. This is a special case and typically means that the ASTE page table word for the page has flags set to indicate a page of zeros, and the page is simply discarded rather than written.

Lap Time

Lap time is the time it takes the page frame management algorithm to scan all the pages in the core map, and is an indication of the amount of time a page will remain in memory if unreferenced and unmodified. This can be directly compared to the typical user *think time* or *intra-eligibility* time period to determine if a process will thrash. Thrashing occurs if a process loses it pages before it requires them again. A process which is not eligible more frequently than the lap time of memory stands a very good chance of losing its pages in just this manner. This will cause extra disk I/O and produce poor response times.

Page Frame Meters

A number of meters exist which indicate the activity and characteristics of the page frame management algorithm. These are displayed by the *fsm* command and indicate quite a bit about the characteristics of the contents of main memory and their utilization.

Steps

This is the number of steps taken around the core map during the metering interval. It indicates the number of core map entries which have been looked at to handle the number of page faults previously given by the *needc* value.

Skip

During the search of the core map to find available page frames, the management algorithm skips a number of frames. The *skip* value indicates how many frames are skipped, and what percentage of all *steps* are *skips*. This is an indication of the number of pages actually replaced within the *lap time* of the core map, and hence, the page I/O rate.

A page frame is skipped if it is *wired* in memory and therefore cannot be discarded; if it has been used in the last *lap time*; if it has been modified and must be written to disk before it can be discarded; if it is pinned in memory for performance reasons.

The disk_meters Command

Almost all interesting values in disk management can be determined from the output of the *disk_meters* command. There are a number of different sections to the meter output:

System Information

System information indicates the utilization of the free queue, including the average number of requests queued when a request is being queued, the number of queue allocations made, the peak depth of requests, and the maximum size of the free queue, as well as the number of requests currently queued system wide.

The system section of the output includes the stagnation time value and the number of times the ALM DIM received a complex interrupt which required it to call the PL/1 DIM for interrupt processing. It also indicates the last time at which the maximum depth value for queue meters was reset.

The last part of the system section indicates the system loading table values, including the I/O type, the maximum load point for system optimization, the current depth of requests for that I/O type, and what counter is being used for counting the current depth. Finally, it indicates the current system optimizing fraction for that I/O type. For example:

```
FREE Queue:Ave 12.1,Alloc 27055222,Max Depth 986/2720,Cur Depth 10
      Stagnate time 5.000 seconds, 646793 PL/1 interrupt services.
          Maximum Depth Meters reset at: 01/05/85  0440.1 mst Sat
   PageRd   Max Load    6, Depth   3 (PageRd), Fraction 0.5000
   PageWt   Max Load 2040, Depth   2 (PageWt), Fraction 0.9990
   VtocRd   Max Load    6, Depth   0 (VtocRd), Fraction 1.0000
   VtocWt   Max Load   12, Depth   0 (VtocWt), Fraction 1.0000
   BootRd   Max Load    6, Depth   0 (BootRd), Fraction 0.0000
   BootWt   Max Load   12, Depth   0 (BootWt), Fraction 0.0000
```

Subsystem Information

The subsystem information lists the error counts of EDAC errors, fatal errors, and general errors. EDAC errors are data errors recoverable through error correction logic; fatal errors are data errors which could not be recovered; general errors are things like seek retries.

The error information is followed by lock information, detailing the number of lockings made, the number of times a locking attempt was made when the lock was already locked, and the percentage of all calls which had to wait. This is followed by the average time of the delay and the percentage of realtime caused by waiting. For allocation locks, the lock count is the number of allocations, and lock waits are due to exhaustion of free queue resources. For example:

```
Subsystem  dska: 8 Errors 39 EDAC Errors
                  Locks    Waits     %Calls  Average      %CPU
  Call Lock:   15175281   586739   3.8664%    0.539   0.08345%
  Run Lock:       36022     1764   4.8970%    0.366   0.00017%
  Int Lock:    15175037   625676   4.1231%    0.498   0.08216%
  Alloc Lock:  15171250        0   0.0000%    0.000   0.00000%
```

Detailed Drive Information

If detailed drive information is requested, a full breakdown of I/O activity per I/O type per drive is provided. This indicates the seek count, average seek length, queue wait average, channel utilization, and channel wait average. In addition, the current queue depth and the optimizing factor *(multiplier)* for the I/O type is given. Queue information may also be listed. It is of the same form as the free queue information in the system information block. The final area of the output is drive busy information, indicating total channel use, the number of I/Os done by combing, and the average drive I/O rate. For example:

```
dska_16:  PV-rpv                of LV-root
         #Seeks AveSeek Queue-wait Channel-wait  Queued Multiplier
PageRd 1006892  143.48       80.0 11.5%   43.2       0       99.8
PageWt  322489  144.87      199.7  3.7%   43.3       0    50000.0
VtocRd   88877  171.56       47.1  0.9%   36.7       0       23.8
VtocWt   53563  107.83       50.6  0.4%   27.5       0       54.5
TEST      0 UNLOADs, 213 TESTs
dska_16 Queue:Ave 2.5,Alloc 650556,Max Depth 200/2720,Cur Depth 0
  Channels 16.41% busy,  4253 Combs,     3.9 IO/second.
```

Simple Drive Information

If detailed drive information is not selected, disk_meters prints a different and simplified output. These outputs are not really equivalent in information content. The simplified output indicates read and write counts, average seek length for the drive, *average time between (ATB)* reads and writes in milliseconds, and average time between I/O for the drive. For 500 and 501 style drives, with a primary and secondary physical volume per spindle, the primary physical volume has the total drive ATB figure. For example:

Drive	PV	Reads	Writes	Ave Seek	ATB Reads	ATB Writes	ATB I/O
dska_01	old_dumps	5523	8272	8	32770	21880	13120
dska_02	root_b	58148	16258	89	3112	11132	2432
dska_03	root_c	47534	9226	49	3807	19617	3188
dska_04	root_a	57184	14423	64	3165	12548	2527
dska_05	p11	11017	10308	11	16428	17558	8487
dska_06	rpv	57417	20301	84	3152	8915	2328
dski_01	pub_a	88237	19220	89	2051	9416	945
dski_02	list_1	75341	8686	36	2402	20837	
dski_03	pub_b	88182	19436	93	2052	9312	961
dski_04	list_2	72532	8131	38	2495	22259	

Channel Information

The channel information block follows the drive information for each subsystem, if selected. It indicates for each channel the number of connects issued, the number of I/Os which were completed while masked, the number of interrupts which did not have a terminate status, the number of times an interrupt occurred for completed I/O, the number of times a termination interrupt was received for a channel which was no longer active, and the current status of the channel. Status information indicates if the channel is currently broken, assigned to IOI, or inoperative. For example:

```
dskc Channel Information
             Term by  Interrupt   get_io  Term w/o
   Connects    RUN    w/o term   w/o term  Active   Status
A28 3150500     1046       1024          6        7
B28 1000147      530        505          3        3
A29  251978      142        139          1        1
B29   25634       22         22
A30     972        2          2
B30       1
A31      27
```

Disk Tuning -- the tune_disk Command

The Multics disk DIM provides a tuning tool with which you can adjust most of the operating parameters of the optimization algorithm. This is the tune_disk command. It is described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No. GB64.

IS THERE A PROBLEM?

The metering values described earlier provide a view into the workings of the various mechanisms which make up the Multics system. They provide most of the information needed to determine if a problem really does exist, and they can guide you to the problem areas.

The most important value to look at when you're trying to determine if there's a problem is the *virtual CPU time* being delivered to the users. If this is high, nothing can be gained by tuning. For example, if a system is running approximately an average load, and is receiving a virtual CPU time of 90%, then there is probably very little to be gained by tuning, since so little is lost to overheads of any kind.

It is important at this point to do a little *windage* calculation to determine if apparent poor response or delivery is simply due to high user loading. If the system is providing a high virtual CPU time delivery, but there are a large number of users, then the formula:

```
Per User = Processors * Virtual CPU Time / Users
```

indicates roughly what fraction of a processor each user is getting. If it is not enough, the only solutions are to add more processors or run fewer users in parallel. One way of doing the latter is to defer the absentee load to a less used time period. It is advantageous to distribute system loading as evenly across the 24 hour day as possible, given the constraints of major interactive delivery during the *working day* time period.

What is the Source of the Problem?

If there is insufficient *virtual CPU time* delivery to users, the next step in handling a problem is to determine what the *idle* and *overhead* percentages are. If a system is running with very low *idle* times, then tuning will be aimed at the reduction of *overhead* costs of operation.

1. *High Idle Time*
   High idle times are the mark of a system which doesn't have enough work to do, or is incapable of getting the work into memory efficiently.

2. *High Overheads*
   High overheads are indicative of high system loading situations, and typically indicate too much work for the available memory space, rather than poorly tuned hardware. (Allocation lock overheads can be an exception, and can be handled by tuning.)

3. *Paging Activity*
   Paging activity essentially occurs when required data isn't currently available in real memory. There are a myriad of causes for this problem. The obvious cause is insufficient real memory for the current parallel system load.

What are the Characteristics of the Problem?

1. *High Idle*

   Zero Idle
   If the system has a high zero idle it means that no work is schedulable. If this is due to no work available at all, then you don't have a problem. If this is due to absentees being held back for extra shifts, then a policy decision may be needed to schedule low priority absentee jobs to fill a determined void. This solution can be verified by checking absentee queue loadings against system activity. If there is a lot of possible pending work while nothing is being done, then you may wish to initiate selected processes.

NMP Idle

If the system has a high NMP idle, many of the problems situations seen with a high zero idle may be present, and may be corrected in the same ways. However, the paybacks will typically be less since there is already work being done, leaving less idle for recovery.

Note that, on a one processor system, NMP idle will be seen instead of MP idle. It can be indicative of higher paging rates, and should be handled like MP idle.

Loading Idle

If loading idle is significant, then the only solutions will probably be introducing additional real memory or increasing process time slices. For loading idle to be high, the system must be thrashing a great deal and MP or NMP idle are probably considerably higher than loading idle.

MP Idle

The major cause of MP idle is an insufficient multiprogramming level for the current I/O delays. You may attempt to correct this by increasing multiprogramming, if it really is too low, or decreasing I/O delays.

MP idle can occur at low loading levels because there are not enough processes to cover normal page fault delays. However, you can usually recognize such a situation by good system responsiveness, low paging and interrupt overheads, and low disk I/O rates.

2. *High Overheads*

Paging Overheads

Paging overheads are typically in the range of 10% or less. Approaching or exceeding 10%, even on a busy system, probably indicates a resource bottleneck in disk queuing. This can be directly seen as allocation locks in output from the disk_meters command, which will also indicate the processor use in the allocation locks. The simple corrective action is to configure sufficient free queue elements via the config deck.

Interrupt Overheads

Interrupt overheads are a direct indication of actual work being done. To deal with them, you have to remove extraneous work by tuning system use or by adding more memory to the system. However, there is one reason for excessive interrupt overhead which is amenable to disk tuning and can be relatively easily identified. It is described next.

3. *Paging Activity*

Each user on a system requires a certain number of pages of his programs and data to be in memory to work efficiently. These pages are termed the *working set*. The more users in parallel on a system, the higher the system working set. If the system doesn't have enough available memory for the working sets of the number of users processing, the system will start to thrash. Thrashing is a positive feedback and only tends to get worse unless the cause is alleviated.

What is the Scale of the Problem?

Typically, you can't remove all the overheads or even all the idle present in any system, and as the parallel loading of the system increases, there will be unavoidable overheads and costs associated with supporting it. If you are not experiencing allocation locks, or highly skewed drive or PDIR loadings, then you would be very unlikely to get back even 25% of any overheads by tuning the hardware.

If that scale of recovery is not sufficient, then you are probably faced with necessary hardware expansion, or a review of the routines in use.

Hardware solution scaling can be determined through a number of methods. When dealing with situations of high system paging with commensurate poor response and low memory lap times, you can do well by trading 1MW for two 501 disk drives. There will be a very noticable system pickup.

Solution scaling is highly tied to existing load levels and user desires. It is difficult, without performing a study of existing loading and user characteristics, to offer much advice in the way of constructive hardware scaling.

What Methods can be Used to Resolve the Problem?

1. *High Idle Time*

   If your site is experiencing high idle times, you should attempt to shift work into periods of high zero and NMP idle, and shift work out of periods of heavy MP idle (as long as the idle is related to overwork). This primarily means shifting absentees, which were intended to be deferred anyway.

   MP idle for high load situations will typically be accompanied by a reasonably high paging and interrupt overhead. You may well see noticable allocation locks occurring, indicating disk bottlenecking. This bottlenecking can be determined by using the disk_meters command to look at queue lengths and averages, and drive and channel busy indications. They may well indicate too tight an allocation of process directory drives. This will show up as a small number of PDIR drives which all have very high I/O rates and appreciable queues. This can be resolved by spreading PDIRs through a wider range of drives and/or putting PDIR drives on otherwise low activity volumes.

   If there are too few PDIR drives, you will probably find a very noticable performance knee at certain load levels. This can be directly attributed to I/O saturation of the PDIR drives.

   If PDIR drives are not being run at more than 60% busy or so, you may find that the burst characteristics of page replacement are causing allocation locks. The resolution of this is very easy: simply configure more free queue elements via the parm config card's dskq parameter. The wide range, 5 to 200 elements per drive, permits sufficient headroom for the free queue. Remember that one queue element is roughly equivalent to 1024 words of memory waiting for I/O. Configuring more queue elements than page frames doesn't buy anything. At most, you probably don't need to do more than about three pages for each queue element, which is probably still high.

A system which has high I/O rates evenly spread across drives may not be able to recover MP idle simply because of the inherent high I/O delays associated with doing a lot of I/O. You will recover the overhead and I/O loading only by adding more real memory or lowering the parallel system load.

2. *High Overheads*

Interrupt Overheads
If the interrupt_meters command indicates that the individual interrupt time for a disk channel is high, probably two milliseconds or more, or disk channels on one subsystem seem to be out of line with disk channels on other subsystems, you may be seeing queuing overheads. Such overheads occur if a subsystem is receiving a very high load and building appreciable queues. In such a case, the throughput of the requests is slaved to the physical throughput of the drives, but the interrupt overhead is a direct relationship to the amount of work required to determine the next I/O to service. A very large queue can require considerably more work and will be directly seen in the interrupt_meters output. You can probably correlate the drive use with the disk_queue meter and the disk_meters command, pinning it down to certain drives and then determining the origin of their loads. Probably it will be due to process directories. The obvious solution is to spread the PDIR activity across a wider range of queue servicers.

If you determine that drive activity is due to specific drive-related I/O uses, rather than PDIR activity, it may be necessary to tune the optimization response of the drive to better handle the characteristic loads presented. If average queue lengths are causing interrupt overheads, you may wish to drop the load point of the loaded I/O types to start to give optimization priorities at lower queue loadings, and hence lower average interrupt overheads. If there will always be a nearly even mix of read and write on the drive, you may wish to drop the response point multiplier for writes to provide a more competitive position and a lower slope to the optimization line. This removes sudden optimization knee effects seen with low queue load points and large optimizing line slopes.

This process will be a combination of trial and error, but will probably be isolated to only one or two drives and thus will not produce cross-drive effects.

3. *Paging Activity*
The Multics disk DIM tends to reduce thrashing to some extent through the alteration of queue priorities and request service ordering. As queues build, the longer queue I/O types gain servicing priority and increase the service times of the blocking I/Os. This in turn tends to throttle back the processes causing the thrashing so they are more in line with the modified pages they are producing. Since these processes are probably drive-localized to certain data sets, the rest of the system continues with good service and the affected processes slow down.

The major ways of alleviating thrashing are to increase the real memory on the system and/or to decrease the number of processes with a parallel demand of memory resources. One obvious method of doing the latter is to defer absentee processing to an offpeak period. Certainly if a system is saturated during the prime shift and has zero idle periods in other shifts, absentee deferral is the obvious choice. However, policy and politics will tend to govern this solution.

If a site determines that none of these methods are applicable, then a review of software systems is required to determine systems which need optimization. Look first at the use of common libraries. Ensure that they are compiled optimized. Then look at the code within the systems themselves.

WHAT DO THE NUMBERS MEAN?

It can be a very interesting exercise to attempt to determine the origin of various meter values and what causes them to be the way they are. The following paragraphs deal with some of the more obvious meter values for the disk meters.

Average Free Queue Depth

The average depth of any queue is determined by summing the current depth at the time a request is being queued. The long term average will then be an indication of typical loading. However, since this is not a time-averaged value, it cannot be used to determine the average I/O load. That can only be determined by the I/O rate values produced by the disk_meters command, or by interrupt counts.

If a queue has a high average depth but a low average activity, as determined by channel busy values, it is indicative of deep burst activity, but infrequent requests. It probably means that the drive still retains acceptable I/O throughput headroom and is only lightly utilized. It probably is indicative of a large memory size for typical system loading, in which the burst activity occurs because the light system loading lets a lot of modified pages accumulate prior to some action which causes their flushing. It is essentially normal. Even if allocation locks are sponsored by this activity, they will probably be a very small portion of available CPU time.

Maximum Queue Depth

This is an indication of peak I/O queue loadings for a drive. If it is very high, but the average queue length is quite low, or the number of I/Os is quite low, then this queue loading is probably the artifact of a memory flush operation to ensure pages have been flushed to disk. It will be a normal occurance.

However, if the peak load is high and the average load is high for a reasonably busy drive, it indicates a situation where the drive is regularly being used and being supplied with a very deep burst of write activity. This is probably indicative of insufficient PDIR spread or a specific application which generates these characteristics, like a Trouble Report database application. It is possible to tune a drive for these I/O characteristics. It's probably best to start by decreasing the load point for the drive's page write I/O type to sponsor optimization at lower queue loads. This will have the effect of decreased responsiveness to files allocated to that drive, but the rest of the system will pick up some of the interrupt overhead released by this action and will not suffer cross-drive effects.

Queue Wait Time

Queue wait times are indicative of the time that requests tend to sit waiting for service. Since they are averages, the deviation from the standard can get quite high for a very small sample of requests, particularly if you notice high maximum queue depths for the drive. Such cases will probably be the cause of any combings which you see. A small percentage of the requests will cause combing, sponsored by memory flushes in the idler off-shift time periods. Such flushes can sponsor quite deep queue bursts to individual drives. But this is a normal system action and occurs only at a time when it makes little difference.

You should see that the lowest priority requests, page writes, tend to have the higher queue waits. If this is not true, and there are a lot of requests processed, then you probably got the priorities backwards.

Channel Wait Times

Channel wait times are indicative of the amount of time it takes for a request to be passed to the MPC, for the MPC to complete the request, and for the DIM to find this out. You might notice high channel times for VTOC writes on MSU0501 disk drives. This is due to the necessary read and rewrite needed to do 64-word sector I/O on these drives. MSU0501 drives are formatted in 512 word sectors, rather than 64 word sectors, to put more information on the formatted drive.

The MPC handling the 501 drive has two 512 word internal buffers which it utilizes for *simulating* 64 word I/O. The MPC recognizes the presence of a write which will not completely fill a 512 word sector. The MPC primes one, or both, of its buffers by reading information from the drive. It then moves data from main memory into the buffer, and re-writes the buffer on the next rotation of the spindle. During this operation the MPC is completely tied up and does not handle any other processing, including seek overlap. Thus, it is not unusual to see channel wait times for sector re-writes as high as 50 milliseconds.

These rewrites will also stop any other I/O through that MPC until they are complete. Given normal VTOC write rates, this is not really a problem. However, AST thrashing from too small an AST pool could increase the degree of this problem.

Since primary and secondary volumes on a spindle share the same head actuator, the disk DIM utilizes only a single queue (the primary queue) and mixes requests for both volumes. These are optimized together to prevent head thrashing. Thus, support of primary and secondary volumes should produce no suprises.

Alloc counts

The allocation count for a queue is indicative of the number of times a request is seen when a drive is already busy. Thus, a comparison of the allocation count for a drive queue with the I/O counts for the drive gives a good indication of the number of requests which are processed when the drive is essentially idle. Most sites will be suprised to see just how idle most of their drives really are. Only I/Os which are counted in the allocation count are also counted for max_depth and average queue length, since no other requests even get queued and immediately acted upon.

Channel Busy

The channel busy numbers for each I/O type, and for the drive as a whole, give an idea of how much a drive is being used and how. Remember that the sum of all busy drives on a subsystem cannot usually be more than about 80% times the number of channels. So even if a drive is not running in the high 90% values, it may have no more to give, at least on that subsystem.

You should also remember that there is a physical load and overhead within the MPC for active drives. A site can overload their MPCs and not get full delivery from the drives. It may be time to bring in additional MPCs and split up subsystems for better physical throughput. One value to look for is the number of transfer synchronization losses reported in the MPC statistics. If you have a number of transfers which lost synchronization and had to be retried, then you are probably overloading either MPC or IOM and hardware expansion may be necessary. (Look to adding memory, not drives; this avoids the I/O in the first place.)

Combing

The combing number indicates the number of disk I/Os completed using the disk combing algorithm on a drive. It is indicative of an extensive queue buildup or abnormal delays, like a drive offline. A really busy drive can probably do a continuous delivery in the range of 25 to 30 I/Os per second, meaning that a five second stagnation equates to at least 125 disk I/Os done continuously without letup, with one of them not getting serviced for the five seconds. So if you see a high number of disk combs and reasonable max_depth, you must be having a large number of deep bursts, rather than infrequent enormous bursts. This may be indicative of a large memory system running nearly idle, and only seeing page flushes, or it may be a danger sign that a drive is being overloaded.

write_limit

Though it has not been mentioned as a tuning factor for disks, page flushing has been mentioned as a factor in the generation of deep bursts of disk activity. The write_limit tuning parameter serves to control the size of memory flushes.

Conclusions

A site can adjust tuning parameters on a very detailed level, but these parameters are fairly general with a broad and smooth tuning response. System default values are based on the number of free queue elements available and rule-of-thumb priority defaults.

A site can adjust the size of the queue buffering resource and remove potential bottleneck areas to recover system overheads, but general system default queue allocation on the basis of the number of drives will probably be sufficiently generous to remove problems. Sites with a small number of very active drives will probably wish to set larger queues for their configurations.

The Multics disk DIM is aimed mainly at the busy site, which finds itself low on resources and suffers from disk overloading problems during peak periods. The disk DIM extends system delivery and responsiveness, without penalizing low and medium disk load sites.

INDEX

B

C

E

F

G

I

K

L

N

T

W

X

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: MULTICS SYSTEM MAINTENANCE PROCEDURES

ORDER NO. AM81-04

DATED NOVEMBER 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

PLEASE FILL IN COMPLETE ADDRESS BELOW.

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

CUT ALONG LINE

FOLD ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U.S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA 02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
Mexico: Av. Constituyentes 900, 11950 Mexico, D.F. Mexico
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

46833, 1186, Printed in U.S.A.

AM81-04