The Adesse Corporation

VM System Product

CP INTERNALS

Richard Alexander

Larry Chace

Bob Cowles

INTRODUCTION

This manual is a companion to the Adesse Corporation's VM System Product CP Internals course. The course has been developed by practicing system programmers and is intended for system programmers who have a basic familiarity with VM/SP. We hope this document strikes the right balance between two competing needs: on the one hand we would like to present the material such that a beginning programmer could benefit from it, and on the other hand we would like to be of help to the veteran programmer. It is clearly impossible for a manual of this size to completely address both kinds of needs.

The first three chapters concentrate on an overview of the hardware and software architectures exploited by CP. A basic understanding of the total system architecture will help you as we then move on to the more specific topics that follow. Most of the manual is concerned with detailed descriptions of the major parts of CP. The control blocks, the modules, and the algorithms are covered to a degree of detail that we hope is appropriate in each case. The final part of the manual covers topics that are more general in nature; that is, they involve several of the previous detailed topics.

You should be familiar with IBM System/370 architecture. We assume that concepts such as DAT, I/O interrupts, and DASD are familiar to you, although we do review such concepts in the first chapter. We also assume that you are basically familiar with the workings of an interrupt-driven timeshared operating system.

We hope that this book will be of value to you. The specific information is based upon VM/SP Release 2 and in many cases has been updated to release 3 with HPO. As with any subject as complex as CP, mistakes are unavoidable; we would be appreciative if you would report any errors to the Adesse Corporation at the following address:

The Adesse Corporation
Post Office Box 607
Ridgefield, Connecticut 06877
Tel: 203-431-3071

May 31, 1985

CONTENTS

PART II -- Specific Topics

Chapter *page*

LIST OF TABLES

LIST OF FIGURES

PART I

GENERAL ARCHITECTURE

The first part of this course concentrates on architecture, both hardware and software. A basic understanding of the general system architecture will help you as we move on to more specific topics.

Chapter 1

SYSTEM/370 ARCHITECTURE

1.1 INTRODUCTION

This chapter will give you a quick review of the architecture of System/370; we will emphasize those features that are of particular interest to a CP system programmer.

1.1.1 Overview

System/370 is an extension of the original System/360 design. The system consists of the following components:

1. Main storage (also called "storage" or "core").
2. Central processing unit ("CPU").
3. Input/output facilities.
4. External interrupt sources.
5. System console.

The next chapter describes how CP divides these real components so that each VM user sees a small replica of a complete System/370.

1.1.2 References

Several documents provide a description of the System/370 architecture:

1. *IBM System/370 Principles of Operation* (GA22-7000) is the official public description of the hardware, as seen from the point of view of a program.
2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203) describes some simulated additional instructions invented by CP.

3. "System/370 Reference Summary" (GX20-1850, also known as the "green card" or "yellow booklet") is a highly concentrated quick guide to many of the hardware features and is invaluable for system programmers.

1.2 MAIN STORAGE

System/370 main storage (also called "real memory") is a contiguous array of memory elements in which programs and data may reside. The memory is also used by the I/O subsystem to contain commands and buffer areas.

1.2.1 Addressing

Main storage is addressed in elements of 8 binary digits ("bits") each; each such element is called a "byte". Each byte is assigned a number, or "address", starting with 0 for the first byte and increasing by 1 for each additional byte. Since each byte contains 8 bits, numeric contents of a byte may range from 0 to 255. For ease of notation, the contents of a byte are usually given as a 2 digit hexadecimal value in the range from X'00' to X'FF'; each digit represents 4 bits and is sometimes called a "nibble". Bytes may be combined as follows to allow the storage of larger values:

1. A "halfword" consists of 2 adjacent bytes starting at an address that is a multiple of 2. As an unsigned decimal number, a halfword's range is from 0 to 65535.

2. A "word" consists of 2 halfwords (or 4 bytes) starting at an address that is a multiple of 4. As an unsigned decimal number, a word's range is from 0 to 4294967295.

3. A "doubleword" consists of 2 words (or 8 bytes) starting at an address that is a multiple of 8. As an unsigned decimal number, a doubleword's range is from 0 to 18446744073703551615, but it is rarely used in that format.

The number of bytes of main storage is usually given in terms of "K" (1024) or "M" (1024K or 1048576). The real storage size may range from 240K to 16M for various sizes of System/370 machines. (Some main storages are now being built with more than 16M, but they require special addressing that we will describe in a later section.)

In most cases, memory addresses are represented as 3-byte values, usually right-adjusted within a word. Such an address has a range from 0 to 16777215, which exactly covers the 16M maximum main storage size. In order to keep instructions to a reasonable size, however, the entire 3-byte address is not used directly. Instead, a 12-bit displacement and a 4-bit register specification are used in instructions; the 12 bit value is added to the contents of the selected register to form the total 24-bit address. The register is called a "base register", since it contains the base address of a 4K block of memory; the block can start at any address.

1.2.2 Protection

System/370 provides a means by which portions of main memory can be protected from undesired writing or reading; this is called memory protection and is implemented by having a special 7-bit memory element associated with each 2K section of main storage. These "protect keys" are not an addressable part of main memory but instead are kept in an independent storage area. The protection applies equally to data and to instructions. The key is formatted as follows:

1. Bits 0 through 3 represent a code number. The only programs that can utilize the associated memory area are those whose codes match the key's code or whose codes are 0. A program key of 0 is considered to "match" with any memory protection code. Therefore, there are 15 "user" protect key values (1 - 15) and one "system" protect key value (0).

2. Bit 4, if set, specifies that fetching (reading) from the associated memory area is to be under control of the protect key code value. If this bit is set, then the memory area is said to be "fetch protected"; that is, it cannot be read by other users. If this bit is zero, then any user is allowed to read from the memory area, since the protect key codes are not checked in this case.

3. Bit 5 is set whenever any part of the memory area is referenced for storing or fetching of instructions or data; the reference may be caused by the CPU or by an I/O channel. This bit can be used to detect which parts of main memory are being heavily used or rarely used.

4. Bit 6 is set whenever any part of the memory area is changed, or, more precisely, is the target of a store operation. The store may be caused by the CPU or by

an I/O channel. (No comparison is made to see if the new value is different from the old value, as might be implied by the term "changed".) This bit can be used to determine whether or not the contents of the memory area must be saved if the area must be taken over temporarily for some other purpose.

1.3 CPU

The central processing unit consists of an arithmetic and logical processing unit (which we will not discuss here), various sets of registers, a program status word, and an address translator.

1.3.1 Registers

The CPU contains a number of registers that are used in the execution of the various instructions in a program. Two different sets of registers are of particular interest to CP, the general purpose registers and the control registers.

1.3.1.1 General purpose registers

The 16 general purpose registers are each 4 bytes in length and usually contain either storage addresses or operands for arithmetic and logical operations. Several registers (0, 1, and 2) have special meanings to the hardware in certain contexts:

1. Register 0, when used as an address, always has the effective value 0, no matter what is actually in the register, so that it may be used to address the first 4K area of storage. Since no real base register is needed for the first 4K, that area is usually reserved by programming systems for special use.

2. Registers 1 and 2 receive results from the TRT (translate and test) instruction. Programmers must therefore be careful when using TRT.

1.3.1.2 Control registers

The control registers are a second set of 16 registers that control special functions; they are not generally available to users' programs. The control registers are used only in

the Load Control (LCTL) and Store Control (STCTL) instructions; no other instructions make explicit use of the control registers. The most important of the control registers are:

1. CR0 contains individual bits that control various options such as block multiplexing, dynamic address translation, and external interruptions.

2. CR1 contains a pointer to the segment table used by dynamic address translation.

3. CR2 contains individual I/O channel interrupt mask bits.

4. CR6 contains flags and a pointer used for the microprogrammed assists that improve the speed of certain CP functions.

5. CR9, CR10, and CR11 are used to control "program event recording", which can be used to aid in program debugging.

6. CR14 and CR15 are used to control hardware error recovery.

1.3.2 Program status word

The program status word (PSW) contains the master control information. The PSW has two formats, basic control mode (BC mode) and extended control mode (EC mode), which are selected according to bit 12 within the PSW.

1.3.2.1 Basic control mode

The System/370 starts execution in BC mode; this is the mode in which System/360 machines always run. Figure 1 shows the format of the BC mode PSW, which is as follows:

1. Bits 0-7 are the "system mask"; bits 0-5 enable interrupts from channels 0 through 5, bit 6 enables interrupts from all other channels, and bit 7 enables "external" interrupts, as described in later sections.

2. Bits 8-11 form the protection key code that is used in conjunction with the memory protection keys already described.

3. Bits 12-15 are various control bits, which are often labelled BMWP. Bit 12 specifies the mode of the PSW (BC or EC), and must be 0 for BC mode. Bit 13 allows machine check (error) interrupts to take place. Bit 14 specifies that the CPU is in a "wait" state; that is, it is doing nothing at all. Bit 15 specifies that the CPU is in "problem" state; that is, no "supervisor" state instructions are allowed to execute.

4. Bits 16-31 contain the last interrupt code, which will be described in a later section.

5. Bits 32-33 contain the length of the last instruction that was executed.

6. Bits 34-35 contain the last "condition code" that was set by an arithmetic or logical instruction. This code may then be used by a "branch on condition" instruction to modify the flow of the program.

7. Bits 36-39 contain the "program mask", which enables interruptions for various abnormal arithmetic conditions such as overflow.

8. Bits 40-63 contain the address of the next instruction to be executed. This field is often referred to as the "instruction counter" or IC.

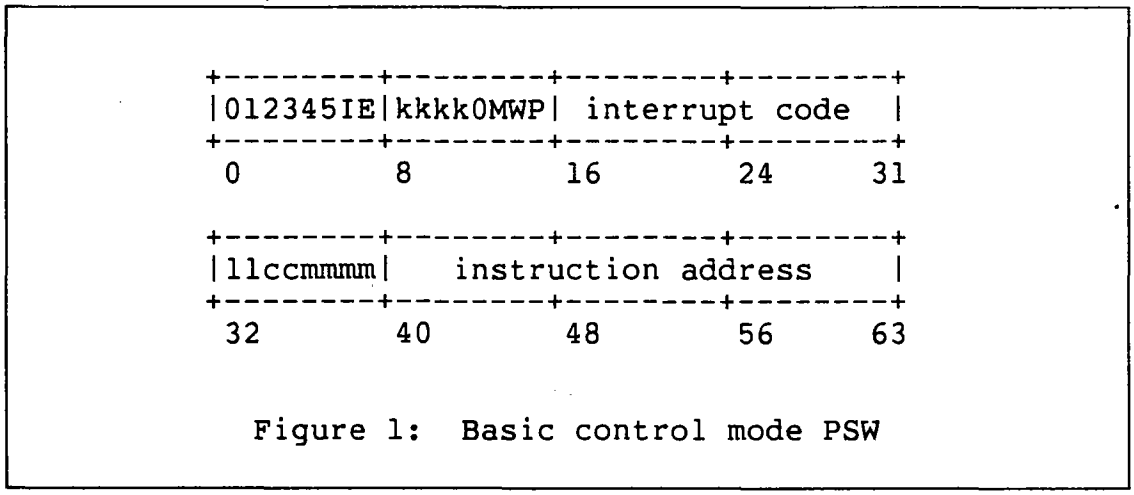


Figure 1: Basic control mode PSW

1.3.2.2 Extended control mode

If bit 12 is on, then the PSW is interpreted somewhat differently. System/370 introduced EC mode to allow several

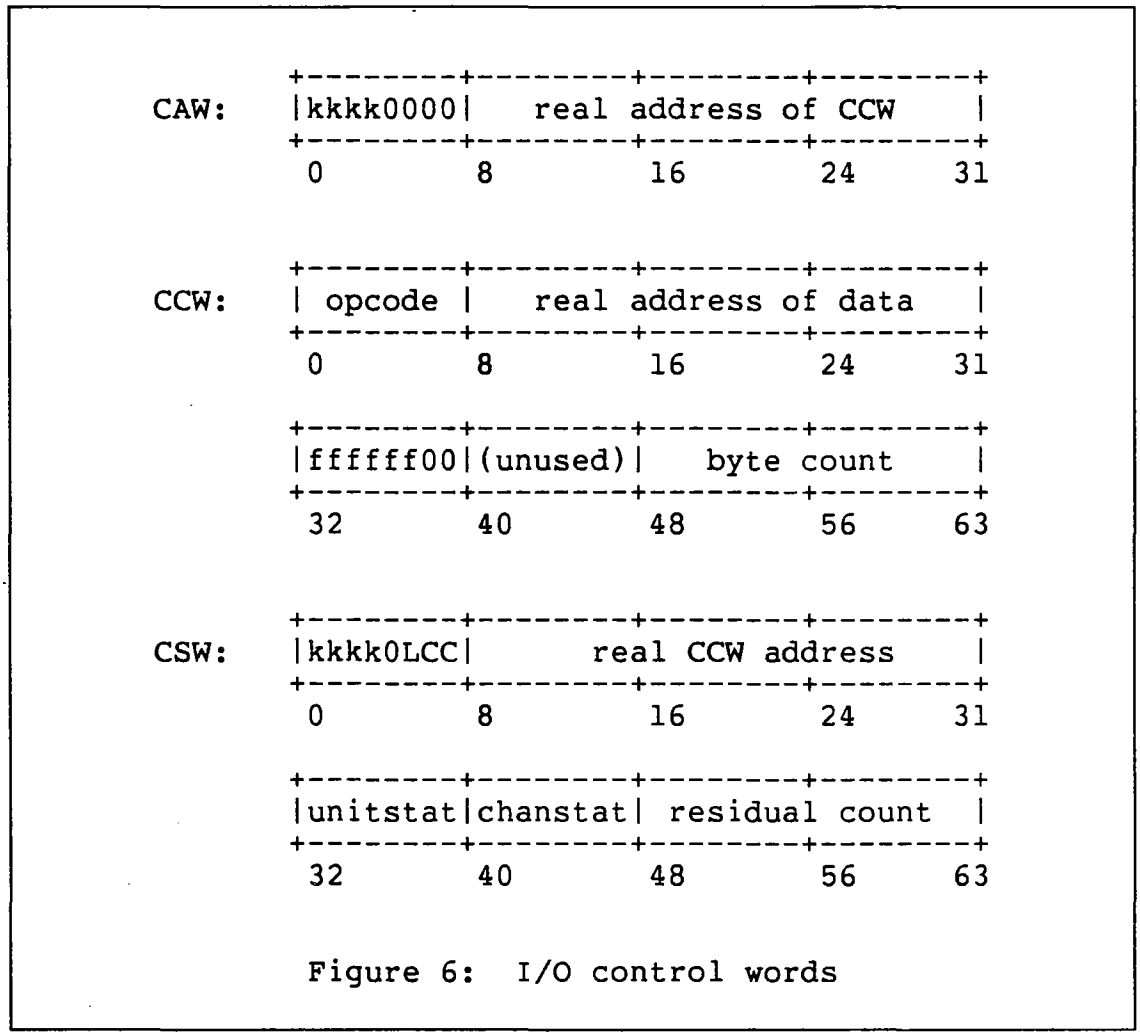


Figure 6: I/O control words

1.4.5 I/O instructions

Several CPU instructions are available for performing and controlling I/O operations. These instructions are all limited to supervisor state and are usually issued only when the CPU is disabled for I/O interrupts.

1. SIO: The Start I/O instruction is used to begin the channel program pointed to by the channel address word. If certain error conditions exist, then the condition code will reflect the errors. Otherwise, the channel takes over the I/O operation and the CPU is free to continue with the next instruction.

2. TIO: The Test I/O instruction can be used to examine the status of an I/O device. A TIO loop is sometimes used as an alternative way of waiting until an I/O

operation is complete since the TIO will clear a pending I/O interrupt.

3. HIO and HDV: The Halt I/O and Halt Device instructions can be used to stop a channel program before it finishes normally. Although this is not commonly used, some types of terminal devices are supported by channel programs containing unending sequences that must then be halted for the next operation to continue.

1.5 EXTERNAL INTERRUPT SOURCES

External interrupts can be caused by the various system timers as well as by other facilities such as the operator's EXTERNAL button and the multiprocessor feature.

1.5.1 Timers

Several hardware timers are provided in System/370 to assist user and system programs. System/360 provided only a single timer, the interval timer described below; System/370 has added several others. Note that there is only one of each of these timers per CPU.

1.5.1.1 Interval timer

This timer consists of the word of main memory at address 80 (X'50'). Bits 0 through 23 of this word are decremented at a rate of 300 times per second; some CPUs offer higher resolution by decrementing more bits more often. When the contents go from zero to -1, an external interrupt is requested and stays pending until the interrupt is actually taken. This timer "wraps" after about 15.5 hours. Figure 7 shows the format of the interval timer.

1.5.1.2 CPU timer

The CPU timer is a special register that can be accessed only with the Store CPU Timer (STPT) and Set CPU Timer (SPT) instructions. The timer is a doubleword in which bit 51 is caused to decrement every microsecond when the CPU is running or waiting, but not when it is stopped (from the operator console). The exact rate of decrementing is such that the timer runs at least as fast as the fastest CPU instruc-

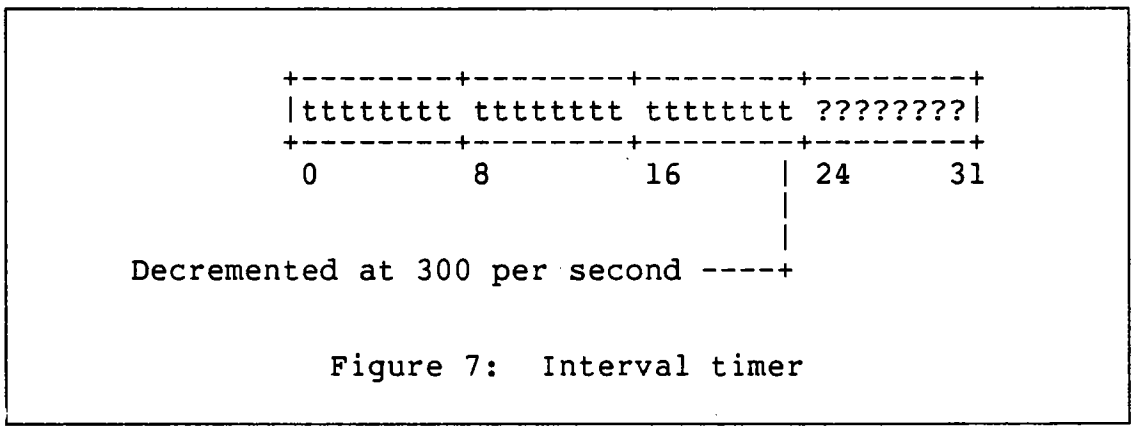


Figure 7: Interval timer

tion. Whenever the time is negative (bit 0 is one), a request for an external interrupt is made; the request stays pending until bit 0 is made 0. Figure 8 shows the format of the CPU timer.

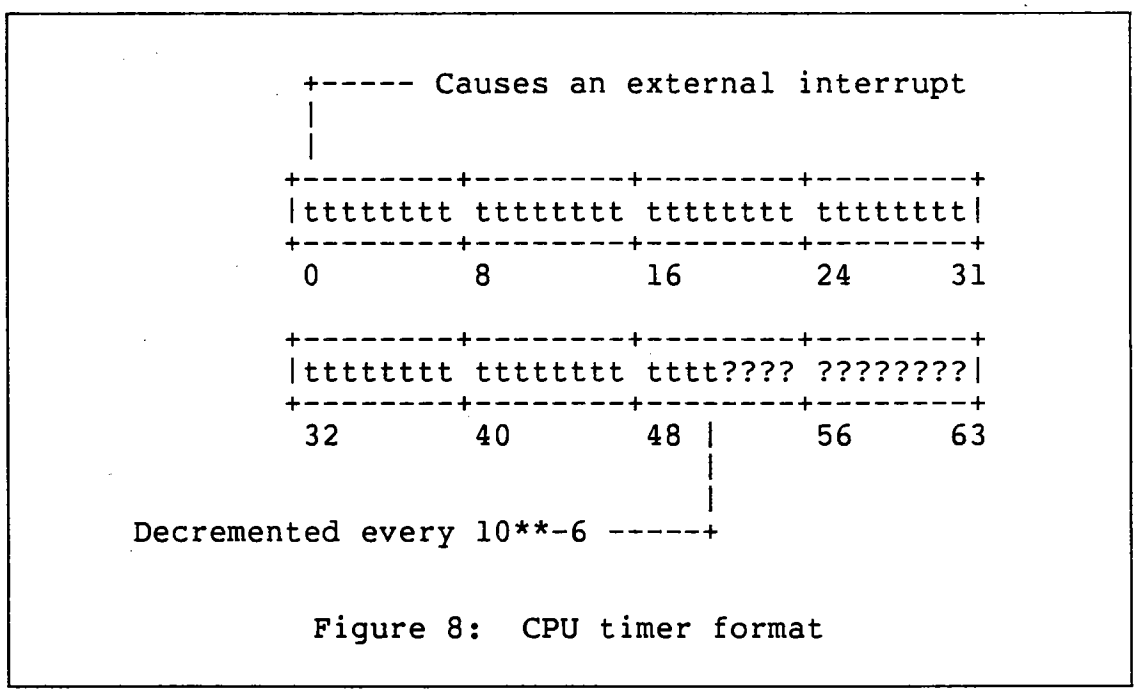


Figure 8: CPU timer format

1.5.1.3 TOD clock

The time of day (TOD) clock is also a special register; it can be accessed only by the Store Clock (STCK) and Set Clock

(SCK) instructions. The TOD clock is a doubleword in which bit 51 is caused to increment every microsecond; bit 31 (the last bit in the left-hand half of the doubleword) increments every 1.048576 seconds and is often used as an approximation to one second. This clock has a period of about 143 years and by convention the value 0 is taken to mean 0:00 a.m. Greenwich Mean Time on January 1, 1900. The TOD clock causes no interrupts. Note that the Store Clock (STCK) instruction is the only timer instruction that can be issued from problem state; all other timer instructions are valid only in supervisor state. STCK is also the only timer instruction whose operand can be on a non-doubleword boundary. Figure 9 shows the format of the TOD clock.

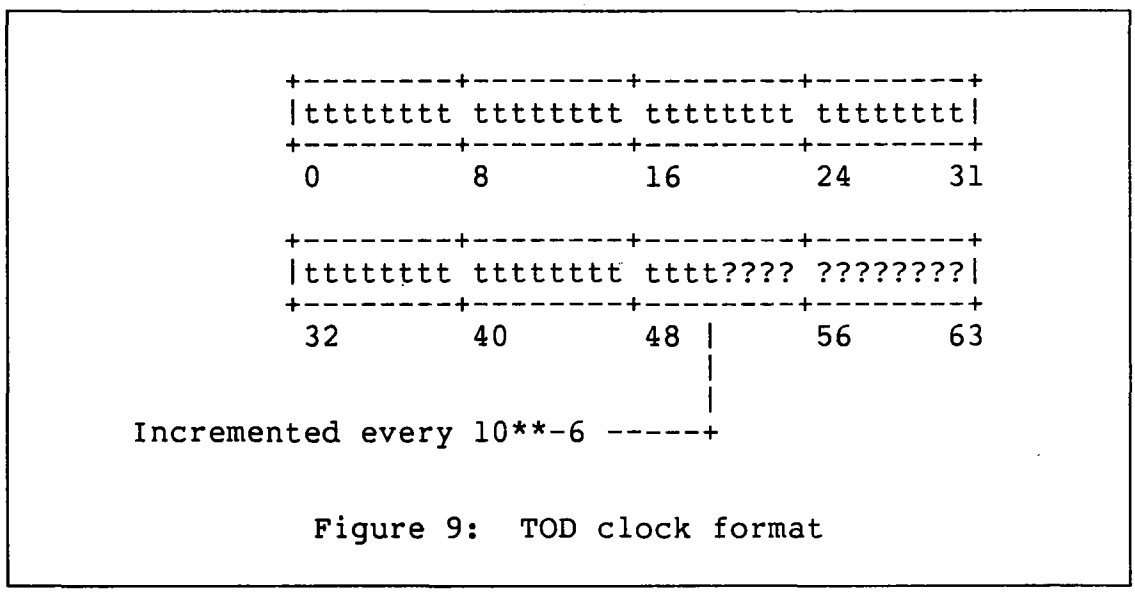


Figure 9: TOD clock format

1.5.1.4 Clock comparator

Associated with the TOD clock is a second special doubleword register, the clock comparator. Whenever the clock comparator contains a value that is less than the value in the TOD clock, an external interrupt is requested, and that request remains pending until the value is no longer less than the TOD clock. The clock comparator can be accessed only via the instructions Store Clock Comparator (STCKC) and Set Clock Comparator (SCKC).

1.5.2 Other external interrupt sources

There are several other sources for external interrupts. The two that are used commonly by CP are the following.

1. The INTERRUPT key on the system console causes an external interrupt to become pending. The interrupt stays pending until it is accepted or until a CPU reset is performed.

2. The multiprocessor facility provides for "malfunction alert", "emergency signal", and "external call" interrupts as a means of signalling between processors. In each case, an interrupt code is stored to further identify the type of interrupt.

1.6 SYSTEM CONSOLE

The *Principles of Operation* describes the system console in terms of "keys", which are set by the operator, and "indicators", which may be read by the operator. Some functions require both input and output and are therefore described as "controls". The exact hardware implementation of the system console depends upon the model of System/370. In some cases it will be made up of real switches and lights and in other cases it will be one or more menus on a CRT device. Some of the common console functions are the following.

1. Address compare controls -- provide a means of stopping the processor when it references a given storage address.

2. Configurator controls -- provide a means of altering various aspects such as main storage size and available I/O channels.

3. Display and enter controls -- provide a means of examining and altering the contents of memory, registers, and the PSW.

4. Interrupt key -- generate an external interrupt.

5. Load key -- read and execute a program, usually the initial phase of an operating system.

6. Load unit address controls -- specify the I/O device address for use with the load key.

7. Manual indicator -- show that the processor has stopped.

8. Power off key -- turn off the processor and attached I/O devices.

9. Power on key -- turn on the processor and attached I/O devices.

10. Rate control -- specify the speed at which the processor should run, usually full speed or 1 instruction at a time.

11. Restart key -- cause the PSW to be stored and a new PSW to be loaded.

12. Start key -- allow the processor to resume executing instructions.

13. Stop key -- cause the processor to halt after the current instruction.

14. Store status key -- cause the processor's registers to be stored into memory.

15. System reset key -- cause the processor and the I/O devices to reset to an initial condition.

16. Wait indicator -- show that the processor is in the wait state.

1.7 SUMMARY

This chapter has presented an overview of those aspects of System/370 that are of most interest to a CP system programmer. For a complete description, you should refer to the *Principles of Operation*.

NOTES

Chapter 2

VIRTUALIZATION

2.1 INTRODUCTION

2.1.1 Overview

The main function of CP is to divide the various components of the real System/370 such that each human user of VM becomes the operator of a virtual machine, a small replica of a complete System/370. Ideally, each replica is an exact functional equivalent of a real system, although some of its components may be smaller or slower; a program that executes correctly on the real system should execute correctly on the replica, within certain limits that we will describe in subsequent chapters.

2.1.2 References

1. *IBM System/370 Principles of Operation* (GA22-7000) is the official public description of the hardware, as seen from the point of view of a program.

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203) describes some simulated additional instructions invented by CP.

3. *IBM Virtual Machine/System Product: Introduction* (GC19-6200) gives an introduction to CP facilities.

2.2 PRINCIPLES OF VIRTUALIZATION

CP's virtualization of System/370 differs from a simulation in that as much of the real System/370 as possible is used for the replica. Pure simulation of the entire System/370 would make the replica run very slowly. By taking advantage of various features of System/370 architecture, CP can safely allow a virtual machine to use many of the real machine features. Only in certain cases does CP have to resort to simulation of a feature.

In order for any machine architecture to be virtualizable, it is necessary that important machine state changes can be trapped by an overall supervisory program (often call the "hypervisor"); this serves 2 main purposes. The first purpose is the protection of the hypervisor and of the other virtual machines. The running virtual machine must not be allowed to modify those portions of the real machine that are assigned for other use. The second purpose is the correct simulation of various functions. If real machine components (such as the general purpose registers) are currently assigned to the running virtual machine, then it should be allowed to use them in a normal fashion, and no trap should occur. Other facilities (such as the I/O subsystem), belong to the hypervisor; virtual machine references to I/O must be trapped and simulated in such a way that the trapped instructions will appear to have executed normally (although perhaps more slowly).

It is not sufficient that privileged instructions be trapped and simulated; various interrupt conditions must also be simulated. That requires the hypervisor to perform the PSW swap just as it would be performed by the real hardware, according to the state of the virtual machine.

An additional requirement for virtualization is that there be protection for any facilites that map virtual machine memory to real machine memory. In order to be protected, the memory mapping components should be located outside of the virtual machine memory; that usually means that the memory map is contained within the real machine memory space.

With very few exceptions, the components of System/370 are virtualizable in a fully functional manner. It is common practice, in fact, to run CP itself in a virtual machine for test purposes. That constitutes a good test of the accuracy of the virtual machine design.

2.3 VIRTUAL SYSTEM/370

In the following paragraphs we will discuss, in general terms, the processes by which CP virtualizes each of the 5 major components of System/370.

2.3.1 Virtual CPU

The virtual machine CPU can be broken down into 2 parts: instructions and interruptions. The state of the virtual machine "hardware" is kept by CP in various control blocks in areas of main storage reserved for CP's own use. In general, every register in the real CPU is replicated in a control block. When the virtual machine is actually running, many of its registers are located in the corresponding real registers.

2.3.1.1 Virtual instruction processing

You will recall that the running CPU is in either of 2 states: problem state or supervisor state. The System/370 instruction set has been so designed that the problem state instructions are "safe"; that is, their execution by a virtual machine cannot adversely affect any other virtual machine. CP, therefore, needs to do nothing to simulate these instructions; they run at full speed on the real System/370 hardware.

Supervisor state instructions, on the other hand, can affect other virtual machines since those instructions might alter the state of the I/O subsystem or important CPU control registers. CP must therefore intercept such instructions when they are about to be executed by a virtual machine. After interception, CP must simulate the instructions in such a way that the instruction appears to have executed, from the virtual machine's own point of view, and that other virtual machines are not adversely affected.

The interception is performed by a simple and efficient mechanism defined in the System/370 architecture. When the current PSW contains the "problem state" bit, then the execution of privileged instructions is not allowed but instead a "privileged instruction" program check interrupt is generated. CP receives that interrupt and can then proceed to simulate the instruction. At the end of the simulation, CP causes the virtual machine to resume execution with the next instruction and at full CPU speed.

For most instructions, then, no simulation at all is needed and the instructions are executed directly by the real CPU. For privileged instructions, CP routines simulate the instruction, thereby assuming the role of "hardware" for the virtual machine. At any instant in time, the real CPU corresponds almost exactly to the virtual machine CPU. Over longer periods of time, however, CP will have switched the real CPU from one virtual machine to another in a time-multiplexed fashion. The net effect is that each virtual ma-

chine appears to have a CPU that is slower than the real CPU; just how much slower is a function of system load and the scheduling algorithms.

2.3.1.2 Virtual program interruptions

In some cases, instructions are not simulated. For example, if the virtual machine is supposed to be in problem state and it nevertheless tries to issue the SIO instruction, then CP cannot simulate the instruction because the instruction would not have been allowed on a real System/370 running in problem state. Instead, CP must do what the real hardware would have done; it must generate an interrupt for the virtual machine. In a similar fashion, if a virtual machine program issues an SVC instruction, then CP must generate a virtual SVC interrupt.

That process (called "interrupt reflection") consists of simulating the PSW swapping procedure that occurs in the real hardware. The virtual PSW is stored into the virtual machine's old PSW location and a new virtual PSW is fetched from the virtual machine's new PSW location. Thus, the program check interrupt handler in the virtual machine gets control to process the attempt to execute a supervisor state instruction from problem state or the SVC interrupt handler in the virtual machine gets control to handle the SVC instruction.

CP takes this action whenever it must give an interrupt to the virtual machine. CP simulates the PSW swapping process that the hardware would have followed. This applies to all classes of interrupts, program check, SVC, I/O, and external; machine-check interrupts, however, are not usually reflected to the virtual machine.

2.3.2 Virtual memory

Since the virtual CPU actually runs on the real CPU, it then follows that the virtual machine's memory actually resides in the real memory. Similarly, just as CP divides the real CPU among the various virtual machines, it also divides the real memory among them. By using the dynamic address translation (DAT) feature of the real System/370, CP is able to accomplish two things: (1) it can scatter a given virtual machine's memory throughout available real memory, and (2) it can allow portions of the virtual machine's memory to be absent from real memory when not being used. The result is that real memory contains parts of several virtual machine memories at any one time; everything is kept sorted out by a combination of DAT hardware and software control blocks.

When a virtual machine references a non-resident portion of its memory, then an interruption is generated by hardware; this is either a page exception or a segment exception. CP must fetch into main storage the current copy of the referenced page, as stored on drum or disk, and then adjust the DAT hardware and software control blocks so that the virtual machine can re-execute the interrupted instruction and proceed from there. The process of fetching a page might require that some other page be written out to drum or disk in order to make available a place into which to read the desired page. This entire process is invisible to the virtual machine's programs.

2.3.3 Virtual I/O

There are 3 kinds of virtual I/O devices: those that are in fact real I/O devices (ATTACHed or dedicated), those that are a portion of a real device (minidisks), and those that are merely simulated. System/370 I/O instructions are privileged instructions and will therefore be intercepted by CP when they are issued by a virtual machine. In all cases, CP must examine and re-build the I/O channel programs to protect itself and the other virtual machines. For simulated I/O devices, various CP routines will perform the necessary operations such that the virtual machine program behaves as if the I/O device were real. For example, if a simulated card reader is being read, then CP will move the "card" image into the virtual machine memory location specified in the READ channel command word. For all 3 types of virtual I/O devices, CP must also reflect I/O interrupts to the virtual machine to indicate virtual channel program completion.

2.3.4 Virtual external operations

Some of the external operations, such as the various timers, are simulated by CP in a manner analogous to CPU and memory; CP uses the corresponding real System/370 facilities for each virtual machine as it is being run. As a result, the timers' contents are switched very often, and CP must use various control blocks to save the timer values for the non-running virtual machines. When a virtual timer would generate an external interrupt, then CP must reflect that interrupt to the virtual machine.

Some of the System/370 timers can be referenced only by means of privileged instructions, and so those timer references can be intercepted by CP. The interval timer, however, cannot be intercepted, since it is just a virtual machine memory location. CP includes special routines that

attempt to simulate the interval timer, and there is also special microcoded support available on some real System/370 processors. The simulation is not perfect but it is adequate in most cases.

Other external operations have been defined for use only in a virtual machine environment. Facilities such as VMCF and IUCV allow virtual machines to communicate with each other according to a defined protocol. Note that these facilities are NOT defined in the *Principles of Operation* and are therefore purely software conventions. Indeed, much of the power of CP can be seen in its ability to simulate nonexistent "hardware" facilities.

2.3.5 Virtual system console

Each System/370 has a system console for control of the CPU. The *Principles of Operation* describes various control facilities, some of which are listed in table 1 below. On real System/370 configurations, the system console may be a panel of lights and switches or it may be a special mode of operation for the normal operator I/O device. CP simulates these facilities by means of commands that can be entered by the user (the virtual machine's operator). These are called "console function mode" commands because when they are issued the user's terminal is operating not as a virtual machine I/O device but rather as the virtual machine's system console.

TABLE 1

System console functions

Console function	*CP cmd or msg*
Address compare controls	ADSTOP and PER
Configurator controls	DEFINE and SET
Display and enter controls	DISPLAY and STORE
Interrupt key	EXTERNAL
Load key	IPL
Load unit address controls	IPL
Manual indicator	"CP Read" msg
Power off key	LOGOFF
Power on key	LOGON
Rate control	PER and TRACE
Restart key	RESTART
Start key	BEGIN
Stop key	#CP or "PA1" key
Store status key	STORE STATUS
System reset key	SYSTEM RESET
Wait indicator	"disabled wait" msg

2.4 CP RESOURCES

For all 5 of the components described above, CP provides simulation for the user virtual machines. In each case, however, CP itself must use corresponding facilities on the real System/370. CP routines when executing use the CPU and they must be resident in main memory. They must perform I/O as a result of either virtual machine I/O requests or CP's own processes such as paging and spooling. CP must use the timers to control the sharing of the CPU and to prepare accounting records.

All of these requirements mean that CP must administer resource allocation for itself as well as for virtual machines. In many of the following chapters we will point out the complexities that result from this dual usage.

2.5 SUMMARY

From the point of view of the real System/370 hardware, CP is just an operating system (that is, it is just a program). From the point of view of the virtual machine operating system, CP is part of the hardware, namely, that part that divides the real System/370 into many smaller private System/370 configurations. When you are looking at CP routines it is often helpful to remember that, from the point of view of the virtual machine, CP is hardware and must therefore behave according to the *Principles of Operation*.

NOTES

Chapter 3

CP ARCHITECTURE AND CONTROL BLOCKS

3.1 INTRODUCTION

3.1.1 Overview

This chapter should give you a general description of the structure of CP. First we will briefly introduce some control block structures, naming conventions, and programming conventions. We will then describe how CP manages each of the 5 main components of System/370, both the real components for itself and the virtual components for the users. We will make brief references to some of the major control blocks.

3.1.2 References

The following IBM publications are the basic documentation for CP, and they are all appropriate for this chapter. At the risk of being repetitious we will include with each following chapter the applicable subset from this list.

1. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

2. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

3. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

4. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic, Volume 1 (CP)* (LY24-5220).

5. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

3.2 GENERAL CONCEPTS

CP consists primarily of a single program that is loaded from a DASD system residence volume when the LOAD function is performed on the real machine. The program is made up of many small routines grouped together into modules, each of which can be assembled separately. In addition to its routines, CP also requires various control blocks, most of which are assigned to locations in real memory as needed. Appendix A in the back of this book gives a very short summary of each of the CP modules and appendix B gives a cross reference of many control blocks.

In this chapter it will be very difficult to avoid references to things that we have not yet discussed. In order to start, let us say that CP consists of a "dispatcher" routine that examines lists of data items representing work to be done. For each piece of work, the dispatcher passes control to an appropriate processing routine that ultimately returns control to the dispatcher. Some of the lists are found via pointer words located in the first page of real memory.

After that over-simplification, let us put the CP program logic aside for a moment and look at some conventions.

3.2.1 Control block structure

When CP needs to store data items in memory, it usually does so by defining an appropriate control block. The block of memory contains the data and also other information that can be used to identify the associated virtual machine or other similar data items. Since there are several different kinds of data and several different ways in which CP uses the data, CP uses several different control block structures. We will describe each control block when we encounter it, but the following generalizations should help you to visualize the different structures.

3.2.1.1 Linked lists

The linked list is a simple structure in which each element of the list points to the next element; this is the singly-linked list, as shown in Figure 10. A word is usually used to point to the first item in the list, and this word is called the "anchor". An optional second word may be used to point to the last item in the list; that makes it easier to add a new item to the list. Sometimes the list is referred to as a queue. (In each case, the numbers used below are arbitrary memory addresses.)

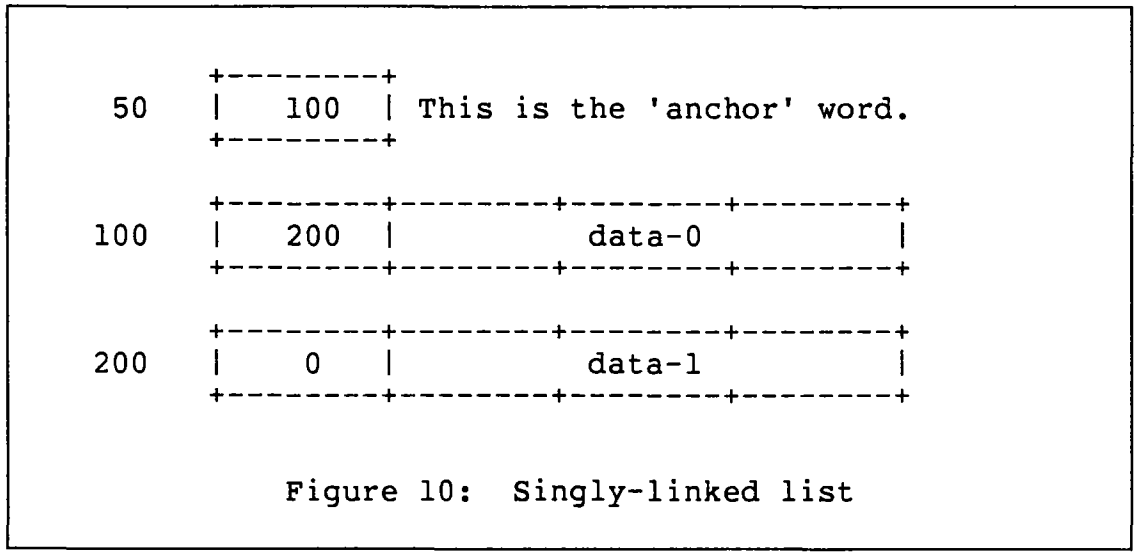


Figure 10: Singly-linked list

In many cases it is important to be able to move backward in the list or to easily add or remove items in the middle of the list. For that purpose a second set of pointers is used to indicate each item's predecessor. Figure 11 shows the doubly-linked list. Again, the second anchor word may or may not be present.

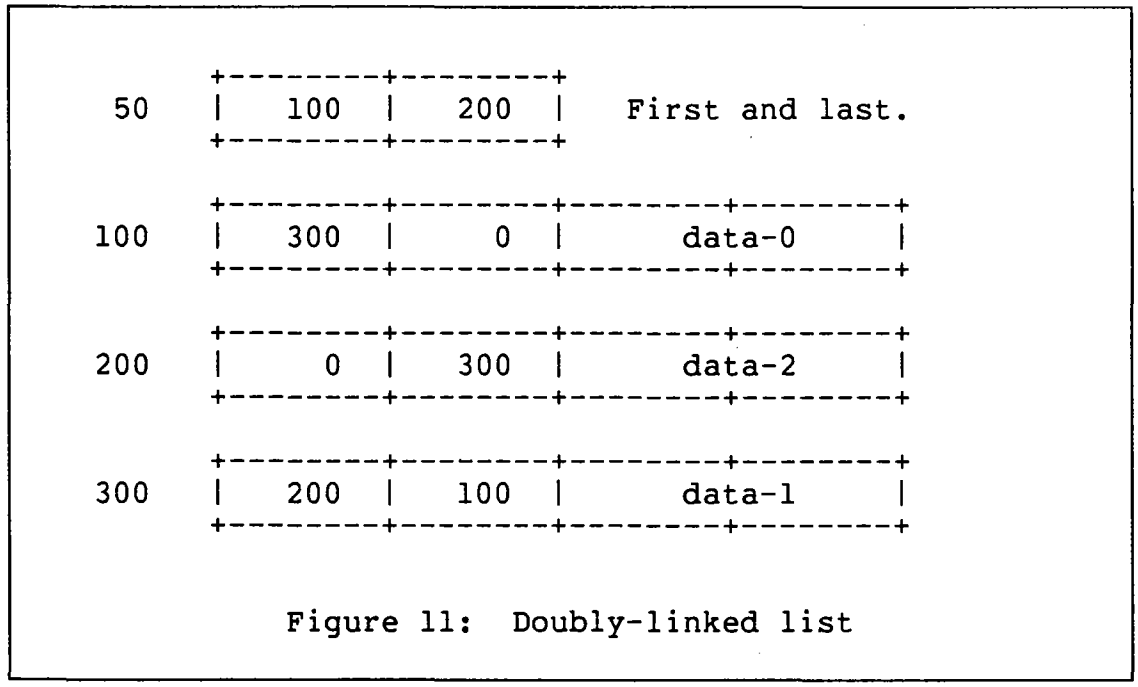


Figure 11: Doubly-linked list

3.2.1.2 Circular linked lists

In some cases it is inconvenient to use a linked list structure since the end conditions (pointer = 0) must always be tested. In such cases, CP uses a circular list, in which the last item points "forward" to the first item. Figure 12 below shows a circular list with single pointers. In processing such a list, you can identify the last item because its forward pointer contains the value in the anchor word.

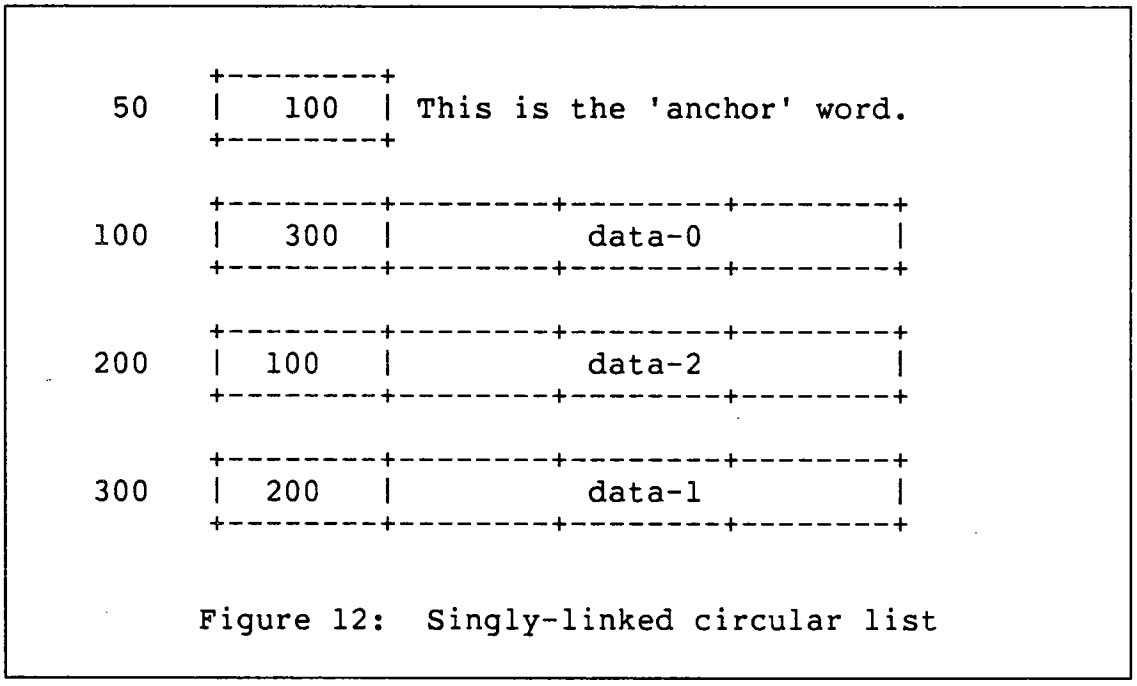


Figure 12: Singly-linked circular list

Double pointers could also be used if it is necessary to easily move backwards in the list or add or delete in the middle of the list. Figure 13 shows an doubly-linked circular list in which the anchors are included in the chain. This is a very common CP list structure.

3.2.1.3 Contiguous elements

In some cases the data items have a constant length and they can be addressed by some kind of indexing operation. For this case a set of contiguous control blocks may be more efficient because the pointer words do not have to be used. The System/370 segment table has this structure, with control register 1 acting as the anchor. Figure 14 shows an anchor and 4 contiguous data items.

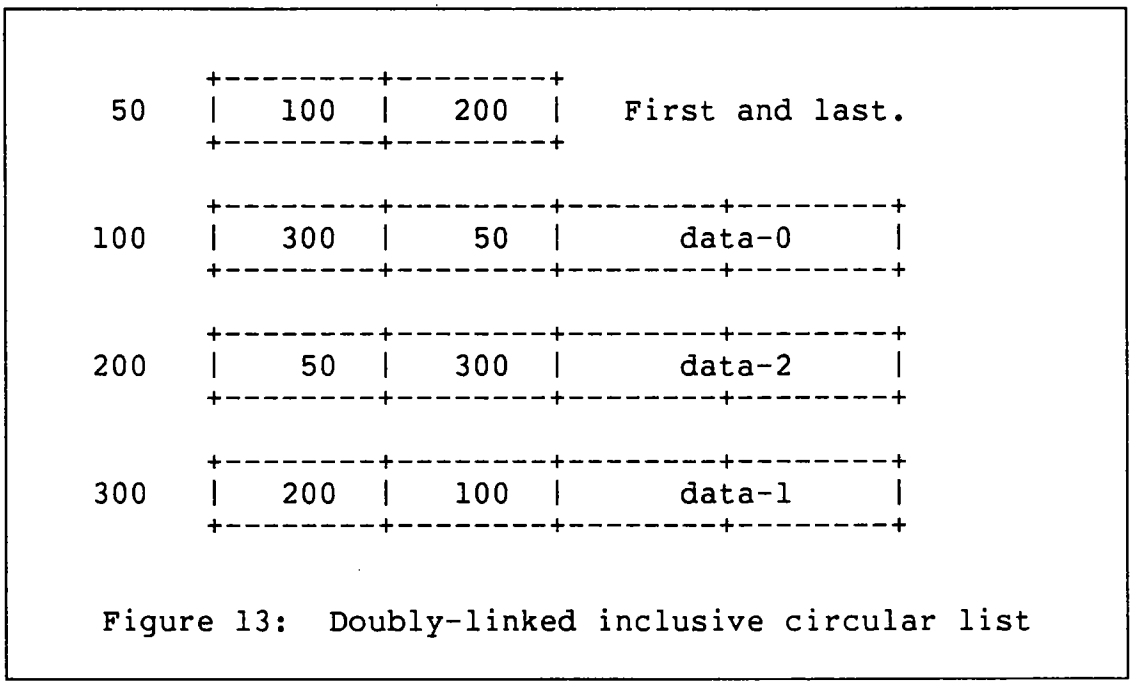


Figure 13: Doubly-linked inclusive circular list

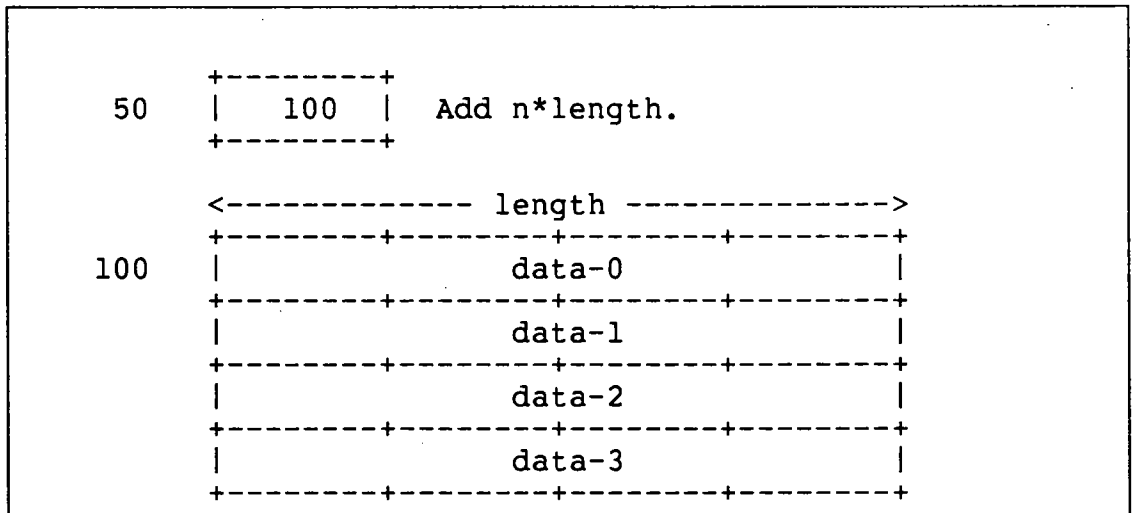


Figure 14: Contiguous items

3.2.1.4 Contiguous pointers

In yet other cases, it may prove more efficient to maintain a set of contiguous pointer words, each of which contains

the address of the actual data item. (This, you will recall, is the structure of the System/370 segment table, where each data item represents a page table.) Figure 15 shows this structure.

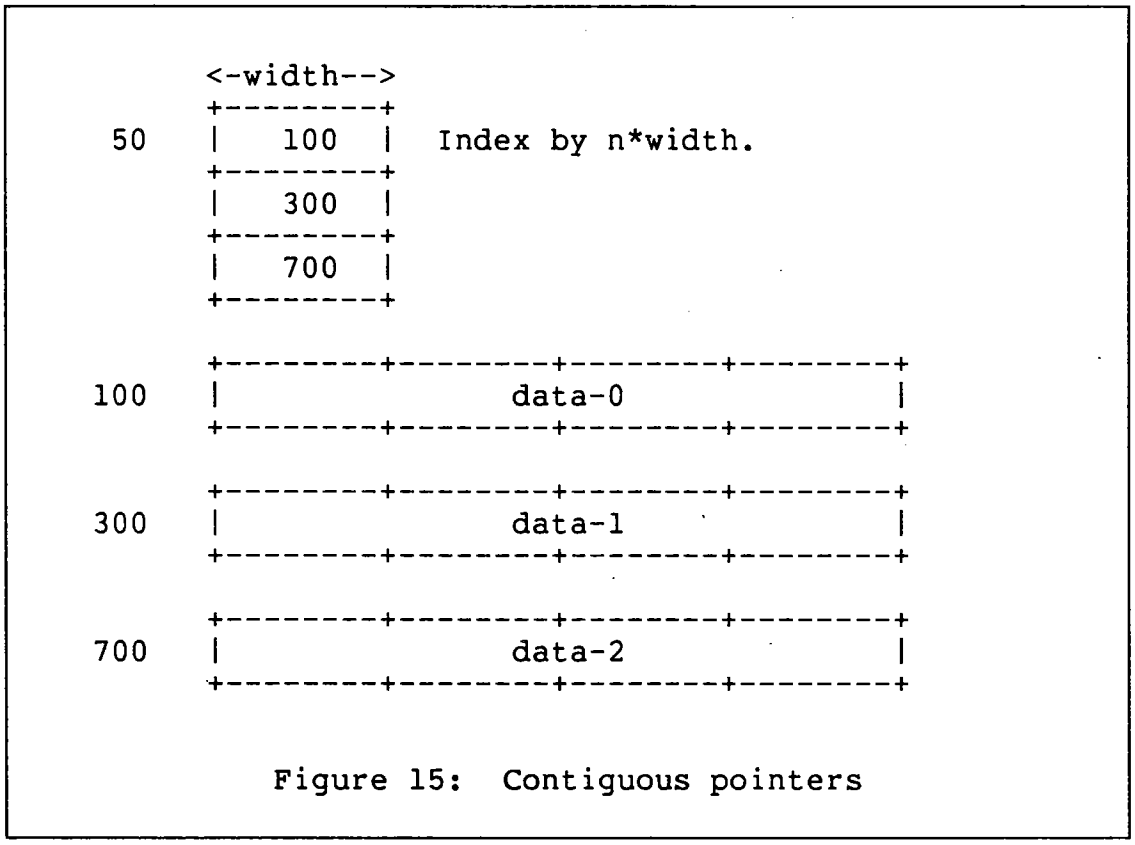


Figure 15: Contiguous pointers

3.2.2 Naming conventions

3.2.2.1 Module naming

CP follows a simple and straight forward naming convention for modules. Every module has a six character name beginning with the letters "DMK". The second three letters usually attempt to describe the function of the module. An example is the module DMKDSP, the CP dispatcher. Table 2 lists some of the modules and their functions; the capital letters show the derivation of the names.

TABLE 2

CP module names

DMKCFC - Console Function Commands
DMKCPI - CP Initialization
DMKDSP - DiSPatcher
DMKDAS - DASD error recovery
DMKFRE - FREe storage management
DMKIOS - I/O Supervisor
DMKPGT - Page GeT
DMKRSP - Real Spooling manager
DMKTHI - Temperature and Humidity Index (IND LOAD!)

3.2.2.2 Entry point naming

The names of module entry points (subroutines) are usually 8 characters long, where the first six are the module name and the last two are often descriptive of the associated function. Most entry point names define subroutines but some define pointer words or system-wide variables. Examples of entry point names follow in Table 3.

TABLE 3

CP entry point names

DMKIOSQV - Queue I/O from Virtual machine
DMKIOSQR - Queue I/O from CP (Real)

DMKDSPCH - main entry point (DiSPatCH)
DMKDSPA - fast reflect path
DMKDSPB - use if VPSW changes

DMKDSPQS - maximum time slice ("Quantum Size")
DMKDSPRQ - Request Queue (CPEXBLOKs and IOBLOKs)

DMKFREE - FREE storage get
DMKFRET - Free storage RETurn

3.2.2.3 Naming trends in CP

Some existing CP modules have been split as new functions or new device support have been added. The new name for the split-off module is usually the same as the name for the existing module, except that the last letter is changed to the next letter in the alphabet. For example, DMKCPI was split into DMKCPI and DMKCPJ. As one might expect, such a convention sometimes runs into trouble. Table 4 lists several module splits and also lists a family of modules.

TABLE 4

Naming trends - module splits

```
DMKPGT - DMKPGU
  DMKPGTPG - DASD Page Get
  DMKPGUPR - DASD Page Release

DMKCPI - DMKCPJ
  CP Initialization

DMKCFM - Console Function Mode processing
  DMKCFC - command name table
  DMKCFD - ADSTOP and LOCATE
  DMKCFF - (subroutine for DMKCFG)
  DMKCFG - IPL
  DMKCFH - SAVESYS
  DMKCFJ - BEGIN, QUERY, REQUEST, SET, and SLEEP
  DMKCFO - operator SET
  DMKCFS - initial parsing routine for SET
  DMKCFT - TERMINAL
  DMKCFU - privileged SET
  DMKCFV - non-privileged SET
  DMKCFW - SCREEN
  DMKCFY - non-privileged SET
```

3.2.2.4 Control block names

Control block names in CP are very regular, typically having a name of up to four descriptive characters followed by the pseudo-word "BLOK". For example, the "VMBLOK" is the control block that describes each virtual machine to the system. Similarly, the "IOBLOK" represents an I/O request, and

"CPEXBLOK" represents a small program to be executed as a part of CP. In most cases, the fields of a control block begin with the descriptive part of the control block name, making it quite easy to associate a field with its control block.

3.2.2.5 System-wide equates

Every CP module includes at least one special copy section; EQU COPY makes available a set of standard global names. These equates control the symbolic names of many hardware and software functions, such as the register names (R0 through R15), the PSW and control register bits, and certain commonly used software flags. Since the names in the EQU COPY are not restricted to any given module, they do not follow any particular naming convention

The conventions used with the names of fields and bit flags help make CP easier to understand; there are a few exceptions to the naming conventions, however, and we will try to point them out as we encounter them in each chapter.

3.2.3 Programming conventions

There is a regularity in CP modules which often makes CP easier to maintain and understand than some other operating systems. Since CP is distributed and maintained in source form, it is relatively easy to accomplish major modifications. In order to preserve order, system modifiers both in and out of IBM should respect the conventions. If the conventions are ignored, the system becomes more difficult to trouble-shoot and maintain.

3.2.3.1 Prologue

Every CP module has an English prologue with a description of what the module does, what its entry points do, what are the entry and exit conditions, and how the registers are used. Unfortunately, some prologues in CP show a certain amount of neglect. Under the pressure of time, programmers sometimes fail to update the English text so that for many critical modules the description is more history than reality. You will find that the prologues are an invaluable aid in understanding the modules, but you must keep in mind that the prologues may not be entirely up to date. You should be particularly careful concerning module names that might not reflect recent module splits.

3.2.3.2 Module attributes

CP modules may be either resident or pageable. The resident modules make up the resident nucleus, which is always present at a fixed address in real storage. The pageable modules are present in real storage only when they have been read in from a paging device and they may be at various storage addresses. They must therefore:

1. be no larger than one page (4096 bytes).

2. use the CALL and EXIT macros for linkage.

3. not use address constants (or the LA instruction) to refer to themselves or to other pageable modules (except in conjunction with the CALL or TRANS macros).

Pageable nucleus modules are temporarily locked into main storage during execution.

Resident modules need not be reentrant, except where required by their use in multiprocessor configurations. Pageable modules are usually reentrant, to avoid having to write them back out to the paging device when they are no longer needed.

With very few exceptions, all CP components run in supervisor state with address translation off and interrupts disabled. In general, therefore, CP modules run without loss of control between instructions.

3.2.3.3 Register conventions

Almost all modules in CP use the "Rnn" notation for registers. There are a very few exceptions, mostly in modules dealing with error recovery. Table 5 shows the register conventions used in most modules.

3.2.3.4 Linkage conventions

The linkage conventions in CP are tailored for high performance. The macros used for subroutine calls and returns are:

1. CALL -- The CALL macro is used to invoke a subroutine, usually located in another module. The macro is intelligent enough to recognize certain commonly-used resident subroutines that have special pre-allocated save areas and will generate a BALR R14,R15 instruction to invoke these subroutines. The macro

TABLE 5

CP nucleus register usage

R6 - VCHBLOK or RCHBLOK
R7 - VCUBLOK or RCUBLOK
R8 - VDEVBLOK or RDEVBLOK
R9 - Command line BUFFER
R10 - IOBLOK or TRQBLOK
R11 - VMBLOK
R12 - CSECT for current module
R13 - SAVEAREA
R14 - BALR return address
R15 - BALR called routine address

generates an SVC 8 instruction for all other subroutine calls. The SVC 8 performs several functions. First, it makes sure the called module is locked into memory (in case it is pageable). Second, it allocates a SAVEAREA for the called subroutine. Third, it preserves R12, R13, and the return address by storing them into the new SAVEAREA. Finally, it points R12 to the called entry point and then branches there. (Note that this is not the standard OS/360 save area format.)

2. EXIT -- If a subroutine was invoked via CALL (using SVC 8), then it must return using the EXIT macro, which restores R0 through R11 from the SAVEAREA and then issues an SVC 12. SVC 12 processing restores the caller's R12, R13, and the return address and also may unlock the called module if pageable. (If a subroutine was invoked via a BALR, then it simply restores the caller's registers from whatever save area it used and then returns with a BR R14 instruction; it must NOT issue an EXIT macro.)

3. GOTO -- The GOTO macro is used when one module wishes to pass control to another module without later getting control back. This occurs most often when a process is complete; such a process ends with a GOTO DMKDSPCH. The macro loads R12 with the specified entry point address and then branches there.

4. RELOC -- The RELOC macro is issued at each entry point in a pageable module. It stores R0 through R11

into the SAVEAREA. Remember that the CALL macro has already saved R12 and R13. RELOC adjusts R12 to point to the beginning of the module rather than to the specific entry point. This method of addressing is standard in CP.

5. ENTER -- The ENTER macro is a trivial subset of RELOC; it just saves R0 through R11 into the SAVEAREA.

3.2.3.5 Save areas

Most CP routines get control with R13 pointing to a SAVEAREA, as defined by the SAVEAREA DSECT. It is 96 bytes long and allows for R0 through R11 to be saved in the fields beginning with SAVEREGS. There is also room for 9 words of working storage labelled SAVEWRK1 through SAVEWRK9. The working storage area is available for any use that the routine requires, and many CP routines work hard to satisfy all their temporary storage requirements with these 9 words. Figure 16 shows the format of the SAVEAREA.

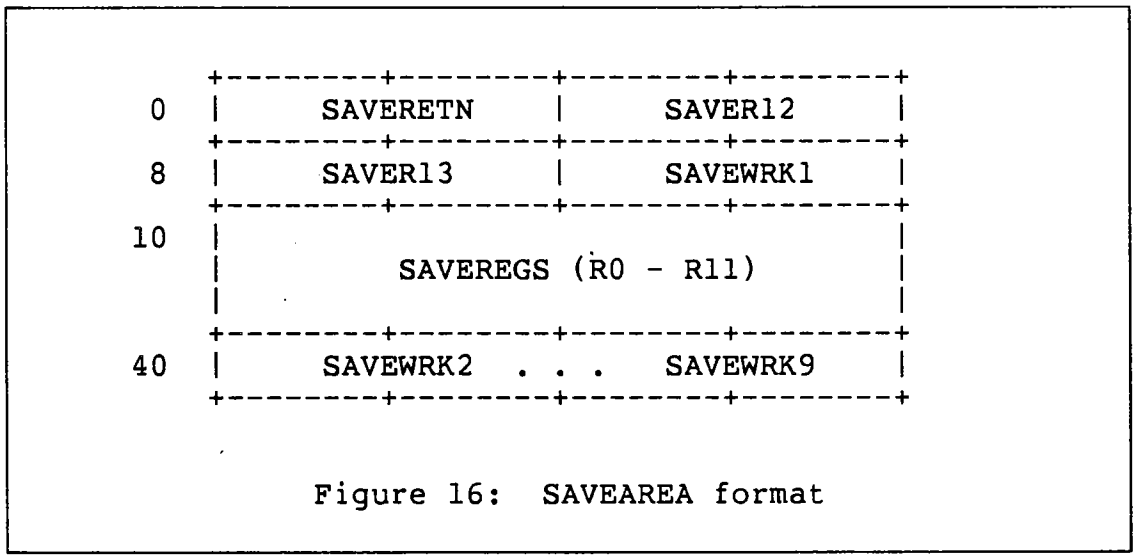


Figure 16: SAVEAREA format

To improve system efficiency, CP initialization constructs a list of pre-allocated SAVEAREAs. SVC 8 (CALL) obtains the next element from the list, if available, and SVC 12 (EXIT) places the SAVEAREA back onto the list for re-use.

Some CP routines do not use the standard SAVEAREA but instead use one of several special register save areas, as listed below:

1. TEMPSAVE (X'200':X'23F') is used as a temporary save area by various routines.

2. BALRSAVE (X'240':X'27F') is used by most routines that are called by BALR and not by SVC 8.

3. FREESAVE (X'280':X'2BF') is used by the free storage management routines in DMKFRE.

4. Several other specialized save areas also exist for use by the first-level interrupt handlers and by the special multiprocessor communication routines.

3.2.3.6 Other SVC usage

In addition to SVC 8 and SVC 12, there are several other SVCs used by CP.

1. SVC 0 results in a CP ABEND. This will be discussed at length during the chapter on the CP trace table and dumps. Let it be said that SVC 0 is invoked whenever any part of CP encounters a serious error. SVC 0 invokes the system dump routine, DMKDMP.

2. SVC 16 returns a SAVEAREA. This SVC simply gives back a SAVEAREA without the loss of control involved with an SVC 12. R13 is restored from the SAVER13 field.

3. SVC 20 gets a SAVEAREA. A SAVEAREA is given to a program from the SAVEAREA stack maintained by DMKSVC. Unlike SVC 8, SVC 20 does not cause loss of execution control. R13 is set to point to the SAVEAREA, which contains the previous R13 value in SAVER13. If DMKSVC's stack of SAVEAREAs is depleted, it will call the free storage manager, which might send the system into that dreaded state, "extend" processing. Extend processing will be discussed in much more detail, but it is important to understand that control can be lost while getting a SAVEAREA if a system extend is necessary.

4. SVC 24 is an esoteric SVC. In a multiprocessor environment it stacks a high-priority CPEXBLOK for the other processor.

5. SVC 76 is the error recording SVC in CP and guest SCPs. If the SVC 76 routine can understand the general register contents it gets on entry, then it will record the error. If, however, the contents are unintelligible, then the SVC is reflected to the virtual machine.

3.3 MEMORY

3.3.1 Real memory

Main storage in a CP system can be divided into several basic areas. For convenience, the initial main storage divisions are given in Figure 17. Each page frame of main storage is described by an entry in a control block called the CORTABLE, located in module DMKSYS. By scanning the CORTABLE, CP can identify the use of each page frame.

3.3.1.1 Prefixed storage area

The PSA is a processor's real page frame 0, in which hardware generated interrupts are posted. It is also one of the main CP control blocks, since it is always addressable without a base register. The PSA contains many anchors for chains of other control blocks, several special pre-allocated save areas, and a set of commonly-used constants.

3.3.1.2 V=R region

The V=R region allows an almost one-to-one mapping of a guest operating system's virtual storage and real main storage. The mapping is exact except that the guest operating system's page 0 (virtual PSA) can not be the real PSA. CP puts the V=R page 0 at the end of the V=R region. The size of the V=R region is set in DMKSLC, which is not actually a module but is instead a TEXT deck containing an "SLC" loader control card.

3.3.1.3 CP nucleus

The CP resident nucleus is always located in contiguous storage just above the optional V=R region. All resident CP modules are in this area. The end of the resident nucleus is DMKCPEND in module DMKCPE.

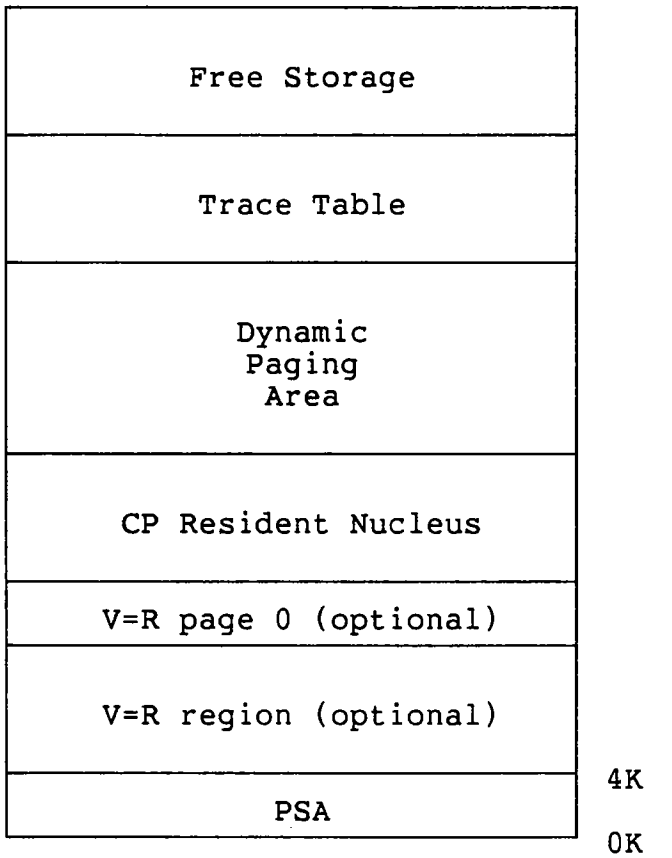


Figure 17: Layout of real memory

3.3.1.4 Dynamic paging area (DPA)

The dynamic paging area is that portion of main storage left after all other sections have been assigned. It is the area in which virtual machine pages and pageable nucleus pages are located. When main storage is a scarce resource, the DPA can also be used to satisfy requests for CP control block storage.

3.3.1.5 CP trace table

The trace table is an area in which critical operations of CP are recorded for error detection and correction purposes. The trace table is logically one endless buffer into which many CP routines insert fixed length entries.

3.3.1.6 Free storage area

The free storage area is that portion of main store set aside for CP control blocks. It is managed by the module DMKFRE. If the free storage area becomes depleted enough, then DMKFRE will "extend" into the DPA by taking control of a page. This involves a great deal of system overhead and is therefore an undesirable condition.

3.3.2 Virtual machine memory

CP uses dynamic address translation to define the memory of each virtual machine. A segment table and associated page tables are built for each virtual machine and that set of tables is used by the real hardware when the virtual machine is running. Since the virtual machine cannot see the tables, CP is free to specify which pages are currently in real memory. The virtual machine will run normally as long as it references only those pages. Any attempt to reference other pages will result in an interrupt that will pass control back to CP. The paging and real storage management routines in CP will cause the referenced virtual machine page to be brought into real storage and will update the segment and page tables so that the virtual machine can be allowed to re-execute the interrupted instruction. The appropriate CORTABLE item will be updated to show that the real page frame now contains the particular virtual machine page.

CP itself uses segment and page tables for certain special functions, thus allowing the use of various common routines that perform paging I/O operations. Note, however, that CP does not run with DAT enabled and therefore does not incur page faults.

3.4 CPU

3.4.1 Real CPU

Conceptually, CP consists of a single endless main program, the dispatcher (DMKDSP). That program passes control to other CP routines to perform pieces of work. If no CP work remains to be done, then the dispatcher in effect re-configures the real System/370 so that a virtual machine will begin execution.

The items of CP work to be performed are represented by 3 kinds of control blocks. These blocks are created by various CP routines and are then "stacked" onto queues of work

to be done. All CP execution is under control of these blocks except for the dispatcher itself and the initial handling of real interrupts.

3.4.1.1 CPEXBLOK

The CP execution block (CPEXBLOK) is the major control block for specifying a piece of work to be performed at a later time. The CPEXBLOK contains the values for the registers (CPEXREGS) and an execution address (CPEXADD). The doubly-linked circular queue of pending CPEXBLOKs is manipulated by DMKSTK and by DMKDSP. DMKSTK "stacks" a CPEXBLOK by placing it onto the queue. Soon thereafter, DMKDSP "unstacks" the CPEXBLOK by taking it off the queue, loading the real registers, and branching to the CPEXADD address; note that this is a branch and not a call. CP is then said to be "running under" the CPEXBLOK. Figure 18 shows the format of the CPEXBLOK and contains an example of its use.

3.4.1.2 IOBLOK

The IOBLOK is the element by which a CP I/O operation is initiated, and it is also the element that controls execution of an I/O interrupt handling routine. When an I/O operation completes, the IOBLOK is "stacked" by the interrupt handler and then later "unstacked" by the dispatcher, which points R10 to the IOBLOK, loads R11 from IOBUSER, and loads R12 from IOBIRA (interrupt return address); no other registers are loaded. The dispatcher then branches to the address in R12, making the interrupt handling routine in effect a subroutine of the dispatcher. The routine will ultimately return via GOTO DMKDSPCH. Figure 19 shows the format of the IOBLOK.

3.4.1.3 TRQBLOK

The timer request block (TRQBLOK) is used by CP routines to request a TOD clock comparator interrupt. At the specified time, the TRQBLOK is stacked to the dispatcher and is then unstacked just like an IOBLOK. In fact, the very same fields and registers are involved: R10 points to the TRQBLOK, R11 is loaded from TRQBUSER, and the dispatcher branches on R12 to the address in TRQBIRA. Figure 20 shows some of the fields in the TRQBLOK.

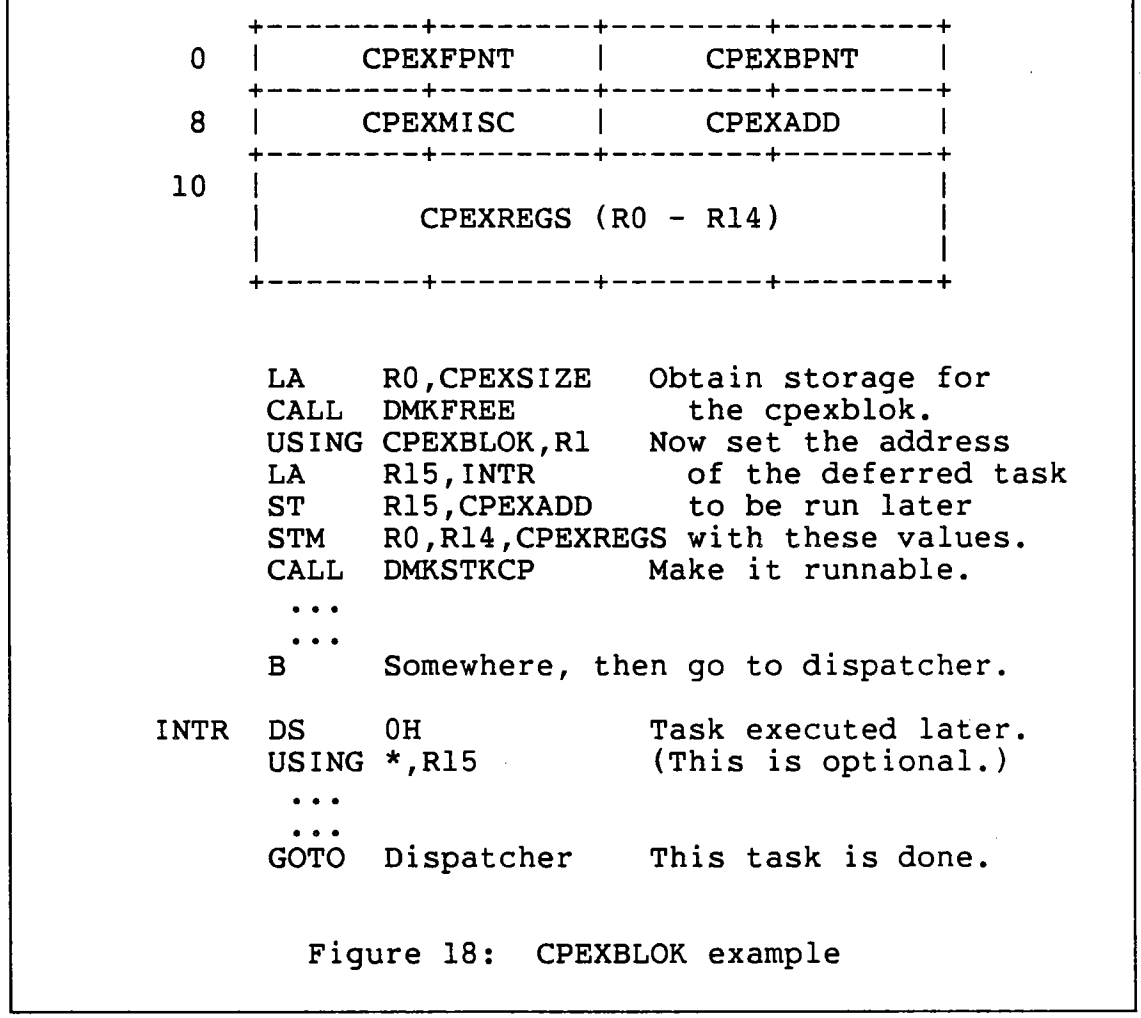


Figure 18: CPEXBLOK example

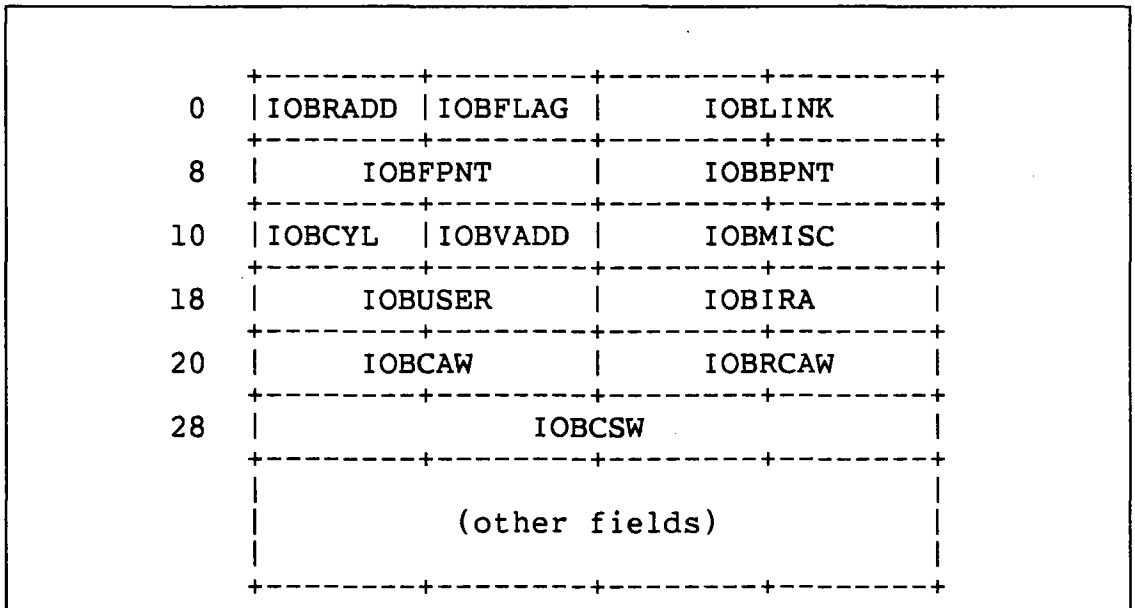


Figure 19: IOBLOK format

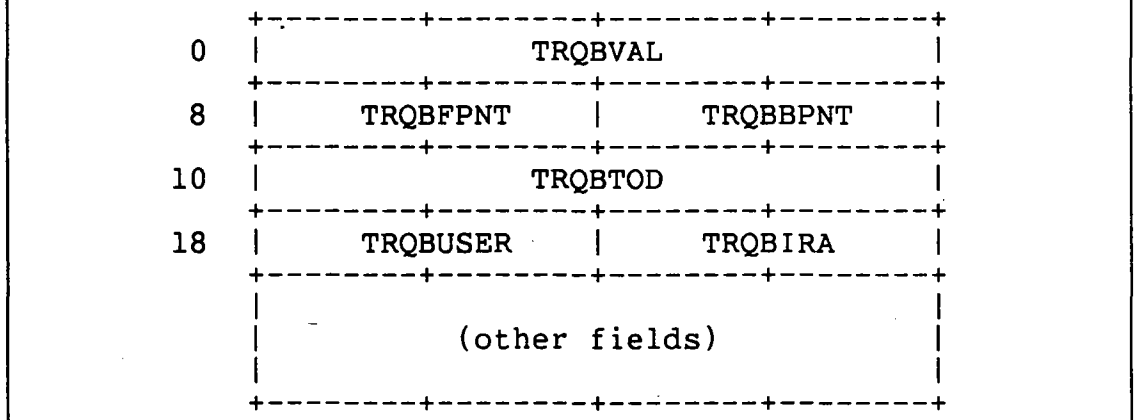


Figure 20: TRQBLOK format

3.4.2 Virtual machine CPU

The virtual machine CPU is represented by the VMBLOK control block. Each VMBLOK contains a virtual machine's status and its register contents. (Since the VMBLOK is the major control block for a virtual machine, it also contains many other fields that have nothing to do with the virtual CPU.) The VMBLOKs are circularly chained together and the chain is anchored in the PSA. The VMBLOK points to other control blocks containing addition information, such as the segment table, the virtual I/O configuration, and the user's real terminal device. The user virtual machine VMBLOK is created at LOGON and is deleted at LOGOFF. The system's own private VMBLOK and the system operator VMBLOK are both assembled as a part of module DMKSYS and therefore always exist as the first two items on the VMBLOK chain.

3.5 I/O

3.5.1 Real I/O

3.5.1.1 Real I/O control blocks

The real I/O configuration is represented by 3 types of control blocks. Each is of fixed length and all of them are located within the module DMKRIO. To change CP's view of the I/O configuration, you must update and assemble DMKRIO and then generate a new copy of the entire CP nucleus.

1. The RDEVBLOK represents a real I/O device and contains the device address, its status, pointers to its control units, and pointers to active or waiting IOBLOKs. All of the RDEVBLOKs are contiguous in storage. A field in the PSA points to the first RDEVBLOK and most other control blocks refer to the RDEVBLOK by an index value which is its displacement from that first one. The RDEVBLOK is generated by the RDEVICE macro in DMKRIO.

2. The RCUBLOK represents a real control unit and contains the controller address, its status, pointers to its channels, pointers to its current IOBLOKs, and indicies to its RDEVBLOKs. All the RCUBLOKs are contiguous and the first one is addressed by a pointer word in the PSA. The RCUBLOK is generated by the RCTLUNIT macro in DMKRIO.

3. The RCHBLOK represents a real I/O channel and contains the channel address, its status, pointers to its current IOBLOKs, and indicies to its RCUBLOKs. All the RCHBLOKs are contiguous and the first one is

addressed by a pointer word in the PSA. The RCHBLOK is generated by the RCHANNEL macro in DMKRIO.

Whenever a CP routine needs to perform I/O, it constructs an IOBLOK (as described above) and calls DMKIOS, the I/O supervisor, which ultimately performs the desired I/O operation. The queue of IOBLOKs for a given device is anchored in the RDEVBLOK.

3.5.1.2 Real DASD areas

There are several different DASD areas that are reserved for CP functions. The areas are:

1. Nucleus space contains a copy of the CP nucleus, both resident and pageable. At CP initialization time, the nucleus is read into main storage.

2. Directory space is an area that defines the configuration of each virtual machine. Each description contains the memory limits, I/O configuration, and options for the virtual machine.

3. Checkpoint space is the area in which critical control blocks needed to recover spool files after a system crash are written. It is used during CP initialization.

4. Error recording space is that area into which CP records hardware errors for later reporting via the EREP program.

5. Page space is the area in which CP keeps virtual machine memory pages when they are not in use in main storage. The page space is sometimes called preferred page space.

6. Spool space is really a misnomer, since there is really no space reserved exclusively for spool files. Internally, CP refers to this space as "temp" space, the area that is used to hold spool files and any pages that overflow the preferred page space.

7. Dump space is a new and optional area that is used to hold the spool pages of a CP system dump. If there is not sufficient dump space defined in a system, CP uses spool space to hold the dump. Because of the relative simplicity of the dump-writing routine, dump files must occupy contiguous DASD records in either dump or spool space.

8. Named system space is a collection of areas described by entries in the module DMKSNT. This facility provides pre-loaded virtual memory contents for commonly used processes such as CMS.

9. Warmstart space is the area in which critical spooling control blocks are saved across planned and unplanned system outages. It is written during system SHUTDOWN and ABEND processing and it is read during system initialization.

Each of these areas is formatted into page-sized records of 4096 bytes. The first data record on track 0 of a given cylinder is record 1, and the record number increases continually from track to track to the end of the cylinder. On some DASD devices filler records are placed between the page records to allow time for track switching within a single channel program. This format is discussed in more detail in the paging chapter.

3.5.2 Virtual machine I/O

The virtual machine I/O configuration is defined by a series of control blocks that are very much like those used for the real I/O configuration. Each virtual device has its VDEVBLOK, each virtual controller has its VCUBLOK, and each virtual channel has its VCHBLOK. These are interconnected by pointers and are anchored in the VMBLOK; in appropriate cases, the VDEVBLOK also points to the associated RDEVBLOK.

CP's handling of virtual I/O instructions involves simulation of the instruction according to the state of the virtual I/O system as shown in the VxxxBLOKs. When appropriate, CP will construct an IOBLOK for the real I/O operation resulting from the virtual operation.

When the dispatcher prepares to let a virtual machine resume execution, it will reflect virtual I/O interrupts to the virtual machine when those interrupts are pending in the VxxxBLOKs and enabled in the virtual PSW and control registers. The dispatcher will reset the appropriate VxxxBLOK status bits to show that the pending interrupt has been taken.

3.6 EXTERNAL OPERATIONS

3.6.1 Real external operations

CP uses the real timers for virtual timer simulation, for dispatching and scheduling purposes, and to accumulate accounting data. TRQBLOKs are used whenever CP needs to set an "alarm clock" to expire at some future time. The queue of TRQBLOKs is kept in order by ascending time-to-expire and the first item in the queue has its value loaded in the real clock comparator. The interval timer and CPU timer are loaded and stored as needed for scheduling and accounting purposes.

Other external interrupts cause control to pass to specific handling routines. These exist for the EXTERNAL key (to disconnect the system operator) and for the various multiprocessor interrupts.

3.6.2 Virtual machine external operations

The chapter on timer handling describes the details of CP's simulation of the various virtual timers, including how and when the virtual values are loaded into the real timers. The dispatcher examines a queue of external interrupt blocks (XINTBLOKs) that represent pending virtual external interrupts. When a virtual machine can be dispatched, and if it is enabled for a pending external interrupt, then the dispatcher will reflect that virtual interrupt via virtual PSW swap.

3.7 SYSTEM CONSOLE

3.7.1 Real system console

The real system console is used by the system operator to IPL CP to begin VM/SP operations. The real RESTART key can be used by the operator to cause a CP ABEND and dump. The system console itself is otherwise hardly used. (This is of course distinct from the I/O device that is the LOGON device for the system operator and from which the operator issues CP commands to control VM/SP.)

3.7.2 Virtual machine system console

The virtual system console is simulated on the user's LOGON terminal through a set of CP commands and messages. These correspond roughly to the real system's keys and indicators. The various commands will change the state of the virtual machine as appropriate to their functions. Some additional features are also provided, such as the ability to define new virtual devices, to change the virtual memory size, or to send messages to the operator of some other virtual machine.

3.8 SUMMARY

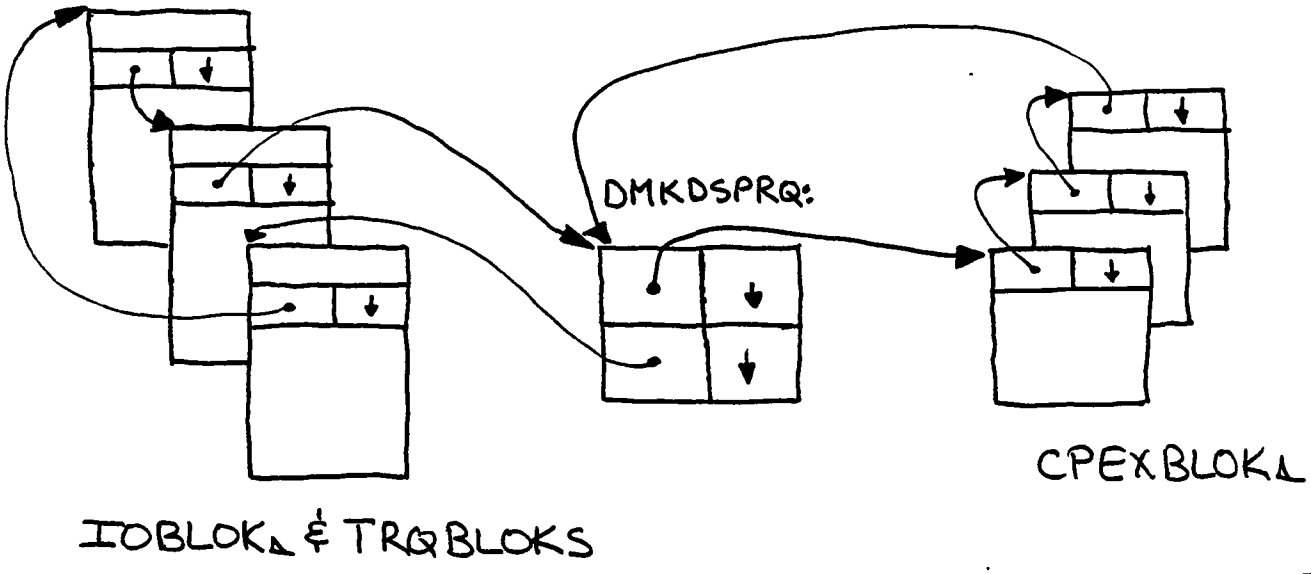
In many cases, in order to understand what CP is doing and why it is doing it, you need only to think about the real System/370 hardware; CP takes the place of hardware for many portions of the virtual machine. Other aspects of CP are more general in nature and are not based upon hardware simulation. Many of these structures have evolved over a long period of time and are not as easy to understand.

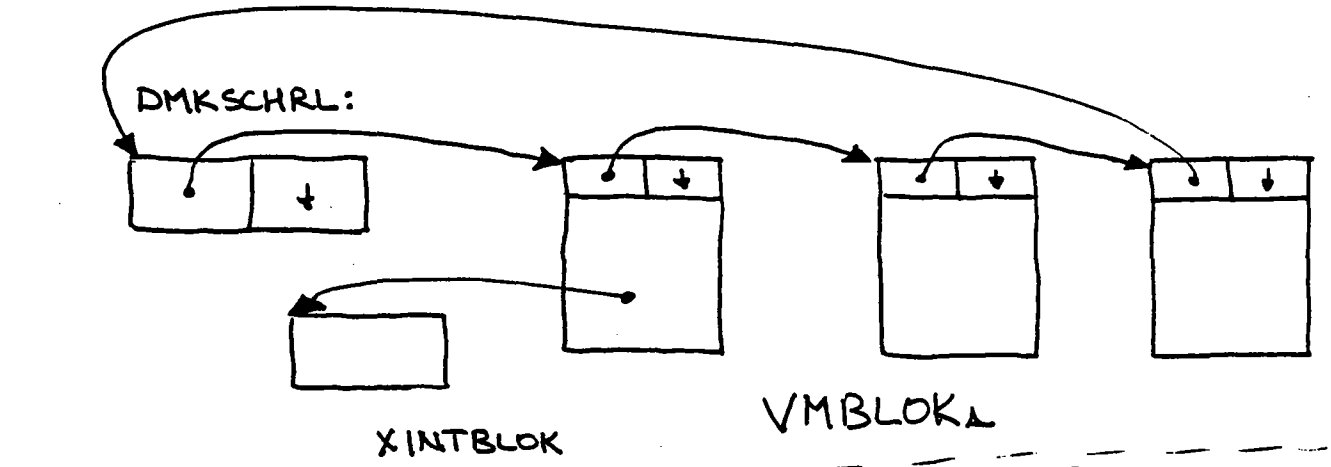
NOTES

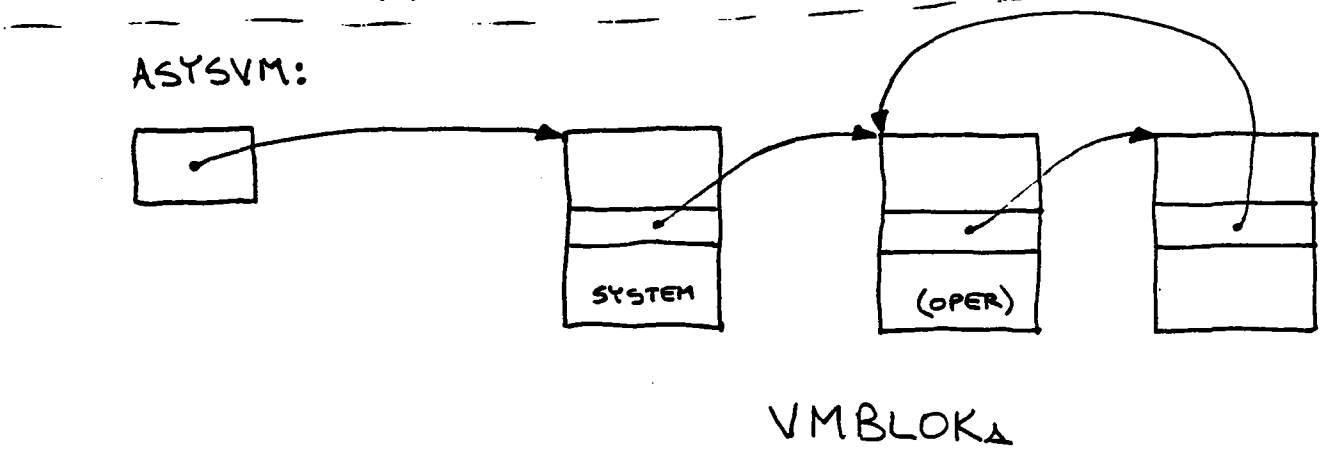
PART II

SPECIFIC TOPICS

The following chapters each discuss some specific topic in the design and structure of CP. Although the topics are interrelated and do interact with each other, we have tried to describe each one independently. In order to make it easier for you to find material about a given subject, we have arranged the chapters into related groups. This leads to some minor inconveniences; for example, storage management and paging are grouped together and they both preceed the chapter on general I/O processing, even though paging of course involves I/O operations. Please keep this in mind if you try to read straight through the chapters.







DISPATCHER OVERVIEW

Chapter 4

DISPATCHER

4.1 INTRODUCTION

4.1.1 Overview

The CP dispatcher, in module DMKDSP, is responsible for reflecting asynchronous events to virtual machines via the simulation of the 370 interrupt mechanism, for the dispatch of asynchronous tasks within CP, and for the dispatch of virtual machines. Understanding the operation of the dispatcher is vital to understanding the flow of control within CP since all paths eventually lead to the instruction labelled DMKDSPCH, "the end of the world".

Detailed discussions of logic concerned with MP/AP support, timer maintenance, single processor mode, and the Quiesce VM support have been deferred to later chapters.

4.1.2 References

4.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

3. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

4. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

4.1.2.2 CP modules

1. DMKDSP - handles primary dispatching function for CP and virtual machine tasks.

2. DMKSTK - places requests for deferred execution on the appropriate dispatcher queue.

3. DMKSCH - handles the ordering and selection of virtual machines that are "in-queue" and therefore eligible to be dispatched.

4.1.3 Flow of control

The dispatcher gains control, usually through its main entry point at DMKDSPCH, when processing is complete for some CP function. DMKDSP also gains control when the initial processing for an interrupt has been completed and an IOBLOK, TRQBLOK, or CPEXBLOK has been stacked for later processing. Other entry points gain control:

1. after a virtual location 80 timer interrupt.

2. for a fast re-dispatch of the current user after a program interrupt.

3. when the virtual PSW of a machine has been switched and therefore must be checked for validity.

The processing at these other entry points differs slightly from the main entry since some tests need not be made at certain of the entry points.

If the last virtual machine dispatched (pointed to by the PSA field RUNUSER) has the VMDSP flag set in the VMDSTAT field of the VMBLOK, the dispatcher will skip many checks (even though entered at its main entry point) in an attempt to reduce overhead as much as possible for this frequently executed code. In particular, if a virtual machine was running at the time of an I/O interrupt (the I/O old PSW has the problem state bit on), then many unnecessary checks can be avoided.

The normal flow of control described in this chapter is:

1. Maintain CPU utilization and wait statistics.

2. Perform the following functions for the virtual machine whose VMBLOK is pointed to by R11.

   a) Simulate or "unstack" virtual machine interrupts.

   b) Validate new PSW and check for idle or disabled wait states.

c) Interface to the scheduler for any virtual machine status changes.

3. Unstack and dispatch pending IOBLOKs or TRQBLOKs.

4. Unstack and dispatch pending CPEXBLOKs.

5. Dispatch highest priority virtual machine that is ready to run.

6. Determine allocation of wait time and load a wait PSW if there is no work to perform.

That is a lot of function for one CP module, but then the dispatcher has grown to require three base registers to cover all its code.

4.2 MAINTENANCE OF CPU UTILIZATION STATISTICS

Among the duties of the dispatcher is the collection of statistics on CPU utilization. As mentioned previously, much of the logic dealing with timer maintenance is covered elsewhere. The material in this section is brief and is included to complete the picture of the tasks performed within the dispatcher.

4.2.1 CP time and problem state time

A flag in the PSA, CPSTATUS, indicates the state of the processor before the current function was performed. If the flag indicates that a virtual machine was being run, various timers are updated. At this time, regardless of which virtual machine was "current" when the dispatcher was entered (R11 pointed to the VMBLOK of the virtual machine), R11 is reloaded to point to the VMBLOK for the virtual machine running when CP gained control (from the field RUNUSER in the PSA). The field PROBTIME in the PSA is updated to reflect the amount of time the processor has been running in problem state (running a virtual machine). The location 80 interval timer for a virtual machine is updated (in processors that do not have microcode assist to maintain the interval timer) and an interrupt is posted if the timer has become negative.

If the CPSTATUS flag indicates that an asynchronous CP task (CPEXBLOK, IOBLOK, or TRQBLOK) has been executing and is now complete, control proceeds to the code that attempts to unstack interrupts for the current virtual machine whose VMBLOK is pointed to by R11.

4.2.2 Wait time

If CPSTATUS indicates that the processor was in wait state prior to performing the current function, then the appropriate values in the PSA are incremented to indicate the amount of time spent in IONTWAIT, PAGEWAIT, or IDLEWAIT. The manner in which the wait time is allocated to I/O, paging, or idle wait when the dispatcher loads a wait PSW is covered in a later section. Since there is no "current" virtual machine to check for interrupts that require unstacking, control is passed to the check of the dispatcher queues for asynchronous CP tasks.

4.3 VIRTUAL MACHINE INTERRUPT SIMULATION (UNSTACKING)

Simulation of the 370 hardware-performed function of storing an old PSW and loading a new PSW from a fixed location in page 0 is performed within the dispatcher. This section of code is skipped if the dispatcher can determine that no event calling for an interrupt to be unstacked (simulated) has occurred (VMDSP in VMDSTAT is on) or if the virtual machine is in a wait for some CP function to be completed.

The form of the interrupt simulation is tied very much to the architecture of the virtual machine. The CP command

SET ECMODE OFF | ON

causes the switch between 360 and 370 architecture virtual machines. CP indicates 370 architecture by setting the VMV370R flag in VMPSTAT of the VMBLOK. Since 370 architecture virtual machines may be in either BC or EC mode, the fact that a virtual machine is currently running in extended control mode is signalled by the flag labelled VMEXTCM in VMESTAT of the VMBLOK.

4.3.1 PER and pseudo page fault interrupts

Virtual PER (Program Event Recording) interrupts are simulated by storing the old virtual machine PSW in the program check old PSW location, saving the appropriate interrupt codes in the hardware-defined locations of the virtual machine's page 0 and branching to the PSW validation section of the dispatcher to check out the Program Check New PSW.

Pseudo page fault (PPF) interrupts are simulated as a special type of program check (X'14') and indicate that a page requested by a task running within a multiprogramming supervisor with VM handshaking (e.g. VS/1) is now available.

These interrupts are unstacked only if the virtual machine is running with an EC mode PSW and the actual work is performed by a routine in DMKVAT (virtual address translation). It is possible for multiple PPFs to be pending; each pending interrupt is described by a PGBLOK and the VMPGPNT field in the VMBLOK points to the queue of PGBLOKs for a virtual machine.

4.3.2 External interrupts

If the virtual machine is disabled for external interrupts or if the VMPXINT field of the VMBLOK is zero (indicating that no external interrupts are pending) processing continues at the point of checking for pending I/O interrupts. VMPXINT points to the queue of XINTBLOKs that describe each pending external interrupt by priority, interrupt code, and any bits that must be enabled in CR0 to accept this interrupt.

Special processing is required for external interrupts caused by functions only provided within CP and not defined by the 370 architecture. For these external interrupts, other CP routines are called (see table 6) to store additional interrupt information such as communication parameter lists.

TABLE 6

Special external interrupts within CP

Code	*CR0 Mask*	*Routine*	*Function*
X'2402'	Bit 22	DMKHPSEX	Logical Devices
X'4001'	Bit 31	DMKVMCEX	VMCF
X'4000'	Bit 30	DMKIUARF	IUCV

If the virtual machine's page 0 is not resident, it is paged-in. The external interrupt is simulated as required by 360 or 370 architecture. If the external new PSW is enabled for interrupts and the interrupt is not class 0 (external interrupts only presented once, such as the location 80 timer), then an external interrupt loop condition exists.

A further check is made for an external interrupt loop that occurs if the external new PSW is invalid (resulting in a program check) and the program check new PSW is enabled for external interrupts. In either case, the virtual machine is made not-runnable (placed in console function mode) and a message is written to the VM's console. When external interrupt stacking is complete, a branch is made to the start of the code to unstack interrupts; a new PSW means that all checks for pending interrupts must be repeated.

4.3.3 I/O interrupts

The code for simulating I/O interrupts is executed only if the virtual machine has an interrupt pending on a channel that is enabled for interrupts. The VMIOINT halfword field in the VMBLOK contains a B'1' for each channel with an I/O interrupt pending. This interrupt pending mask is "ANDed" with the mask of enabled channels (constructed from the VM's PSW and, for virtual 370 architecture machines, the contents of virtual control register 2). If the result is zero a branch is taken to the dispatcher code that checks idle PSW wait conditions (see below).

The subsequent eight-page section of code in the dispatcher deals with scanning the virtual I/O control blocks looking for the first channel/control unit/device combination that has an interrupt pending. Once the pending interrupt is found the virtual CSW is constructed, PSWs are swapped, and the interrupt pending status is cleared from the appropriate virtual I/O control blocks. Control is then transferred to the dispatcher routine that validates a new virtual PSW.

4.4 NEW PSW VALIDATION

The purpose of this section of code is to ensure architectural consistency within the tables and control blocks being maintained for a virtual machine. The only check for a 360 architecture virtual machine is to ensure that the EC mode bit in the PSW is not turned on; if it is, the VM is put in console function mode and an error message is written to the virtual machine's console. For 370 architecture virtual machines (having VMV370R set in VMPSTAT), the major concern is for virtual machines that are entering or leaving extended control mode (VMEXTCM in VMESTAT and EXTMODE in VMPSW+1 are not both B'1' or B'0'). In the case of a mode change, routines in DMKVAT are called either to release or construct the shadow page tables that are needed to handle multiple address spaces within a virtual machine.

Once the new PSW check is complete, control normally transfers to the section of code that unstacks pending interrupts. This is necessary in case the new PSW is enabled for interrupts.

4.5 CHECK FOR DISABLED OR IDLE WAIT

The purpose of this section of code is to check for a virtual PSW wait condition that will never be satisfied (disabled wait) or one that may not be satisfied for more than several hundred milliseconds (idle wait).

The halfword mask field VMIOACTV in the VMBLOK is "anded" with the mask of enabled channels to determine if any I/O operations are active that would remove the virtual machine from the wait condition; if so, this section of code is exited since the virtual machine is probably in a short wait condition. Positive tests for IUCV or VMCF messages outstanding to virtual machines with the SET QDROP OFF USERS option (if the current virtual machine is enabled for VMCF or IUCV interrupts) or for pages in transit when VM handshaking is being used (VMNDCNT in the VMBLOK is nonzero) imply a short wait condition so this section of code is exited.

If the virtual machine is enabled for I/O interrupts on any channel, or if it is enabled for external interrupts and the control registers indicate the machine is enabled for timer, IUCV, VMCF, or external button interrupts, then the virtual machine is flagged as being in idle wait (VMIDLE in VMRSTAT field of the VMBLOK is set to '1'B) and this section of code is exited. When the above tests fail, the virtual machine has entered a disabled wait state. The virtual machine is put in console function mode and a warning message containing the hexadecimal representation of the disabled PSW is written to the virtual machine's console.

4.6 INTERFACE TO SCHEDULER

At this point in the processing, the interrupt simulation is complete and DMKDSP is just about ready to get down to the real task of dispatching. However, the current virtual machine may have experienced an important status change during the processing just completed. This change may require that the scheduler be notified of the new state of affairs.

The check as to whether or not the scheduler must be called is rather cursory but does eliminate unnecessary calls to the scheduler for the most common cases (e.g. a

running virtual machine was interrupted by a privileged operation that has been simulated and the machine can be run again without delay). The scheduler is called unless all the following conditions are true:

1. The VMRSTAT field of the VMBLOK is clear, indicating that the virtual machine is not waiting on any external events or CP services (i.e. the virtual machine is runnable).

2. The VMDSTAT field of the VMBLOK indicates that the virtual machine was runnable the last time the scheduler was called for this virtual machine (VMRUN is set on), that the virtual machine is in the run list (VMINQ is set on), and that there has not been a queue slice end or a time slice end.

3. The VMOSTAT field of the VMBLOK indicates that the virtual machine is not being logged off the system (VMKILL is not set).

4. The VMQSTAT field of the VMBLOK indicates that the virtual machine has not just performed some console I/O (VMPRIDSP is not set).

If the test fails because the VMKILL flag is set, the module DMKUSOFF is called to start logoff processing before the scheduler is called.

4.7 DISPATCH CP SERVICES

There are two queues of deferred CP functions maintained for the dispatcher: the IOBLOK and TRQBLOK queue consisting of completed I/O and timer requests, and the CPEXBLOK queue consisting of other deferred CP tasks that must now be performed. These queues are anchored at DMKDSPRQ, a 4-word area whose format is show in figure 21. Note that the deferred tasks gain control by direct branch and not via CALL or BALR. Return of control to the dispatcher from one of these tasks is usually by a return branch to the entry DMKDSPCH.

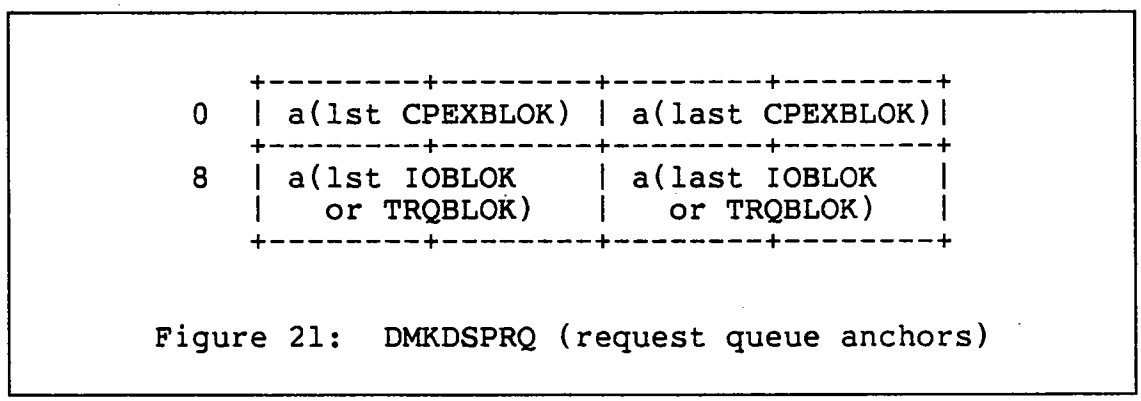


Figure 21: DMKDSPRQ (request queue anchors)

4.7.1 Unstack and dispatch IOBLOKs and TRQBLOKs

Since the fields of the IOBLOK and the TRQBLOK that the dispatcher is concerned about are at the same offsets within those two blocks, except as noted below, the dispatcher makes no distinction between the two types of deferred service request. For the normal case, when the system is not extending, the dispatcher loads R11 from IOBUSER in the IOBLOK, R12 from IOBIRA in the IOBLOK, and the CPU timer from the VMTTIME field of the VMBLOK pointed to by R11. With R10 pointing to the IOBLOK (or TRQBLOK), it then branches to the interrupt processing routine whose address is in R12. Note that the interrupt routine gains control with only R10-R12 set to defined values, and that the interrupt routine is responsible for releasing the storage occupied by the IOBLOK/TRQBLOK.

When the system is extending, only IOBLOKs for paging tasks (IOBPAG in IOBFLAG is set) are unstacked. IOBLOKs and TRQBLOKs are distinguished by the fact that the uppermost bit in the TOD clock value is set on for the next 60-odd years; since the corresponding bit position in an IOBLOK is always zero, a simple test prevents TRQBLOKs from being unstacked during periods when the system is extending. This kludge should work until about the year 2043, when the TOD clock wraps.

4.7.2 Unstack and dispatch CPEXBLOKs

If there are no IOBLOKs or TRQBLOKs to unstack, the dispatcher performs a SSM ENABLE/DISABLE sequence to allow any hardware stacked interrupts (both external and I/O) to occur. Assuming the system is not extending, the first CPEXBLOK is removed from the queue (at label DMKDSPRQ). The full set of 16 registers is loaded from the CPEXBLOK; the

storage occupied by the CPEXBLOK is released (with appropriate contortions to preserve the register contents); the condition code is set by doing an LTR on R15 (loaded from CPEXADD in the CPEXBLOK) and the dispatcher exits by branching to the address in R15.

If the system is extending, only CPEXBLOKs that relate to paging activity are unstacked. The dispatcher contains a list of valid (paging related) exit addresses and compares CPEXADD in the CPEXBLOK to the list to see if the request should be unstacked.

If there are no CPEXBLOKs to unstack, then the dispatcher is free to run a virtual machine, unless the system was extending. For the latter case, control is passed to the code that loads a real wait PSW.

4.8 DISPATCH VIRTUAL MACHINES

The shortage of certain critical resources during extend processing means that virtual machines should not be dispatched at this time. Any of these conditions cause a branch of control to the code that loads a real wait PSW.

4.8.1 Select highest priority ready virtual machine

The list of dispatchable virtual machines is maintained by the scheduler and pointed to by two words at the label DMKSCHRL. The dispatcher scans this "run list" for the first virtual machine that is ready to run. A virtual machine is ready to run if VMRUN in VMDSTAT is set on and no CP wait flags in VMRSTAT are set. A fast re-dispatch path is taken for the virtual machine if the following conditions hold:

1. The selected virtual machine is the one that was running before an interrupt caused control to return to CP.

2. There is a problem state PSW in the I/O old PSW field, indicating that CP was entered due to an I/O interruption.

3. The virtual machine is still eligible for fast re-dispatch as indicated by VMDSP in VMDSTAT being set on.

The ECPS microcode does not implement this particular fast re-dispatch path.

4.8.2 Clean-up after previous VM if not current

If the selected virtual machine is not the same as the one previously dispatched, and the previous machine was using at least one shared system, DMKVMASH must be called to verify that no shared pages have been altered. The newly selected virtual machine is allocated a fresh quantum of time in the location 80 timer. The standard quantum size is maintained in DMKDSPQS; however, if the virtual machine has already experienced a quantum end (VMCOMP in VMQLEVEL is set on) then the size of the allocated quantum is multiplied by 4 (so that long running compute-bound tasks require less system overhead). The floating point registers are also loaded at this point. The fast re-dispatch paths do NOT restore the floating point registers.

4.8.3 Setup for dispatching a virtual machine

The code to load the real control registers and PSW with the proper information to run a virtual machine has never been easy, but the addition of the microcode assists (VMA, ECPS, and 370E) along with features like single processor mode (SPMODE) have made this complex piece of code almost impossible to understand. Suffice it to state that the contents of the new PSW and control registers are constructed piecemeal, with most of the code being dedicated to setting the correct bits in CR6 for the microcode assist options. For 4300-class processors this section of code is microcode assisted and, since options like SPMODE are not available on those machines, the microcode implementing this path can be greatly simplified. At any rate, the control registers get set up, the timers get loaded, the general purpose registers get restored, and finally the LPSW is executed to dispatch the virtual machine.

4.8.4 Fast reflect dispatching path

Normal entry to the fast re-dispatch entry of the dispatcher (DMKDSPA) assumes that the running virtual machine attempted a privileged operation that has now been simulated and the virtual PSW with which to dispatch the virtual machine is in the real program check old PSW field. Therefore, by making a few simple tests to verify that no major status change has occurred either to the system (not extending?) or to the virtual machine (VMDSP still set?), it is necessary only to update the virtual location 80 timer (if the timer is running and microcode assist is not available) and re-dispatch the virtual machine with a very minimum of setup. Of course, if any of the tests for using the fast re-dispatch

path fail, the long path through the main entry point is taken.

One additional possibility exists for skipping the majority of the virtual machine setup code. As mentioned above, when the previously running virtual machine is about to be re-dispatched, it is eligible for fast re-dispatch (VMDSP is still set), and the PSW with which to resume its execution is in the I/O old PSW field, then the virtual machine is dispatched along the fast re-dispatch path without going through the full setup.

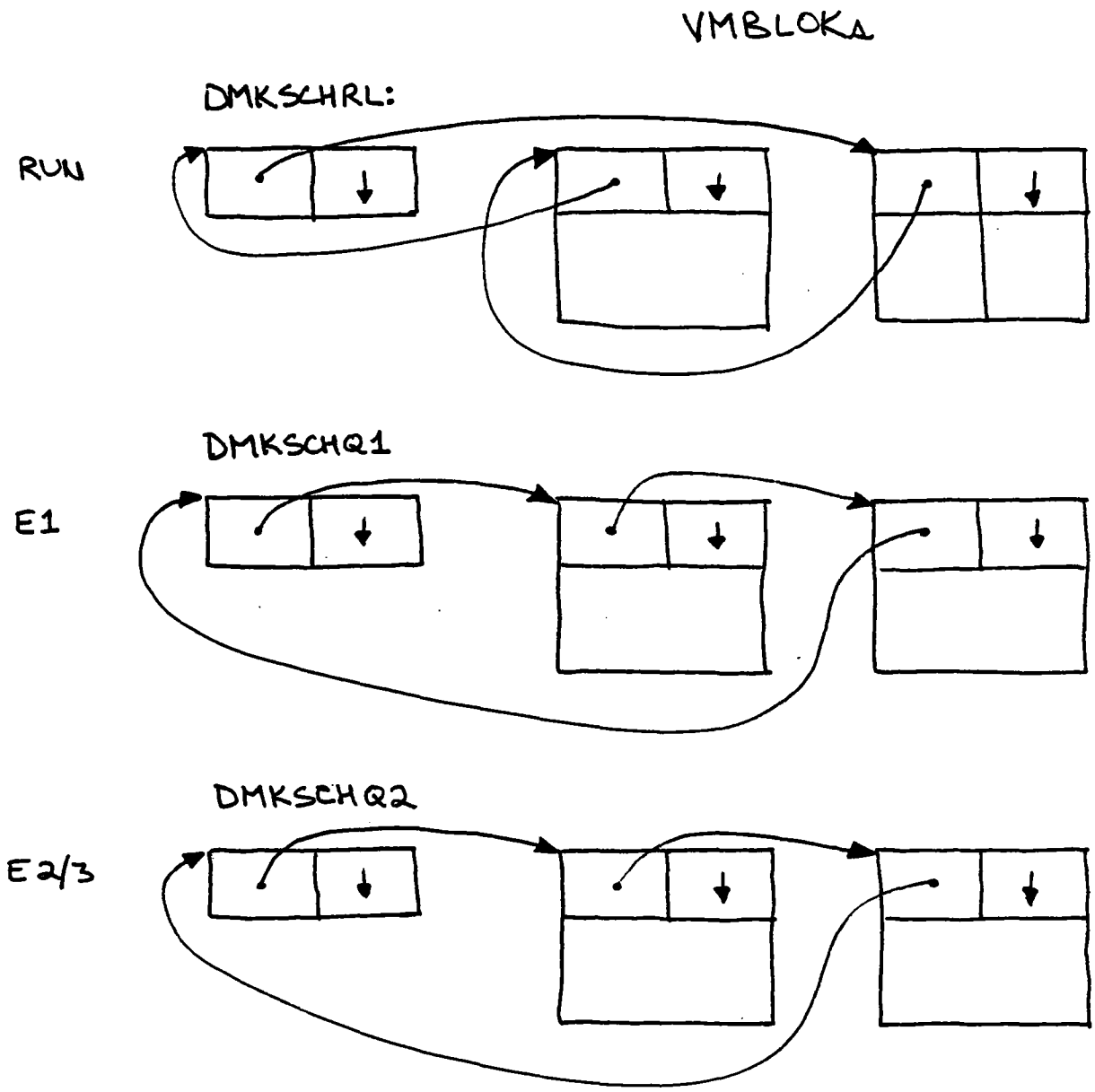
4.9 WAIT TIME ACCOUNTING

If the dispatcher searches through all its queues and cannot find any work to perform, it branches to the section of code that loads a real wait PSW. However, as mentioned at the beginning of this chapter, when the system comes out of wait state the amount of wait time is decremented from one of three "timers" depending on the reason for the wait: really nothing to do (IDLEWAIT), or the system was bottlenecked due to high paging activity (PAGEWAIT), or high I/O activity (IONTWAIT).

The wait time is classified as IDLEWAIT if there are no virtual machines on the run list, or if none of the virtual machines on the run list is in page wait or I/O wait (as indicated by VMPGWAIT or VMIOWAIT in the VMRSTAT field of their VMBLOK). If more than half the pageable page frames are allocated to virtual machines in the run list that are in page wait (VMPGWAIT in VMRSTAT is set on) or if there are virtual machines in page wait but none in I/O wait (VMIOWAIT in VMRSTAT is set on), then the wait time is classified as PAGEWAIT; otherwise, the wait time is considered IONTWAIT.

Once the classification is performed, the CPU timer is loaded with the appropriate time value and flags are set in the PSA in the field CPSTAT3 to indicate to which field the timer value should be returned when the system leaves wait state. Finally, after setting up CR0 for possible MP signal events, the wait PSW is loaded.

NOTES



SCHEDULÉR OVERVIEW

Chapter 5

SCHEDULER

5.1 INTRODUCTION

5.1.1 Overview

The purpose of a scheduler is to make resource allocation decisions that maximize the throughput of the system and minimize response times. Achievement of these goals must be subject to installation-specified constraints on the relative rates various virtual machines should receive resources. Since these goals are often in direct conflict with each other, this chapter explains the algorithms used by DMKSCH to implement and to resolve apparent conflicts in resource allocation policy.

5.1.1.1 Good response time

By far the majority of interactive transactions consume very little resource. The CP scheduler employs several mechanisms to give short transactions (and the initial parts of long transactions) preferential treatment at two critical times, when the check is made to see if the virtual machine's working set fits into the page frames available, and when the internal priority is calculated. To improve the human factors associated with using the system, virtual machines performing console I/O are usually given the same preferential treatment. The VMQ1 flag in the VMQLEVEL field of the VMBLOK indicates that a virtual machine is receiving this type of preferential treatment.

5.1.1.2 Maximize throughput

Resource contention is the major reason that most computing systems do not perform at full capacity. Historically, the paging and memory subsystems have been a major source of contention in timesharing systems. The CP scheduler is no exception to the long list of timesharing schedulers that make careful working set calculations in an attempt to improve system throughput by avoiding "thrashing".

One source of increased contention, and decreased throughput, is the loading and unloading of pages for large or long transactions. To improve throughput, the scheduler implements a special algorithm for virtual machines running large or long transactions. The execution of this algorithm has the result of loading and unloading the machines' pages one-eighth as often, but it runs the machines eight times as long while they are in the run list. Various adjustments are made to the run list priority of these machines to prevent them from depriving more interactive machines of resources. They are also eligible for pre-emption from the run list if an interactive machine requires their page frames to be able to run.

One other well-known way to improve throughput is to insure that compute-bound virtual machines get control of the CPU after the I/O-bound machines have started their I/O and are waiting for it to complete. CP uses the location 80 timer to determine when a particular virtual machine may be "hogging" the CPU. The scheduler attempts to improve throughput by moving such machines to a position that is lower in the run list and setting the VMCOMP flag to signal the dispatcher that this virtual machine is compute-bound and should receive a longer time slice.

5.1.1.3 Relative resource consumption rate

The scheduler dynamically calculates each virtual machine's use and "fair share" of the CPU resource. Virtual machines that are receiving more or less than their fair share of the CPU time receive appropriate adjustments to their internal priority until their resource consumption rate is on target. A virtual machine's relative fair share of CPU is controlled by its CP directory priority, with a priority of 64 (in the range of 0-99) being considered the "normal" priority value.

5.1.2 References

5.1.2.1 Publications

1. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

2. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

3. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

4. Young, C. J., *VM/370 Biased Scheduler, Release 1 PLC 9*, TR 75.0001, IBM Development Center, Burlington, MA, August, 1973.

5. Schatzoff, M. and L. H. Wheeler, *CP-67 Paging Priority Dispatcher*, IBM Cambridge Scientific Center Technical Report No. G320-2088, March, 1973

6. Wheeler, L. H., *C. S. C. VM/370 Extended I, Dispatching and Scheduling*, IBM Cambridge Scientific Center Technical Report No. ZZ20-6001, July, 1974.

7. Cogger, R. and R. Cowles, *VM Scheduler, A White Paper*, Proceedings of SHARE XLVI, Feb. 1976.

8. VanLeer, Paul, *The Truth about the Fair Share Scheduler*, Proceedings of SHARE LV, Vol. 2, August, 1980.

5.1.2.2 CP modules

1. DMKSCH - is the "boss" who makes the decisions.

2. DMKSTP - periodically looks around the system and calculates new averages for various control values that the scheduler uses. The TRQBLOK is rescheduled to expire at a later time, just like a snooze alarm.

3. DMKCPI - schedules the TRQBLOK that triggers the timer driven calculations performed by DMKSTP and calls an entry point in DMKSTP to initialize certain fields.

4. DMKDSP - determines when a virtual machine has entered a "long wait" condition and also performs the majority of calls to the scheduler to check for a change in status of the virtual machine.

5.2 VM SCHEDULER TERMINOLOGY

The scheduler maintains several flags in the VMBLOK that summarize the state of the virtual machine the previous time the scheduler was called. By examining the current state of the virtual machine (determined primarily by the flags in VMRSTAT) and its previous state (determined primarily by the VMRUN and VMINQ flags in VMDSTAT), the scheduler can determine if any important status change has occurred and take actions it deems necessary. The scheduler maintains three queues of virtual machines, the run list and two eligible lists. At some level, the job of the scheduler can be

viewed as one of moving virtual machines among the queues and ordering the queues in a fashion that implements policies concerning the relative importance of throughput, response time, and fair distribution (or appropriate unequal distribution) of resources.

5.2.1 In-queue versus in a queue

For mostly historical reasons, a virtual machine that is in the run list is said to be "in-queue" (and the VMINQ flag is on). In earlier versions of the VM scheduler, the virtual machines on the run list were a subset of the "in-queue" virtual machines. Only runnable virtual machines were allowed in the run list (makes sense, doesn't it?). However, that bit of consistency was dropped in an effort to cut down on the frequency with which machines had to be added or dropped from the run list. Note that the "in-queue" condition is not the same as "in a queue". The latter means that a virtual machine is in the run list or one of the eligible lists, implying that one or both of the flags VMRUN or VMINQ are set on.

5.2.2 Time slice end vs. queue slice end

As explained above in the introduction and in the chapter on the dispatcher, the location 80 interval timer is used to help prevent a single virtual machine from blocking other run list virtual machines from accessing the CPU resource. The initial interval timer value for a time slice (or "quantum" as it is often called) is maintained in the dispatcher at label DMKDSPQS ("quantum size"). On the other hand, the queue slice is the maximum amount of virtual *or* overhead time that a virtual machine is allowed to consume before being dropped from the run list (in order to allow other virtual machines to be added to the run list.) The queue slice values are maintained in the scheduler at labels DMKSCHQ1 and DMKSCHQ2 for the interactive and "less interactive" queue slices. Initially, DMKSCHQ1 is eight times the time value of DMKDSPQS, and DMKSCHQ2 is eight times the value of DMKSCHQ1. While the virtual machine is dispatched, the virtual CPU time remaining in the queue slice is normally kept in the CPU timer. When looking at CP code, be sure to look at the instructions and not just at the comments; many of the comments are incorrect when referring to the meaning of VMQSEND (queue slice end) and VMTSEND (time slice end).

5.2.3 Q1, Q2, and pseudo-Q3

The terms referring to the "Qn" for a virtual machine are mostly incorrect or at least misleading. A virtual machine is said to be in "Q1" when it is in the run list *or* the Q1 eligible list and is distinguished by having the VMQ1 flag set in VMQLEVEL.

A "Q2" virtual machine is basically one that is not in the high priority interactive queue (Q1) and it has not (yet) reached the status of a long-running transaction (Q3). Both Q2 and Q3 transactions wait on the Q2 eligible list until sufficient page frames are available for them to run.

Once a virtual machine running a transaction completes six consecutive Q2 slices, it is "moved" to Q3 (pseudo-Q3). The idea behind Q3 is to cut down on overhead by running the really long transactions for eight times as long while they are in the run list but to run them only one-eighth as often. Q3 is implemented by means of some flags, a counter (VMQ3CNT) containing the number of Q2 slices remaining in the Q3 slice, and a little bit of extra code to recognize a Q3 virtual machine and not drop it completely from the run list until its quota of Q2 slices is exhausted.

5.2.4 Run list

The run list contains those virtual machines whose working sets can be contained within the available page frames. A virtual machine must be on the run list to receive system resources for any purpose except CP commands entered through the console. A virtual machine in the run list may or may not be runnable, therefore it is distinguished from other virtual machines by having the VMINQ flag of VMDSTAT set on. The *priority* of the virtual machine within the run list is the time at which it would have received a queue slice under the assumption that the CPU time was received at the "fair share" rate for that virtual machine. The queue anchor for the run list is at label DMKSCHRL.

5.2.5 Eligible lists

The eligible lists contain those virtual machines that are runnable but whose working sets cannot fit into the available page frames. The scheduler ensures that if a machine is runnable, then it is in either the run list or an eligible list; therefore, any virtual machine with VMINQ set off and VMRUN set on in VMDSTAT must be in one of the eligible lists. The state of the VMQ1 flag in VMQLEVEL determines

which eligible list the virtual machine is on. The queue anchors for the eligible lists are in the control blocks of DMKSCH that are mapped by the VMQBLOK DSECT. The VMQBLOKs are labelled DMKSCHQ1 and DMKSCHQ2 and contain the length of the queue slice (in timer units) and other counters and statistics of importance to the scheduler.

5.2.6 Drop from queue

A virtual machine is removed from the run list when it completes a queue slice, enters CP console function mode (CP READ), or loads a wait state PSW when it has no high-speed (disk or tape) I/O outstanding. When a virtual machine is removed from the run list it is "dropped from queue".

5.3 CHECK VIRTUAL MACHINE FOR STATUS CHANGE

The virtual machine for which the scheduler is called may be in one of several different states. The first partition of the states is based upon whether or not the virtual machine is currently runnable (VMRSTAT in the VMBLOK is zero).

5.3.1 Virtual machine is not runnable (VMRSTAT not zero)

If the virtual machine is in the run list (VMINQ in VMDSTAT is on) then a branch is taken to check for a long or idle wait condition. Otherwise, if the virtual machine was runnable on the previous call to the scheduler (VMRUN in VMDSTAT is on) then since it is not in the run list, the virtual machine must be in an eligible list. Since only runnable virtual machines are allowed in the eligible list, the current machine must be removed from the eligible list, but no other status change is required. If the virtual machine was not runnable on the previous call to the scheduler (VMRUN in VMDSTAT is off), then no status change has occurred and the scheduler exits immediately.

5.3.2 Virtual machine is runnable (VMRSTAT is zero)

If the virtual machine was runnable on the previous call to the scheduler (VMRUN in VMDSTAT is on) and it is not in the run list (VMINQ in VMDSTAT is off), then the machine is already in an eligible list and no status change has occurred and the scheduler exits immediately. If the virtual machine is not in the run list and was not runnable on the previous

call to the scheduler (VMINQ and VMRUN in VMDSTAT are both off), then the virtual machine must be added to an eligible list (see the discussion later in this chapter). The final alternatives apply to a virtual machine that is runnable and in the run list. If the virtual machine was runnable on the previous call to the scheduler then a check is made to see if a time slice has expired while the virtual machine was running; otherwise, the scheduler checks whether the virtual machine has exceeded its queue slice.

5.3.3 Time slice end

The time slice ("quantum") is the amount of CPU time that a virtual machine is allowed to use without another virtual machine being dispatched. Recall that the dispatcher allocates a new time slice by setting the real location 80 timer to the value in DMKDSPQS whenever a new virtual machine is dispatched. A time slice end indicates that the virtual machine has used a significant quantity of CPU while not allowing any virtual machines of lower priority to be dispatched; in other words, the virtual machine is probably compute-bound.

When a time slice end occurs (VMTSEND in VMDSTAT is on), the scheduler temporarily alters the run list priority of the offending virtual machine so that it is moved lower in the run list. The amount of the change in priority is 1/4 of the time remaining until the machine is scheduled to drop from the run list (VMEPRIOR is the scheduled time for the drop). Since the change in VMEPRIOR is only temporary, the run list is, in this case, not maintained in strict priority order.

5.3.4 Queue slice end

When a virtual machine exceeds its queue slice (VMQSEND in VMDSTAT is set), it is dropped from the run list and added to the appropriate eligible list (assuming it is currently runnable). However, virtual machines that are running long transactions and are therefore in the pseudo-Q3 go through a full drop from the run list only when the number of Q2 slices allowed in Q3 (VMQ3CNT) is decremented to zero.

5.3.5 Terminal I/O

Virtual machines that have recently performed I/O to their consoles are indicated by having VMPRIDSP set in VMQSTAT. Since console I/O is normally indicative of the start of a new transaction (and most transactions are short) or of the output generated at the end of a transaction, the scheduler attempts to insure rapid processing of the transaction by placing the virtual machine in Q1 if it is not already there. If the virtual machine is in an eligible list, it is dropped and control is passed to the routine that performs the Q1 priority calculation before placing the virtual machine in the Q1 eligible list. If the virtual machine is currently in the run list, it is dropped and added to the Q1 eligible list only if the deadline by which it should have been dropped has passed by more than one second. This last test allows a Q1 priority slice to be delivered to a virtual machine that is otherwise "buried" at the bottom of the run list and that might otherwise take a relatively long time to complete a queue slice.

5.3.6 Long wait

Any virtual machine that is detected to be waiting for an event taking longer than a normal disk or tape operation is dropped from the run list. The scheduler always drops virtual machines from the run list when they have VMIDLE, VMLOGON, or VMLOGOFF set in VMRSTAT. A virtual machine in console function wait (VMCFWAIT in VMRSTAT set) is also considered to be in long wait unless the machine is executing a virtual console function (VMVIRCF and VMCF in VMDSTAT set) and the console function is waiting on a page (VMPGWAIT in VMRSTAT set). After dropping the virtual machine from the run list, the scheduler checks the eligible list for virtual machines that may now fit into the available page frames.

5.3.7 Delayed queue drop and paging checkpoint

When the virtual machine's total number of page reads (VMPGREAD) exceeds its paging checkpoint value (VMPCKP), the virtual machine is checked for a "delayed queue drop" request (only possible for pseudo-Q3) and has its working set adjusted if previous projections appear to be incorrect.

As discussed later in this chapter, the scheduler overcommits the real memory frames in order to add certain virtual machines to the run list (e.g. Q1 virtual machines). To ensure that this overcommitment does not lead to long term thrashing, at the time of overcommitment a Q3 virtual

machine is flagged for delayed queue drop (VMDLDRP in VMQSTAT). When the paging checkpoint is reached, a virtual machine with VMDLDRP set is dropped from the run list if the number of available page frames exceeds the sum of the working sets of machines in the run list (PAGUSAGE) by 12.5%.

As programs running within a transaction go through different phases, the working set of a virtual machine can undergo significant changes in size. To track such changes, at a paging checkpoint the scheduler adjusts the working set size if the number of resident pages (VMPAGES) is larger than the projected working set size (VMWSPROJ) plus 16. The adjustment is performed by averaging the VMWSPROJ and VMPAGES to produce a new VMWSPROJ.

Whether or not the virtual machine's working set projection is adjusted, the next paging checkpoint must be set up. The minimum of the projected working set size (VMWSPROJ) and 16 is added to VMPGREAD to be stored in VMPCKP as its next checkpoint.

5.3.8 Pre-emption of pseudo-Q3 virtual machines

When a virtual machine running in pseudo-Q3 reaches the end of one of its Q2 slices, the scheduler checks the eligible lists for interactive virtual machines that have already been delayed past their deadline. If such a machine is found on the eligible list and has been waiting on the list, the pseudo-Q3 virtual machine is dropped from the run list to make room for the more interactive machine. The actual logic that performs the tests is quite a bit more complicated than described here and is covered in more detail in the section dealing with selection of virtual machines from the eligible list.

5.4 MAINTENANCE OF THE SCHEDULER QUEUES AND STATISTICS

In the previous section the major concern was to look for a status change that implies the scheduler should *do* something. The next section discusses the things that the scheduler does, namely, maintaining the scheduler and dispatcher queues and collecting statistics.

new functions to be supported. Figure 2 shows the format of the EC mode PSW, which is formatted as follows:

1. Bits 0-7 (the system mask) are redefined. Bit 1 enables program event recording (PER). Bit 5 enables dynamic address translation, which will be described later. Bit 6 enables I/O interrupts from any channel that is also enabled in CR2. Bit 7 enables any external interrupt that is also enabled in CR0.

2. Bits 8-15 are the same as in BC mode.

3. Bits 18-23 contain the condition code and program mask fields (bits 34-39 in BC mode).

4. Bits 24-31 and 32-39 are reserved and must be 0.

5. Bits 40-63 are the instruction counter, as in BC mode.

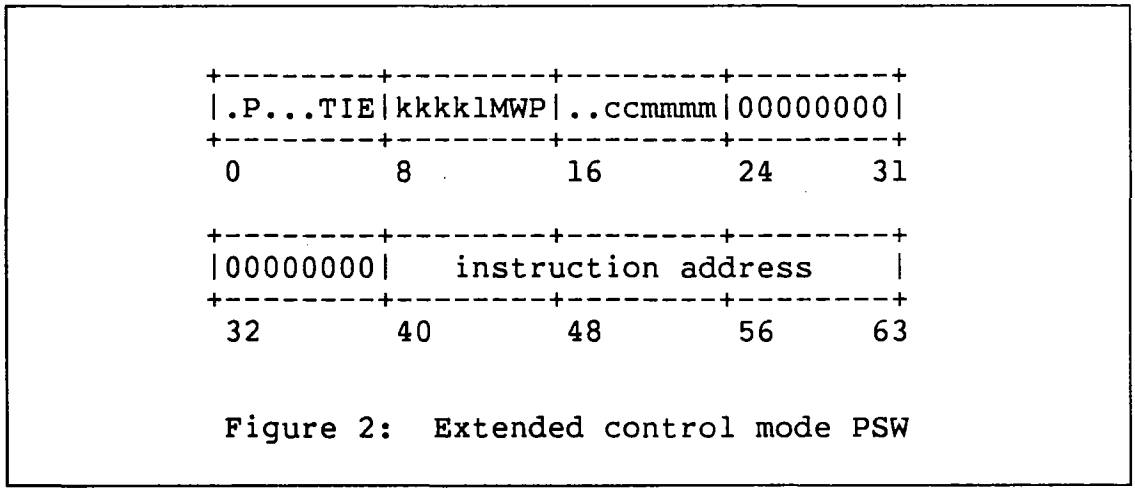


Figure 2: Extended control mode PSW

1.3.2.3 Swapping

The CPU changes status from time to time by storing the current PSW into main storage and fetching a new PSW from somewhere else in main storage. Such PSW swapping may of course change any or all of the various fields in the PSW and usually indicates a change from a user program to a system program. In addition to swapping PSWs, the CPU can also simply load a new PSW, in which case the old PSW is lost. The Load PSW (LPSW) instruction is available only in supervisor state and is the usual way for a system program to return to an interrupted user program.

5.4.1 Working set size prediction

The use of a feedback loop to help calculate a predicted working set size is a central feature of the VM/SP scheduler. The code implementing the working set size prediction is fairly difficult to understand. It is highly recommended that the paper written by Paul Van Leer and found in the *Proceedings of SHARE LV* (see references) be used in conjunction with a listing of DMKSCH if an in-depth understanding of this topic is desired.

5.4.1.1 Resident pages averaged over page reads

The paging subsystem maintains several VMBLOK values of interest to the scheduler.

1. VMXPG - is the maximum number of resident pages during a stay in the run list.

2. VMRDINQ - is the number of page reads performed for this virtual machine during the most recent stay in the run list.

3. VMPAGES - is the current number of resident pages for the virtual machine.

4. VMPGRINQ - is the sum of the number of pages currently resident for a virtual machine at each page read.

One method of calculating the average number of resident pages is to merely divide the sum of the resident pages by the number of page reads. Performing that calculation, one arrives at the average number of resident pages PER PAGE READ, not per unit of time; the time-averaged value for resident pages may be as much as twice this value.

The scheduler uses various heuristics to determine what adjustments have to be made to arrive at a reasonable value for the average number of resident pages. The two extreme case for adjustment are:

1. A virtual machine that starts with no resident pages, reads 20 pages immediately, and then completes the slice without reading additional pages. For this virtual machine the average number of resident pages is 20 when averaged over time and 10 when averaged over page reads.

2. A virtual machine that starts with 20 resident pages, reads a few pages during its slice, and (possibly) has a few pages stolen from it during the slice so

that it has about 20 pages resident at the end of the slice. For this virtual machine, the average number of resident pages is 20 when averaged over either time or page reads.

The scheduler attempts to distinguish between these cases (and several others) and to arrive at a reasonable value for the average number of resident pages using information it can glean from the VMBLOK fields mentioned above. For a more detailed explanation, refer to the SHARE presentation given by Paul VanLeer (see the references).

5.4.1.2 Resident pages averaged over CPU use

One serious concern throughout the scheduler algorithms is the amount of productive CPU time (CPU used for functions other than paging) that is delivered between page reads. The system-wide average for the amount of productive CPU time delivered per page read is maintained in DMKSCHKA. The ratio of the virtual machine's CPU time between page reads and the average for the system is used to adjust the value for average resident pages. It is biased upward for virtual machines using less than the average amount of CPU time between page reads; the implication being that increasing the number of resident pages would have increased the amount of productive CPU time between page reads.

5.4.1.3 Prediction of new working set size

Except for a number of special cases that must be considered, the new projected working set size is taken to be the square root of the product of the two forms for the average resident pages calculated above.

5.4.2 Calculation of queue priority

The priority that determines the ordering of the run list and the eligible lists is considered to be the deadline for the virtual machine to receive the CPU time within its queue slice. As a partial TOD clock value, the priority has a precision of approximately 30 milliseconds, is based at the time of IPL, and can run for approximately 14 years before going negative (probably longer than the system will run before going down). The priority is calculated by adding several "expected delay" factors to the current TOD to arrive at a new deadline. Several important factors in computing the amount of delay a virtual machine will experience are discussed below.

5.4.2.1 Calculate bias due to external priority

The CP directory and the privileged SET PRIORITY command allow the specification of an external priority for a virtual machine. The scheduler uses this external priority in determining the relative importance of the virtual machine by adjusting the virtual machine's expected CPU use delay with a bias factor. Table 7 indicates the bias factor resulting from a number of different values for the external priority.

TABLE 7

Effect of external priority on CPU delay factor

External Priority	Bias Factor for CPU
0	1
11	2
21	4
32	8
43	16
54	32
64	64
75	128
85	256
96	512

The interpretation of the numbers is that the expected delay for a virtual machine at priority 64 is 32 times the delay for an equivalent virtual machine running at priority 11. After appropriate scaling, the product of this bias and the average delay for virtual machines at the standard priority of 64 is used as a major part of the deadline priority calculation.

5.4.2.2 Calculate CPU use delay

System-wide averages are maintained on the recent averages for certain values, namely, the amount CPU time used while on the run list and the length of stay on the run list. The scheduler calculates the amount of time the virtual machine will spend on the run list just to receive a queue slice of

CPU time given the bias resulting from the external priority.

5.4.2.3 Combine delay factors for priority

Although only the calculation for the delay in CPU use was mentioned above, a calculation for the delay due to paging activity is also performed. These delays are combined and then adjustments are made in order to bias the priority in favor of virtual machines running shorter transactions. The amount of the bias is largely a function of how many Q2 slices have already been completed by the transaction.

5.4.3 Add to eligible lists

5.4.3.1 Necessary conditions

Any virtual machine that is runnable and not currently in the run list is added to an eligible list. An exception to this rule is the case of "assured execution" machines, those that have been SET FAVORED and therefore have the VMAEX flag set in VMQLEVEL. Virtual machines that are favored are never placed in the eligible list but instead proceed directly to the run list.

5.4.3.2 Necessary actions

1. Timestamp the entry to the eligible list by storing the TOD clock into VMTODINQ.

2. If the difference between the previous timestamp (when the virtual machine dropped from a list) and the current timestamp is greater than about 5 seconds, then the count of consecutive Q2 slices is set to zero.

3. If the VMQ1 flag in VMQLEVEL is set the virtual machine is added to the Q1 eligible list; otherwise, the virtual machine is added to the Q2 eligible list. VMEPRIOR is used as the priority to determine the point of insertion within the list; the list pointers are chained through VMQFPNT and VMQBPNT at the beginning of the VMBLOK.

4. A Monitor Call instruction is issued to allow the (optional) collection of data about virtual machines being added to the eligible lists.

5.4.4 Eligible list selection algorithm

Whenever the set of virtual machines in the eligible lists or in the run list changes, the scheduler checks the eligible lists for virtual machines that may now be added to the run list.

5.4.4.1 Add virtual machines that fit in run list

The standard algorithm for adding virtual machines to the run list is to add them in priority order, first from the Q1 eligible list and then from the Q2 eligible list, until the addition of another machine would cause the sum of the projected working set sizes for the machines in the run list to exceed the number of page frames available.

5.4.4.2 Special checks

While virtual machines are being selected for addition to the run list, additional checks as outlined below are performed.

1. Prevent Q2 lockout by Q1 virtual machines. This check is performed whenever Q1 virtual machines are being selected for the run list. If the highest priority virtual machine in the Q2 eligible list has a priority that is better than the highest priority Q1 machine, and if the difference between the priorities is more than one-eighth of the average eligible list delay (maintained in DMKSCHET by DMKSTP), then a Q2 lockout condition is considered to exist and Q1 virtual machines are not considered for adding to the run list.

2. Overcommit available pages by 12.5% for Q1 virtual machines.

3. Preempt Q3 machines to make room by scheduling them for a delayed drop from the run list (VMDLDRP in VMQSTAT set). Virtual machines preempted in this fashion are set up for a paging checkpoint when they have performed five additional page read operations.

   a) The system must not be running heavily in problem state for it to overcommit memory by this method.

   b) The preempting machine must be past its deadline for the current queue slice.

c) The preempting machine must have an earlier deadline than the machine being preempted.

d) The preempted machine must not be favored.

e) The preempted machine must be in Q3 and have executed at least one of its Q2 slices.

f) The preempted machine must not already be flagged for a delayed drop. If the machine is so flagged, its working set size is discounted from the number of pages in use.

4. If attempting to add a Q1 virtual machine and there are no Q1 virtual machines in the run list, allow the available pages to be overcommitted by 50%.

5. If the number of virtual machines in the run list is less than 4, check for large virtual machines that are causing the low multiprogramming level. If the current virtual machine has a working set greater than twice the average working set or greater than one third of the available pages, it is considered large. If the next virtual machine on the eligible list has a deadline that is close to that of the current virtual machine, the current virtual machine is bypassed and lower priority virtual machines may be added to the run list.

5.4.5 Drop from eligible lists

5.4.5.1 Sufficient conditions

A virtual machine is dropped from an eligible list for one of three reasons:

1. The virtual machine is not currently runnable.

2. The virtual machine has been selected for addition to the run list.

3. The virtual machine has performed I/O to its console and it is being dropped so that it may be re-added to the Q1 eligible list.

5.4.5.2 Necessary actions

The virtual machine is removed from the eligible list and the VMRUN flag is reset to indicate that it is no longer in one of the lists.

5.4.6 Add to the list of dispatchable machines

Because of certain functions, such as the drop that occurs between Q2 slices for the pseudo-Q3 machines, adding virtual machines to the run list is separated into two routines and their functions are referred to as "add to queue" and "add to run list".

5.4.6.1 Add to queue

This routine performs most housekeeping functions necessary when a virtual machine is added to the run list.

1. Timestamp the VMBLOK with time of entry to run list. Update statistics, maintained by the scheduler, on the total amount of time virtual machines are spending in the eligible lists.

2. Perform necessary maintenance of virtual machine timers. The details of actions performed here are covered in the chapter on timer maintenance.

3. Initialize the paging counters.

   a) VMRDINQ is set to the current number of page reads (VMPGREAD).

   b) VMPGRINQ and VMXPG (maximum number of pages resident) are set to VMPAGES, the current number of pages resident for the virtual machine.

   c) The number of pages stolen while in the run list (VMSTEALS) is cleared to zero.

4. Add the projected working set size of this virtual machine to the working set sum of machines in the run list (PAGUSAGE).

5. Set the next paging checkpoint to occur when the virtual machine has read a number of pages equal to its working set size (or 16, whichever is larger).

5.4.6.2 Add to run list

Based upon the value of VMEPRIOR, the VMBLOK is inserted at the appropriate spot in the run list and the VMINQ and VMRUN flags are set on.

5.4.7 Drop from the list of dispatchable machines

Just as there are routines for adding machines to the dispatch list, there are also several routines for removing machines. The routine that removes the VMBLOK from the run list is basically the same one that removes a VMBLOK from an eligible list and has already been discussed. There are two routines used to "drop from queue", one for the "pseudo-drop" performed on Q3 transactions between each of the Q2 slices comprising the Q3 slice and another for the normal drop from queue that is performed for each virtual machine at the end of its queue slice.

5.4.7.1 Perform pseudo drop and add for Q3

The following actions are performed for Q3 virtual machines when they exceed the in-queue limits for a Q2 slice, but the field VMQ3CNT indicates that more Q2 slices remain within the Q3 slice.

1. Adjust the virtual machine's priority.

   a) Subtract the current TOD from the deadline to determine the amount of time the virtual machine is ahead of schedule (minimum of zero).

   b) Divide the amount ahead of schedule by two and add it and the current TOD to the expected amount of time in the run list for this virtual machine to complete a Q2 slice (VMQPRIOR). The result is stored in VMEPRIOR and is a new deadline for this virtual machine to complete its next Q2 slice.

2. Update the timers associated with the virtual machine and maintain various CPU utilization statistics. Much of this work is covered in the chapter on timer maintenance, so the finer details are left until then.

   a) Update VMVTIME to include all virtual CPU time received while in the run list.

   b) Calculate the amount of overhead CPU time used in the queue slice, add to the virtual CPU time used in the queue slice, and update the CPU utilization statistics for the virtual machine (VMUHS).

   c) If the virtual machine did not achieve the system average for the amount of CPU used between page reads, then it is penalized by having an additional amount added to its current CPU utilization value (VMUHS).

3. Initialize the paging counters.

   a) VMRDINQ is set to the current number of page reads (VMPGREAD).

   b) VMPGRINQ and VMXPG (maximum number of pages resident) are set to VMPAGES, the current number of pages resident for the virtual machine.

4. The average of the current number of pages resident (VMPAGES) and the projected working set size (VMWSPROJ) is calculated and becomes the new projected working set size.

5. The new paging checkpoint value (VMPCKP) is set to be the projected working size (but at least 16) plus the current number of page reads (VMPGREAD).

5.4.7.2 Perform full drop from queue

Much of the code executed in this section of the scheduler performs the working set size projection and the calculation of the virtual machine priority based on the transaction characteristics during the just completed queue slice. Other functions performed are described below.

1. Remove the working set size of this virtual machine from the sum of working sets of virtual machines in the run list.

2. Save statistics about the elapsed time the virtual machine was in the run list. Also, timestamp the VMTODINQ field with the time of drop from the run list.

3. Update the timers associated with the virtual machine and maintain various CPU utilization statistics. Much of the work necessary is covered in the chapter on timer maintenance, so the finer details are left until then.

   a) Update VMVTIME to include all virtual CPU time received while in the run list.

   b) Calculate the amount of overhead CPU time used in the queue slice, add to the virtual CPU time used in the queue slice, and update the CPU utilization statistics for the virtual machine (VMUHS).

   c) If the virtual machine did not achieve the system average for the amount of CPU used between page

reads, then it is penalized by having an additional amount added to its current CPU utilization value (VMUHS).

4. For non-Q1 transactions, increment the count of consecutive Q2 slices (VMQ2CNT).

5. For non-Q3 transactions, clear the count of Q2 slices remaining in the Q3 slice.

6. Issue the Monitor Call instruction so that statistics may be (optionally) collected about virtual machines that drop from the run list.

5.5 SCHEDULER EXIT PROCESSING

Before the scheduler exits, it reloads R11 (the VMBLOK pointer) and the CPU timer so as not to surprise the caller. If the virtual machine for which the scheduler was entered was dropped from the run list, DMKPTRRS is called to place the virtual machine's page frames on the "flush list" (see the paging chapter). Under any of the conditions listed below, the call to DMKPTRRS is not made.

1. The virtual machine is running in the V=R area.

2. The virtual machine has the SET QDROP OFF option.

3. The virtual machine has been re-added to the run list, and the system is not experiencing heavy paging activity as indicated by the upper byte of DMKPTRXX (see the paging chapter).

5.6 TUNING OPTIONS

5.6.1 SRM command

Various "knobs" are available to control how the scheduler allocates resources between various types of transactions. Access to these knobs is through the QUERY SRM and SET SRM commands.

5.6.1.1 APAGES

The number of page frames currently available for paging (DMKDSPNP) can be queried and altered. In many cases, raising APAGES causes increased paging but cuts down on the time

There are 6 conditions under which a PSW swap can occur. The following is a list of these conditions along with the locations of the old (stored) PSW and the new (fetched) PSW:

1. Restart: (new location: X'0'; old location: X'8') A restart interrupt is taken whenever the system operator presses the "restart" button on the system console. CP processes this interrupt by stopping all processing immediately and taking a memory dump; this is a PSA002 ABEND.

2. External: (old: X'18'; new: X'58') An external interrupt is taken whenever any of several events occurs. The most common external interrupts are those that are triggered by the several system timers, which are described in a later section.

3. SVC: (old: X'20'; new: X'60') An SVC interrupt is taken whenever any program executes the Supervisor Call (SVC) instruction. The second byte of the instruction is a code that is stored as the interrupt code. SVC is the only explicit means by which a problem state program can cause a supervisor state program to start execution. The interrupt code is usually set to indicate what supervisory function is desired.

4. Program check: (old: X'28'; new: X'68') A program check interrupt is taken whenever the CPU detects an error in the use of an instruction, such as an invalid address or an arithmetic overflow. Several other functions also generate program checks, such as dynamic address translation, program event recording, and monitoring.

5. Machine check: (old: X'30'; new: X'70') A machine check interrupt is taken whenever the CPU detects that a hardware error has occurred. Control is usually passed to routines that attempt to retry the failing instruction and record the error for later diagnosis and repair.

6. I/O: (old: X'38'; new: X'78') An I/O interrupt is taken whenever an I/O channel signals the CPU that some I/O operation is complete or has encountered an error. The interrupt code gives the I/O device address and the channel status word gives additional information that is discussed in a later section.

Each new PSW in CP is in EC mode and turns off address translation and I/O and external interrupts. Each IC field points to the "first level interrupt handler" (FLIH) for each of the interrupt types. When the PSWs are swapped, an

spent by virtual machines in the eligible lists. The effects on response time of altering this value are in general unpredictable since they are dependent on the characteristics of the load.

5.6.1.2 Interactive bias (IB)

The interactive bias shift value (DMKSCHIB) can be queried and altered. The value is used as a shift amount to bias priority calculations in favor of virtual machines in Q1 or for the first Q2 slice following a Q1 slice. A value for IB of 2 means that virtual machines in Q1 are projected to drop 32 times (2 ** (3 + IB)) sooner than normal, relative to the current time.

5.6.1.3 MAXWSS

The maximum working set size to be projected for any virtual machine (DMKSCHWX) can be queried and altered. The scheduler never assigns a projected working set size larger than DMKSCHWX (assuming that it is positive). When the maximum value is used to trim a working set size calculation, VMWSERNG is set on in VMQSTAT to force the virtual machine to "self-steal" once its resident page count reaches the working set size value. This parameter may be useful in cases where excessively large virtual machines are blocking other virtual machines on the eligible lists. Careful monitoring of response times and paging rates should be performed if this feature is used.

5.6.1.4 DSPSLICE

The dispatcher's value for the time slice or "quantum size" (DMKDSPQS) can be queried and altered. The initial value of the time slice is determined in DMKSTP during CP initialization and is based upon processor speed. This initial value may be altered with the effect of changing the frequency at which the scheduler may be able to detect a high priority virtual machine that is blocking other virtual machines from receiving CPU time. Some installations have reported a small variation in the initial value of DMKDSPQS; IBM evidently considers this to be normal behavior.

5.6.2 SET PRIORITY command or CP directory priority

As discussed earlier, the external priority of a virtual machine has a significant effect on the rate at which it is expected to use CPU time. Table 8 illustrates the impact of having machines at various priorities in terms of how the scheduler attempts to apportion the CPU resources. The standard value used both in the table and by the scheduler is for a virtual machine with priority 64. The values in the table represent relative CPU use, not absolute CPU use; the effects can only be observed when virtual machines of different priorities are actually competing for CPU resources. From the table you can see that a priority 64 virtual machine competing for CPU "sees" the same load if there are eight other machines running at priority 64 or one other machine running at priority 32.

TABLE 8

External priority effect on observed load

External Priority	# Equivalent VMs
0	64
11	32
21	16
32	8
43	4
54	2
64	1
75	1/2
85	1/4
96	1/8

5.6.3 SET FAVORED and SET FAVORED with percent

The SET FAVORED command allows the specification of virtual machines that are important enough never to be placed on the eligible lists. The standard deadline priority is calculated for such machines except that no eligible list delay is factored in. Normally, this option would be used in conjunction with the SET RESERVED and/or SET QDROP OFF options so that the potential effects of overloading the paging subsystem are minimized. This option is indicated by the VMAEX flag in VMQLEVEL.

The SET FAVORED with percent command allows the specification of a target level of CPU time that should be given to a given virtual machine. A virtual machine with this option set (VMAEXP in VMQLEVEL) has a deadline priority calculated assuming that it will receive CPU time at exactly the specified rate; no other special functions are provided by this option. If more than 100% of the CPU time is allocated to virtual machines, each of the favored machines simply receives proportionally less of the time. Again, this option is normally used in combination with other options to achieve the desired results.

5.6.4 SET QDROP command

This command is a rather strange recent addition to VM. The reason it is strange is that in what might be considered the normal mode of use,

SET QDROP <userid> OFF

the command doesn't do what is suggested. In this mode, the action really performed is to set a flag indicating that the virtual machine's pages should not be placed on the flush list if it is dropped from the run list; the virtual machine is still dropped from the run list when it enters a "long wait" condition. The flush list is discussed in the chapter on paging; however, the result of the above action is that the virtual machine's pages are not available for immediate reuse by other virtual machines but have to be "stolen" one at a time. Since it involves relatively high overhead, page stealing is done after the flush list is exhausted; therefore, the pages of a virtual machine with QDROP OFF are typically used after the flush list is empty, making it more likely that the pages can be reused when the virtual machine becomes runnable again.

Another form of this command really does have the expected effect. The second form of the command,

SET QDROP <userid> OFF USERS

is useful for service virtual machines using VMCF or IUCV. The effect of this command on the virtual machine for which it is issued is the same as described with the first form. The difference is that when virtual machine A communicates with virtual machine B (that has QDROP OFF USERS) via VMCF or IUCV, a counter is incremented and virtual machine A is not dropped from the run list for loading a wait PSW as long as the counter is non-zero and the machine is enabled for the appropriate kind of external interrupts. In effect, QDROP OFF USERS allows the specification of a particular

kind of wait condition that the dispatcher is to consider as a short wait rather than a long wait; when that condition is satisfied, the dispatcher does not set the VMIDLE flag in VMRSTAT when it checks out a newly loaded wait PSW. BEWARE: Some applications using VMCF keep a dummy request outstanding during the duration of a set of transactions (so that an application can determine when a central server has been reset or re-IPL'ed). If such an application is being used, the virtual machines communicating with the central server that has QDROP OFF USERS will always remain in the run list and may greatly degrade the performance of other virtual machines.

5.7 SUMMARY

The scheduler is a very complicated module and it has the unenviable task of implementing policy decisions. The policies implemented by the scheduler can be very frustrating if the management at your installation desires different or additional policies to be implemented.

NOTES

Chapter 6

TIMER HANDLING

6.1 INTRODUCTION

6.1.1 Overview

CP's task of simulating all the timing facilities of a System/370 is not an easy one. Since CP has its own requirements for these facilities, some amount of complexity arises from the necessity of sharing the timers between the requirements of the simulation and CP's own requirements (e.g. for scheduling and accounting).

This chapter discusses the manner in which CP's requirements and the virtual machine's simulation requirements are handled for:

1. Location 80 interval timer.
2. TOD clock.
3. TOD clock comparator.
4. CPU timer.

The major control blocks used in CP timing facilities are discussed, as are the techniques for using those facilities. The rules for maintaining consistency between the various flags, registers, and timer contents are also covered in some detail.

6.1.2 References

6.1.2.1 Publications

1. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).
2. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

3. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

6.1.2.2 CP modules

The following CP modules are of major interest in understanding timer maintenance:

1. DMKBLD - initializes the virtual machine control blocks associated with the virtual timer simulation.

2. DMKCPI - initializes the TOD clock, stores the current date and TOD clock value at midnight to simplify future calculations, and schedules a timer request for midnight to update the stored date.

3. DMKDSP - has primary responsibility for simulation of the location 80 interval timer and is responsible for dispatching expired timer requests.

4. DMKPSA - contains the external first level interrupt handler.

5. DMKSCH - handles simulation of timers that must run when the virtual machine is in voluntary wait state. Contains the routines to maintain the timer request queue.

6. DMKTMR - handles simulation of privileged instructions that reference the timers.

6.2 CP TIMER MAINTENANCE

6.2.1 Location 80 timer

CP uses the location 80 timer for two purposes: to maintain the virtual machines' location 80 timers if they have SET TIMER ON or SET TIMER REAL, and to prevent a virtual machine that is relatively high in the run list from locking out other virtual machines' access to the CPU resource. There are three fields in the PSA that are used for location 80 timer maintenance:

1. QUANTUM at location 84 (X'54') contains the interval timer value when the virtual machine was dispatched. (Note that this field is only indirectly related to the dispatcher's "quantum" or time slice value.)

2. TIMER at location 80 (X'50') is the actual interval timer.

3. QUANTUMR at location 76 (X'4C') is the value of the interval timer when an interrupt caused control to return to CP.

The dispatcher sets a new value in TIMER whenever a virtual machine other than the previous RUNUSER is about to be dispatched. The new value comes from the field DMKDSPQS unless the virtual machine is marked as compute bound (VMCOMP in VMQLEVEL is on), in which case the new value is 4 times the contents of DMKDSPQS. In setting up to actually dispatch a virtual machine, the dispatcher moves the current value of TIMER to QUANTUM. All of the first-level interrupt handlers move the current value of TIMER to QUANTUMR, thereby preserving the interval timer value when CP was entered due to an interrupt. Upon entry to the dispatcher, the difference between QUANTUM and QUANTUMR is used to update the virtual machine's location 80 timer. A virtual external interrupt is queued if the value of the virtual timer has gone negative. If the real location 80 timer is negative, the virtual machine is flagged as having gotten a time slice end (VMTSEND in VMDSTAT set on) for later action by the scheduler.

The first level interrupt handler for external interrupts "handles" location 80 interval timer interrupts. It moves the current value of TIMER into QUANTUMR and exits to the main entry point of the dispatcher when various tests indicate no other possible cause for the external interrupt. Note that the ECPS microcode support for the location 80 timer attempts to present the interrupt directly to the virtual machine; if the interrupt cannot be presented virtually, then an external interrupt with a half-word code of X'01xx' (where the "xx" bytes are not checked) is presented to the real machine. When the external interrupt handler detects such an interrupt, it exits to a dispatcher entry point that queues a location 80 timer interrupt for the current RUNUSER virtual machine.

The initial value of DMKDSPQS is determined during CP initialization. Execution time of a certain part of the initialization code is compared with a "standard" value and then DMKDSPQS is adjusted to account for the speed of the real CPU.

interrupt code is stored in main memory to indicate the reason for the interrupt. Each interrupt routine must save the general purpose registers into a reserved area in the first 4K of memory and is then free to examine the interrupt code and take appropriate action. When the interrupted program is to be resumed, its registers must be reloaded and the old PSW must be restored via an LPSW instruction. The interrupted program will then resume execution immediately.

1.3.3 Dynamic address translation

Dynamic address translation (DAT) is also referred to as "virtual addressing" and is the technique by which a program is allowed to use addresses that do not necessarily correspond to the real addresses used by the hardware. DAT has four major functions in CP:

1. A user can be given apparent memory addresses, such as the first 4K, which have special meaning in the real hardware and which must therefore actually be used by CP itself.

2. A user's memory may be fragmented into a great many pieces in the real memory, and yet these pieces will appear contiguous to the user.

3. A user may have addressable memory that is not entirely located in the real memory. Thus, many users can all be allowed to have only some of their apparent memory available at any given time, thus reducing the amount of real memory that would otherwise be necessary.

4. Each user's virtual memory is completely isolated from all other virtual memories as well as from CP's (real) memory. This allows total memory protection without any conflicts in the use of the protect keys.

1.3.3.1 Registers

In order to support DAT, several registers have been added to System/370.

1. An EC mode PSW allows DAT to be turned on or off, according to the setting of bit 5. CP itself runs with bit 5 turned off, so that no address translation takes place. User virtual machines are run with bit 5 on, and are therefore subject to translation.

6.2.2 TOD clock

6.2.2.1 Initialization

A significant portion of code in the CP initialization module, DMKCPI, is devoted to initializing the TOD clock. The current date and day-of-week index are stored in various places (PSA, DMKDMP for dumps, DMKCQY to record IPL time, DMKSYS), and the value of the TOD clock at midnight is calculated and saved. This code in DMKCPI is the only place in CP where a complete conversion from TOD clock value to Gregorian date and time is required because it is the only place that issues a Set Clock instruction within CP (except for AP/MP systems that have to synch their TOD clocks); all subsequent calculations can use the stored values of the current date and the TOD clock value at midnight to simplify the calculation of the current time. A TRQBLOK is scheduled to expire at midnight, at which time DMKMID is invoked to change the date and day-of-week index in all the relevant places and write a message to all logged on users. The only other item worth noting about initializing the TOD clock is that if the operator enters a year with a value of less than 50, the TOD clock value is set with the assumption that it is now after the year 1999.

6.2.2.2 Use of the TOD clock within CP

The current TOD clock contents are often stored in a local data area. When queuing a TRQBLOK, TRQBTOD (see Figure 22) provides a handy doubleword aligned place to put the value temporarily when calculating the TOD clock value at which the TRQBLOK should expire.

One important aspect of using the TOD clock is that the condition code returned by the instruction is always checked, and if it is non-zero (indicating that the value stored may not be valid), the module almost invariably performs a GOTO DMKCVTAB to promptly terminate the system with a CVT001 ABEND code. The only time the system is not terminated by branching to DMKCVTAB is in sections of code that may be called during CP initialization before the TOD clock is initialized.

The Set Clock (SCK) instruction always results in a return code of zero when executed by a virtual machine. However, no action is taken other than setting the return code.

6.2.3 Clock comparator

CP uses the clock comparator for all internal functions that require the use of timing services. In addition, the clock comparator is used to simulate a clock comparator for 370 architecture virtual machines and to simulate the proper action of the virtual CPU timer and location 80 interval timer for a virtual machine that enters a voluntary wait state (loads a wait PSW).

6.2.3.1 TRQBLOK maintenance and queues

The CP control block used to govern all clock comparator requests is the TRQBLOK mapped by the TRQBLOK DSECT in the TIMER COPY file (see Figure 22).

```
TRQBVAL  DS    D    TOD clock value for expiration
TRQBFPNT DS    F    Forward pointer in TRQBLOK chain
TRQBBPNT DS    F    Backward pointer in TRQBLOK chain
TRQBTOD  DS    D    TOD when TRQBLOK was queued
TRQBUSER DS    F    A(VMBLOK) for associated VM
TRQBIRA  DS    F    Address to receive control
          .
          .
          .
```

Figure 22: Required fields in a TRQBLOK

TRQBLOKs are similar enough to IOBLOKs that some parts of CP treat them almost equivalently (e.g. the dispatcher). Since the storage occupied by a TRQBLOK is never released by any of CP's timing services, a TRQBLOK can reside either in permanently allocated storage or can be acquired dynamically by calling DMKFREE. In the latter case, it is the responsibility of the caller to insure that the storage is later released.

The clock comparator of the real machine is loaded with the time of expiration (TRQBVAL) of the first TRQBLOK on the request queue, anchored at the symbol DMKSCHTQ. The request queue is ordered by TRQBVAL and is maintained by two routines within the scheduler module, DMKSCHST and DMKSCHRT.

6.2.3.2 Scheduling a timer interrupt request

A TRQBLOK must be at least 4 doublewords long. The convention usually followed within CP is that TRQBLOKs, whenever possible, are based from R10. The fields that must be filled in are:

1. TRQBVAL - a doubleword containing the TOD clock value when this TRQBLOK is to cause an interrupt to be generated. Normal procedure is to STCK into TRQBVAL and perform a doubleword add of the appropriate constant. Note that the constant may be easily defined (letting the assembler do the hard work) by using a DC FL8S12E6'seconds' or a DC FL8S12E3'milliseconds'.

2. TRQBUSER - a fullword containing the address of the VMBLOK to be in register 11 when the TRQBLOK is unstacked and the interrupt routine receives control. Normally, this is the same as the current R11 value so a ST R11,TRQBUSER is sufficient.

3. TRQBIRA - a fullword containing the address of the routine to receive control when the TRQBLOK is unstacked. The routine receives control with this address in R12. Please note that this routine receives control from the dispatcher via direct branch rather than a CALL; therefore, the routine must be in a CP resident module or in a module that has been locked into memory for the time that the TRQBLOK is outstanding. A technique used in some parts of CP is to add a small appendage to an existing resident module that then issues a CALL macro to load the appropriate pageable module.

The TRQBLOK is placed on the timer request queue by loading the address of the TRQBLOK into R1 and calling DMKSCHST. Note that DMKSCHST uses the low memory save area BALRSAVE, so be sure that the module from which the call is made does not also use that save area.

6.2.3.3 Resetting an outstanding request

A TRQBLOK is often scheduled as a time limit for some event to occur. If the event occurs prior to the expiration of the time interval, the TRQBLOK may be removed from the timer request queue by loading R1 with the address of the TRQBLOK (it must have been saved away somewhere to do this) and calling DMKSCHRT. Again, note that DMKSCHRT uses BALRSAVE so some care must be exercised. If the TRQBLOK is to be returned to CP free storage, be sure to clear the pointer to it (it is usually a good idea to do this before calling DMKFRET).

If the TRQBLOK is not immediately released or resides in permanently allocated storage, it is a good practice to clear TRQBFPNT to zero. Following this practice simplifies later tests that need to know whether or not the time interval request is active. The main reason this practice is necessary is that DMKSCHRT is constructed to remove the TRQBLOK from whatever queue it might be on, either the timer request queue, or the dispatcher queue of unstacked TRQBLOKs; as a result, it performs NO checks to verify that the TRQBLOK is on any queue or that the addresses in TRQBFPNT or TRQBBPNT are valid.

6.2.3.4 Return of control when timer event occurs

When the clock comparator interrupt associated with a TRQBLOK occurs, the block is placed on the dispatcher's queue of IOBLOKs and TRQBLOKs (via CALL to DMKSTKIO) in FIFO order. When the TRQBLOK is unstacked, the routine specified in TRQBIRA receives control with its entry point in R12, the address of the TRQBLOK in R10, the address of the associated VMBLOK from TRQBUSER in R11, and the CPU timer containing the CP overhead time from the VMTTIME field of the associated VMBLOK. At this point, the routine receiving control should establish a good base register (usually the beginning of the module is loaded into R12) and clean up any TRQBLOK pointers as described above.

6.2.4 CPU timer

The CPU timer serves a dual purpose in CP; it is used both in the simulation of a virtual CPU timer and in controlling the CPU time used by a virtual machine during one stay in the run list (the amount of time is called a queue slice). Maintenance of the CPU timer while a virtual machine is dispatched is covered later; this section primarily discusses the maintenance of the CPU timer while within CP.

6.2.4.1 Maintaining the proper timer value

The rule for maintaining the proper CPU timer value is simple: when running CP code the current value for VMTTIME from the VMBLOK pointed to by R11 has been loaded into the CPU timer, and when running in problem state the current value for VMTMOUTQ from the VMBLOK of the RUNUSER has been loaded into the CPU timer. Since the VMTTIME field is used to account for CPU time in CP on behalf of a virtual machine, it is important for R11 and the CPU timer to be con-

sistent and to charge the CPU time to the virtual machine being serviced.

If it is absolutely necessary to have a section of code where R11 and the CPU timer are not consistent, that code must not contain calls to other modules that have the possibility of losing control of the CPU or storing the current value of the CPU timer into the wrong VMBLOK. Common errors in this regard are calling a module via SVC 8 or calling DMKFREE to request free storage; SVC 8 requires that the system obtain a save area, and any request for CP free storage runs the possibility of losing control while the system extends its free storage into the dynamic paging area.

6.2.4.2 Macros for maintaining the CPU timer properly

With the addition of the AP/MP support to CP, several macros were added to aid programmers in maintaining consistency between the VMBLOK pointer (R11) and the CPU timer. This discussion covers only the uniprocessor versions of these macros; use of the macros in AP/MP systems is discussed in the chapter on multiprocessor support.

The CHARGE macro is used in most circumstances when dealing with the CPU timer.

1. CHARGE START - causes the current value of VMTTIME (from the VMBLOK pointed to by R11) to be loaded into the CPU timer.

2. CHARGE STOP - causes the current value in the CPU timer to be stored in VMTTIME (from the VMBLOK pointed to by R11). Note that timing is not actually stopped by this macro, so it can be used in cases where a current value for VMTTIME is required but charging of the CPU time to the virtual machine continues.

3. CHARGE SWITCH,operand - causes the following actions to be taken:

    1. The current contents of the CPU timer are stored in the VMTTIME field of the VMBLOK pointed to by R11.

    2. R11 is loaded with the contents of the location defined by "operand" (assumed to be the address of a VMBLOK).

    3. The CPU timer is loaded with the VMTTIME field from the new VMBLOK pointed to by R11.

The other macro of interest is SWTCHVM. For uniprocessor systems, SWTCHVM is equivalent to CHARGE SWITCH,(R1); however, this equivalence is NOT true on AP/MP systems. Refer to the chapter on AP/MP support for an explanation of the impact of using SWTCHVM.

6.2.5 Maintenance of wait time statistics

When the dispatcher cannot find any work, it attempts to determine if a resource shortage is causing the wait condition. The current virtual machines in the run list are scanned to add up the total working set of virtual machines in page wait and the number of virtual machines in I/O wait. The wait state is considered to be due to paging activity if the sum of the working sets of virtual machines in page wait is more than half of the pages available (pageable + reserved - shared). Otherwise, if there is at least one machine in I/O wait, the wait state is considered to be due to I/O activity; if both tests fail, the real machine is considered to be in an idle wait state. Based on the above classification, an appropriate flag is set in CPSTAT3 indicating the kind of wait (CPTPAGE, CPTIONT, CPTIDLE) and the appropriate timer value is loaded into the CPU timer (PAGEWAIT, IONTWAIT, IDLEWAIT). The CPWAIT flag is set in CPSTATUS to indicate that the processor is entering wait state, and an enabled wait PSW is loaded.

When an interrupt causes an exit from wait state, the first level interrupt handlers store the current value of the CPU timer into the field WAITEND. Upon entry to the dispatcher, since CPWAIT is set indicating that wait time accounting should be done, the flags in CPSTAT3 are checked to determine which timer value was being used and WAITEND is moved to PAGEWAIT, IONTWAIT, or IDLEWAIT as appropriate.

6.2.6 Maintenance of problem state statistics

Whenever a new virtual machine is dispatched, the value from VMTMOUTQ is placed in the PSA field PROBSTRT. Subsequent entries to the dispatcher then result in the PROBTIME field being updated to include the difference between the current VMTMOUTQ for the RUNUSER and PROBSTRT; of course, PROBSTRT is then updated to contain the new value of VMTMOUTQ for that user. Any code within CP that changes the value of VMTMOUTQ for the current RUNUSER must also perform the same maintenance on PROBTIME and PROBSTRT as the dispatcher would have done. In particular, DMKTMR and DMKSCH have routines that perform this function if CP has been running a virtual machine (CPRUN in CPSTATUS is on) and the RUNUSER field is equal to the VMBLOK pointer in R11.

6.3 VIRTUAL MACHINE TIMER MAINTENANCE

A virtual machine can have either 360 or 370 architecture (controlled by the SET ECMODE OFF/ON command or the ECMODE option in the CP directory). In 360 architecture (VMV370R in VMPSTAT is off), the only timing facilities available are the location 80 interval timer and the TOD clock. In 370 architecture (VMV370R in VMPSTAT is on), the virtual machine can use simulated versions of the CPU timer and clock comparator in addition to the facilities available to 360 architecture virtual machines.

6.3.1 Location 80 interval timer

Under control of the CP SET TIMER command or CP directory options, the location 80 interval timer maintenance is in one of the following modes:

1. OFF - no location 80 timer maintenance is performed and no external interrupts are presented. (In VMTLEVEL, VMTON and VMRON are off.)

2. ON - virtual CPU time used by the machine is decremented from the virtual timer, but CP time and wait time are not. (In VMTLEVEL, VMTON is on and VMRON is off.)

3. REAL - virtual CPU time and voluntary wait time (virtual wait PSW loaded) are decremented from the value in the timer, but involuntary wait and CP time are not. (In VMTLEVEL, VMTON is off and VMRON is on.)

6.3.1.1 SET TIMER ON option

The change in the real location 80 timer is used to decrement the virtual location 80 timer for virtual CPU time, as has already been described in the section about real location 80 timer maintenance. The additional comment that should be made is that when the virtual machine's page 0 is not resident in real memory, the current value of the location 80 interval timer is maintained in the field VMTIMER within the VMBLOK. Special checks within CP's paging system recognize when a virtual machine's page 0 is moved in or out of real memory and move the current timer value between virtual location 80 and VMTIMER as appropriate.

6.3.1.2 SET TIMER REAL option

When a real machine enters a wait state, the location 80 interval timer continues to run. Similarly, if a virtual machine is using the REAL option for the location 80 interval timer enters a voluntary wait state, the timer continues to run. CP code in the scheduler accomplishes this by queuing a TRQBLOK that is set to expire when the interval in the virtual location 80 timer would expire; the field TRQBTOD is timestamped with the TOD clock value when the request is queued. When the virtual machine comes out of voluntary wait, the TRQBLOK is reset (via call to DMKSCHRT) and the difference between the current TOD clock value and TRQBTOD is used to update the contents of the virtual location 80 timer. Updating a REAL interval timer for virtual CPU time is exactly the same as described above for maintaining a timer that is ON.

If by chance the time interval for the TRQBLOK should actually expire, then the dispatcher unstacks the block to pass control to a routine in the scheduler module (DMKSCH80) that queues an external interrupt for the virtual machine and exits back to the dispatcher.

6.3.2 Clock comparator

6.3.2.1 Initialization

For a 370 architecture virtual machine (VMV370R is VMPSTAT is set on), the VMBLOK extension (ECBLOK) contains a pointer to a TRQBLOK used to simulate the virtual clock comparator. The TRQBVAL field of this TRQBLOK contains the value of the virtual clock comparator and its initial value is zero. The CKCMASK flag in byte 2 of virtual control register 0 is initially set to zero, indicating the virtual machine is not enabled for clock comparator interrupts.

6.3.2.2 Simulation of clock comparator instructions

The Store Clock Comparator (STCKC) instruction is simulated by placing the contents of the TRQBVAL field in the instruction's operand location after making appropriate checks for storage protection, addressing, and alignment.

The Set Clock Comparator (SCKC) instruction is simulated by the following:

1. Check for storage protection, addressing, and alignment. If problems are encountered, call DMKPRG to reflect a program interrupt.

2. Call DMKSCHRT to remove the current request from a queue, if necessary (determined by whether or not TRQBFPNT is zero).

3. Place the new value for the virtual clock comparator into TRQBVAL and store the current TOD into TRQBTOD. If the request has already expired, place the TRQBLOK on the dispatcher's queue by calling DMKSTKIO; otherwise, place the TRQBLOK on the timer request queue by calling DMKSCHST.

4. Check the queue of external interrupts pending for the virtual machine. If an interrupt for the clock comparator is found, remove it from the queue and release the storage for the XINTBLOK.

When the TRQBLOK is unstacked by the dispatcher, indicating that the interval has expired, the following actions are taken:

1. Clear the TRQBFPNT to indicate that the TRQBLOK is no longer on a queue and disallow fast re-dispatch for the virtual machine.

2. Obtain storage for an XINTBLOK and enqueue it on the pending external interrupt queue for the virtual machine.

6.3.3 CPU timer

6.3.3.1 Initialization

For a 370 architecture virtual machine, the ECBLOK also contains a pointer to a TRQBLOK used to simulate the virtual CPU timer. Unlike the virtual clock comparator, the value for the virtual CPU timer is placed in the EXTCPTMR field of the ECBLOK rather than being strictly maintained in the TRQBVAL field of the TRQBLOK. The initial value is zero.

6.3.3.2 Interaction with CP's use of the CPU timer

As noted before, CP normally maintains the remaining queue slice time in the real CPU timer while a virtual machine is actually executing. When an interrupt occurs and control is transferred to CP, the first level interrupt handlers store the current contents of the CPU timer in the VMTMOUTQ field of the VMBLOK and reload the CPU timer with the VMTTIME field of the VMBLOK of the virtual machine causing the interruption.

A special case that must be handled occurs when the time remaining in the virtual CPU timer is less than the time remaining in the queue slice. To handle this situation, a significant amount of complexity is introduced into the timer maintenance. First of all, the field TRQBQUE is synchronized with EXTCPTMR such that the time remaining in the queue slice in TRQBQUE corresponds to the time remaining in the virtual CPU timer stored in EXTCPTMR. When the value in EXTCPTMR is less than TRQBQUE, the real CPU timer is loaded with the virtual CPU timer value and a flag is set (VMCPUTMR in VMTLEVEL). In addition to the code in DMKTMR to maintain consistency for these fields, there are several routines in the scheduler to maintain these timer values and also to update VMVTIME. The VMVTIME field of the VMBLOK contains the actual virtual time only when the virtual machine is not in the run list; when the virtual machine is in the run list, the amount of time used in the current queue slice must be added to VMVTIME to give an accurate value for the total virtual CPU time.

6.3.3.3 CPU timer simulation during virtual wait

If the virtual machine enters a voluntary wait state, the virtual CPU timer must still run. Its current value is added to the TOD clock value (placed in TRQBTOD) and the TRQBLOK is added to the timer request queue by calling DMKSCHST. When the virtual machine exits from voluntary wait, the TRQBLOK is removed from the queue by calling DMKSCHRT; the difference between the current TOD and TRQBTOD gives the amount to subtract from EXTCPTMR to yield a current value for the virtual CPU timer. Whenever the TRQBLOK is queued because the virtual machine is in voluntary wait, the timer is said to be "stamped" and the VMSTMPT flag in VMTLEVEL indicates this condition. If the TRQBLOK expires, then control passes to a routine in the scheduler module (DMKSCHCP) that queues the external interrupt for the virtual machine and exits back to the dispatcher.

6.3.3.4 Simulation of CPU timer instructions

For a Set CPU Timer (SPT) instruction, the following actions must be taken:

1. Check for storage protection, addressing, and alignment. If there are problems, call DMKPRG to stack a program interrupt.

2. Update TRQBQUE by calculating the amount of CPU time used recently; this may be either the difference be-

2. CR0 contains bits that specify the format of the virtual memory configuration. The page size may be either 2K or 4K, and the segment size may be either 64K or 1M. CP selects 4K pages and 64K segments.

3. CR1 always contains the real memory address of the segment table; this register is often referred to as the "segment table origin" (STO) register. Each virtual machine has its own segment table, and therefore CP must place the correct table address into CR1 before letting a virtual machine execute. Figure 3 shows the register fields used by DAT.

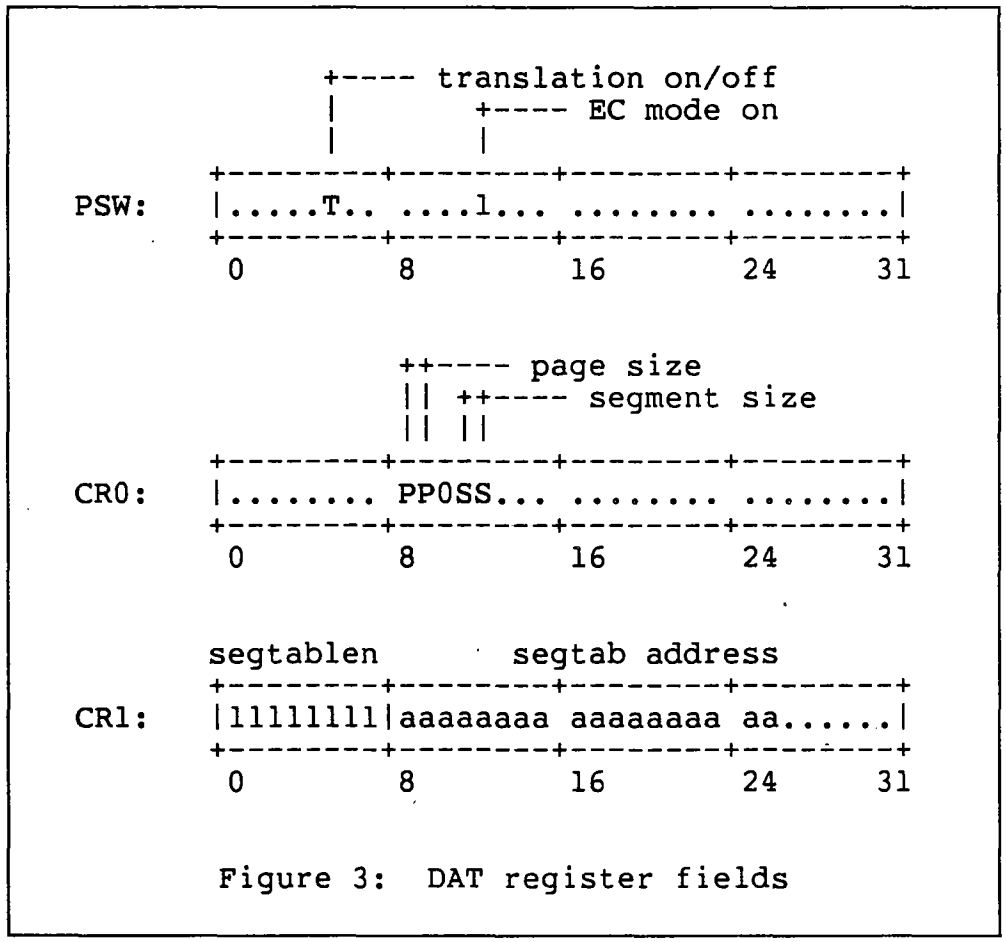


Figure 3: DAT register fields

1.3.3.2 Tables

The tables used by DAT are the segment table and the page table; these tables reside in real memory. For each virtual machine, CP maintains one segment table and enough page ta-

tween TRQBQUE and VMTMOUTQ or the difference between EXTCPTMR and VMTMOUTQ, depending on the setting of VMCPUTMR in VMTLEVEL.

3. Check for a possible CPU timer interrupt that may be pending; dequeue the XINTBLOK and release it if found.

4. Place the new value for the CPU timer in EXTCPTMR. If the value is negative, call DMKFREE to get a XINTBLOK, queue the external interrupt request, and exit to the dispatcher. If the new value is less than the time remaining in the queue slice (TRQBQUE), also place the virtual CPU timer value in VMTMOUTQ and set the VMCPUTMR flag indicating that the virtual CPU timer is "tracking in the real CPU timer". If the new value is greater than the time remaining in the queue slice, move the TRQBQUE value to VMTMOUTQ and clear the VMCPUTMR flag.

Execution of the Store CPU Timer (STPT) instruction proceeds in a similar fashion as the first part of the SPT instruction. However, once the queue slice time remaining has been updated, the value of the virtual CPU timer in EXTCPTMR is also updated and the result is placed in the operand location.

6.4 PSEUDO TIMER AND DIAGNOSE CODE X'0C'

A virtual machine can access certain date and timing information by using DIAGNOSE code X'0C' or by issuing a SIO to a special "timer" device (defined by the CP DEFINE command). Both methods return the date (mm/dd/yy) and current time (hh:mm:ss). In addition, the virtual time and the total time (virtual + overhead CPU) are returned; however DIAGNOSE code X'0C' returns these times in microseconds (8 bytes, rightmost bit represents one microsecond), and the pseudo timer returns the time values in location 80 interval timer units (4 bytes, rightmost bit is approximately 13 microseconds). The code that implements the two functions, although in different modules, is almost the same in that they both call DMKTMRPT to get the total problem time and DMKCVTDT to obtain the current date and time information. Of course, to convert to interval timer units, the returned values are divided by 13; note that the result is modulo the value that can be stored in the interval timer. The code to handle DIAGNOSE code X'0C' is located in DMKHVC and the pseudo timer code is in DMKVSP.

6.5 DIAGNOSE CODE X'70' FOR SCP TIMING SUPPORT

Certain operating systems use the TOD clock to perform accounting for CPU utilization among various tasks. DIAGNOSE code X'70' allows such operating systems a relatively undisruptive way of obtaining accurate timing information without requiring extensive changes. The DIAGNOSE is issued once after the OS is IPL'ed and points to a 16 byte area in which CP subsequently maintains the TOD clock value and the total CPU time each time the virtual machine is dispatched.

The algorithm for using the timers to do task timing is:

Save the total CPU time when a task is dispatched. This value may be placed in the same field that previously held the TOD at which the task was dispatched. Note that a privileged operation, like STIDP, is necessary to get the current total CPU time field updated in the DIAGNOSE code X'70' area. When it is desired to calculate the task's CPU usage:

1. Execute a STCK to get the current TOD.

2. Subtract from this value the TOD at the last dispatch; the result is the amount of CPU time used since the last dispatch.

3. Add the result of the previous step to the total CPU time at the last dispatch.

4. Subtract from the result of the previous step the CPU time that was saved when the task was first dispatched. The result is the CPU time used by the current task.

5. Check that none of the DIAGNOSE code X'70' values changed during the above calculation; if they have, redo the calculation.

Note that while this DIAGNOSE code was added especially to improve the accuracy of MVS task timing, MVS does not use this DIAGNOSE code; MVS was modified instead to use the CPU timer for task timing. Also, use of this diagnose code means that certain ECPS functions are not performed; in particular, the DSP2 ECPS instruction that performs the function of setting up to dispatch a virtual machine is at least partially disabled.

6.6 SUMMARY

This chapter has covered the use of the real hardware timers by CP and the techniques used by CP to simulate these timers for virtual machines. A description of the "pseudo-timers" used to record the amount of time the CPU spends in wait state was included, as was a description of the handling of TRQBLOKs along with guidelines for using TRQBLOKs without getting into trouble.

NOTES

Chapter 7

INTER-VIRTUAL-MACHINE COMMUNICATIONS

7.1 INTRODUCTION

7.1.1 Overview

There are two distinct and quite different inter-virtual machine communication facilities supported by CP. The two facilities are significantly different in the capabilities provided; therefore, some understanding of how they work internally is very useful in determining the best match between application and communication facility.

Virtual Machine Communication Facility (VMCF) was introduced in VM/370 Release 3 PLC 8 and provided a much-needed high performance way of transferring data between virtual machines without having to resort to shared minidisks or spool files. Although the function provided by VMCF was sorely needed, IBM and customer installations used VMCF only for some specialized applications (for example, Interactive File Sharing and the Real Time Monitor). The limitations of the VMCF protocols and support made it unsuitable or undesirable for the more complex application requirements that soon developed.

In Release 6 of VM/370, IBM released the Inter-User Communication Vehicle (IUCV) to provide a more powerful inter-virtual machine communications facility and to support the requirements of their VS/1-based SNA support (VCNA). IUCV provides functions that more closely resemble those one would expect to find in a modern communications network; it serves as a base for communicating both among virtual machines *and* with CP-provided services.

The two communications protocols have a number of functions that must be performed in a logically similar fashion. Table 9 indicates the names of the functions that are common between the VMCF and IUCV protocols. Since IUCV communications includes the construction of a logical communications path, there are additional IUCV functions to handle these paths (CONNECT, ACCEPT, and SEVER). In addition, IUCV communications may be programmed in a manner that eliminates most external interrupts during normal message passing. The IUCV functions of SET MASK, TEST COMPLETION, and DESCRIBE are used in this "almost no interrupts" mode.

TABLE 9

Functions common between IUCV and VMCF

VMCF	*IUCV*
AUTHORIZE	DECLARE BUFFER
UNAUTHORIZE	RETRIEVE BUFFER
SEND	SEND (1-WAY)
RECEIVE	RECEIVE
CANCEL	PURGE
REJECT	REJECT
REPLY	REPLY
SEND/RECEIVE	SEND
QUIESCE	QUIESCE
RESUME	RESUME

7.1.2 References

7.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

3. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

4. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

7.1.2.2 CP modules

1. DMKIUA - handles initial processing of IUCV request and contains several high-use routines for things such as external interrupt reflection and enqueuing message blocks.

2. DMKIUC - handles low-use functions such as QUIESCE and RESUME for IUCV communications.

3. DMKIUE - handles high-use functions such as SEND and RECEIVE for IUCV communications.

4. DMKIUG - handles low-use functions such as PURGE and REJECT for IUCV communications.

5. DMKVMC - handles all VMCF communication functions.

7.2 VMCF CONTROL BLOCKS

The VMCF control blocks are relatively simple to understand and maintain. While a virtual machine is performing VMCF communications, the VMBLOK contains a pointer (VMCPNT) to a master VMCBLOK. The master VMCBLOK contains information such as:

1. Whether or not special messages (SMSG) are being accepted.

2. Whether or not priority messages are being accepted.

3. The userid of a specific virtual machine that is authorized to communicate with this virtual machine, if communications are so limited.

4. The count of the number of outstanding messages sent from this virtual machine.

5. The count of messages sent to virtual machines with the QDROP OFF USERS option.

6. A pointer to a chain of VMCBLOKs representing pending interrupts or messages for this virtual machine.

7. The TOD clock value when VMCF communication was initialized.

8. The address and length of the external interrupt buffer.

The messages sent between virtual machines are specified by a VMCF parameter list mapped by the VMCPARM DSECT. Within VMCF, the parameter list is copied to CP free storage and converted to a VMCBLOK.

The VMCBLOK contains fields that describe the specific message and indicate its current status.

1. The address and length of the SEND buffer in the source virtual machine. The reply buffer is also specified for SEND/RECV functions.

2. A flag indicates whether or not a VMCF interrupt is pending from this VMCBLOK.

3. A flag indicates whether or not the communications process is complete (following the reflection of the last interrupt) and that the VMCBLOK can then be released.

4. A flag indicates whether or not this message was sent to a virtual machine with the QDROP OFF USERS option.

7.3 IUCV CONTROL BLOCKS

The IUCV control blocks are a bit more complicated than VMCF due to the increased capabilities and performance requirements. A virtual machine that has initialized for IUCV communications has a field in its VMBLOK (VMIUCV) that points to a IUCVBLOK. The IUCVBLOK contains the standard sort of control information that one might expect based on the VMCF control blocks. However, in the IUCV world, the information is divided between the IUCVBLOK and another block, the Communications Control Table (CCT). The IUCVBLOK contains flags indicating the kind of interrupts enabled and pointers to control blocks called MSGBLOKs representing external interrupts that are pending for control-type functions (QUIESCE, RESUME, CONNECT, SEVER). The CCT contains the queue pointers for:

1. messages that are pending and not yet received (the SEND queue),

2. messages that have been RECEIVEd but no reply has been sent (the RECEIVE queue), and

3. messages that have been completed and that are waiting to generate an external interrupt or for a TEST COMPLETION function to be executed.

Like VMCF, the queues are singly-linked lists. Unlike VMCF, pointers are maintained to both the head and the tail of the list to shorten the path-length of adding an entry to the end of the list.

Because IUCV is a path-oriented communication protocol, there must be cross-links between the virtual machine control blocks at either end of any communications path. At the end of the CCT are pointers to Path Descriptor Segments (PDSEGs). Each PDSEG contains a number of entries linking paths and indicating the path status (connection pending, SENDs are not allowed, etc.). The PDENT performs the cross link by having a pointer to the CCT of the virtual machine

at the other end of the path and the path-id by which the other virtual machine knows the path. The path-id for a given path is local to each end of the communication link and is formed from the PDSEG index and the PDENT index to provide a unique and high performance method of determining paths.

7.4 HIGH LEVEL PROCESSING

The execution of a "communications instruction" in a virtual machine (X'B2F0' for IUCV and DIAGNOSE code X'68' for VMCF) results in a real program check interrupt that the first level interrupt handler examines and then passes control to the appropriate communications module. The communications module (DMKIUA for IUCV and DMKVMC for VMCF) has a high level routine that performs the following functions:

1. Verify that the user has issued all necessary initialization calls so that control blocks are initialized and the location and length of the external interrupt buffer is known.

2. Copy the communications parameter list into CP free storage. Since the parameter list may be split across page boundaries within the virtual machine, the code for checking all the nasty error conditions is centralized and processing of the parameter list is simplified.

3. Examine the parameter list for the communications function requested and invoke the appropriate processing routine.

4. When the prrocessing routine is complete, return control to the high level routine to pass the results back to the requesting virtual machine.

7.5 INITIALIZING FOR COMMUNICATIONS

The VMCF AUTHORIZE or the IUCV DECLARE BUFFER functions must be executed by a virtual machine before it is allowed to send or receive messages using the corresponding protocol. For the initialization call, the virtual machine must provide the address of an external interrupt buffer that will be used for storing the message header when a message arrives or when a message has been sent. The interrupt buffer must be within the virtual machine's memory and have a protect key such that "store access" is allowed based upon the PSW protect key when the initialization function is invoked.

bles to completely cover the virtual machine's defined memory size.

1. The segment table consists of a series of fullword values. Each word corresponds to a segment (64K bytes) and contains the real memory address of the page table for that segment. The entire segment table must be located in a single contiguous area of real memory. A control bit in each word can be set to show that the segment is "invalid"; that is, the segment table entry does not currently point to a valid page table. If the "invalid" bit is set, then the CPU will generate an interrupt when any addressing process tries to use that segment. Figure 4 shows the format of a segment table entry.

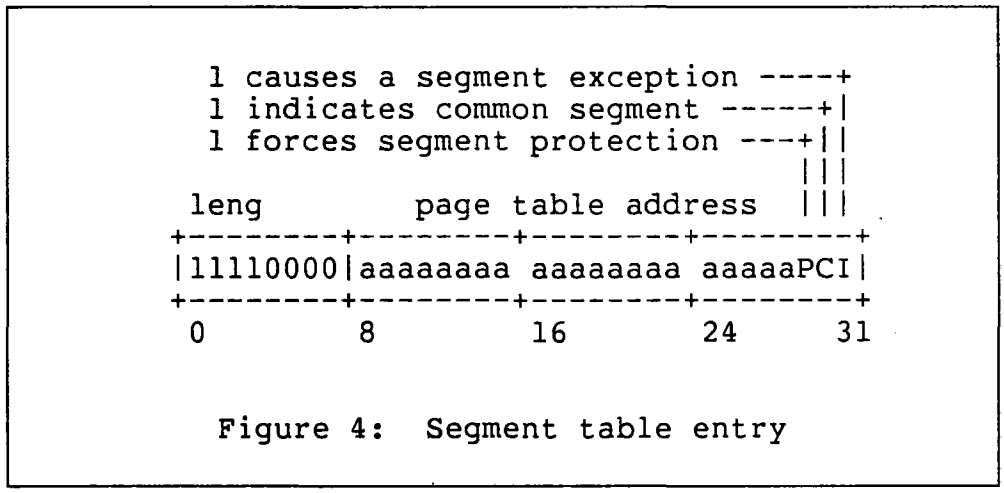


Figure 4: Segment table entry

2. The page table consists of a series of halfword items, one for each page (4K bytes) in a segment. The first 12 bits of the halfword contain the high-order 12 bits of the real memory location where the page is located. If the system has more than 16M of storage, then the otherwise unused bits 13 and 14 are used to contain the even higher order real address bits. An "invalid" bit allows the generation of an interrupt if the page table entry is used in any addressing process. CP sets that bit to indicate, among other things, that the virtual machine's page is no longer resident in real memory; the interrupt serves to notify CP that the virtual machine wants to reference the page and that CP must restore the page to real memory before the virtual machine can be allowed to continue processing. Figure 5 shows the format of a page table entry for a 4K page.

Once the validity checks in the interrupt buffer address are successfully completed, the control blocks are obtained and initialized. Any special options specified by the caller are set (e.g. willing to accept priority messages) and control is returned to the high level routine.

7.6 REFLECT EXTERNAL INTERRUPT

The following description generally applies to both VMCF and IUCV communications; IUCV also has the DESCRIBE and TEST COMPLETION functions that allow the application to get the same information as would be provided by the external interruption but without having to go through the interrupt simulation. Throughout the remainder of the chapter, the control block that describes the message is referred to as a Message Descriptor (MD). For VMCF and IUCV, the control block corresponding to the MD is the VMCBLOK and the MSGBLOK, respectively.

When the MD is queued for a virtual machine, a flag in the MD indicates the virtual machine does not know of the presence of that MD until informed by an external interrupt. When the virtual machine enables for external interrupts (with the appropriate additional bit set in CR0), the queue of MDs is searched for one with the flag indicating that an external interrupt should be presented. For VMCF communications, the virtual machine is not allowed to RECEIVE or otherwise process the message described by the MD while this flag is set.

Once an appropriate MD is found, descriptive information about the message is moved into the interrupt buffer (the parameter list for the synchronous IUCV functions) and the flag changed to indicate that an interrupt is no longer pending for this MD. Since moving the data to the interrupt buffer may involve bringing several pages of the virtual machine into memory, it is recommended that the interrupt buffer not cross a page boundary and, if possible, that the buffer be located in page 0 of the virtual machine (page 0 has to be resident to reflect the external interrupt in any case). Considering the amount of free storage available in page 0 of a CMS virtual machine, such a recommendation almost seems like an open invitation to use the IBM copyright area. However, such practices should be discouraged since there will most certainly be problems when some other programmer decides to use that prime piece of memory.

Once the message information has been copied, the storage occupied by the MD is released if the transaction is complete. This normally occurs when the message sender is given the final status, but certain functions like SMSG imple-

ment a 1-way transfer of information with no acknowledgement that the data was received.

7.7 COMMUNICATIONS

In this section, the normal communication functions are described: SEND, RECEIVE, and REPLY. At the conceptual level, the functions provided by VMCF and IUCV are very similar; however, the ability of IUCV to send messages of various classes along a communications path may provide a facility that simplifies code within an application.

Descriptions of VMCF and IUCV refer to the message sender as the "source"; the receiver of a message is called the "target" by IUCV and the "sink" by VMCF. In the following discussions, we adopt the IUCV nomenclature of "source" and "target" for the sending and receiving virtual machines, respectively.

7.7.1 SEND

The purpose of the SEND function is to queue the MD for another virtual machine. The MD specifies the source virtual machine and the virtual address and length of the message. For 2-way communications, the MD also contains the virtual address and length of the reply buffer within the source virtual machine (for VMCF, 2-way communication is indicated by the SEND/RECEIVE function).

The steps performed by the SEND function are as follows:

1. Validate the parameters supplied by the source virtual machine. Besides checking that the caller is not violating memory protection or addressing in the specification of the message (and reply) buffers, the message identifier is checked to insure that there are no outstanding messages between the source and target with the same identifier.

2. Allocate storage for the MD and initialize based upon fields in the parameter list. Note that for VMCF, additional storage is not obtained; the parameter list is simply reused as a message descriptor by shuffling some of the fields around.

3. Add the MD to the queue MDs awaiting processing by the target virtual machine.

4. Ensure that an external interrupt of the appropriate type is queued for the target virtual machine. Note that the flag is set in the MD to indicate that an external interrupt is pending.

Control is returned to the high level routines (VMCF just exits with an appropriate return code).

7.7.2 RECEIVE

Once the target virtual machine has been notified of the existence of the message by receiving the external interrupt (or by using the IUCV DESCRIBE function), the transfer of data is initiated by the RECEIVE function. The steps for performing the RECEIVE are as follows:

1. Validate the parameter list. The buffer to receive the data must be within the virtual machine memory.

2. Move the data from the source virtual machine's memory to the target virtual machine's memory. Due to the need to check for storage protect violations, data movement is performed in blocks of up to 2K at a time. Greatest efficiency can be obtained for moving large amounts of data if the source and target buffers start at the same offset within a 2K block of memory.

3. If the message was for a 1-way communication (VMCF SEND function or IUCV SEND with the 1-WAY flag set), enqueue the MD on the source virtual machine and insure that an external interrupt of the appropriate type is pending for the source virtual machine.

If the communications is 2-way, the MD is moved to the REPLY queue for IUCV; but for VMCF communications (SEND/RECV function), the MD is merely flagged that the receive function has been performed.

7.7.3 REPLY

The REPLY function specifies the data returned to the source virtual machine in response to a message. The following steps are performed by the REPLY function:

1. Validate the parameter list. The normal validation of virtual addresses and memory protection occur; also the MD for which the function is requested must indicate that the message has been RECEIVEd and is waiting for a REPLY function to be executed.

2. Move the data from the target virtual machine's memory to the source virtual machine's memory. The same efficiency considerations apply here as discussed for the RECEIVE function.

3. Enqueue the MD for the source virtual machine and insure that an external interrupt of the appropriate type is outstanding for the source virtual machine.

All data transfer operations for a message transaction are now complete. The only remaining task is to inform the source virtual machine by external interrupt (or the IUCV TEST COMPLETION function).

7.8 CONTROL

In addition to the normal message-passing functions discussed in the previous section, both VMCF and IUCV implement certain control functions that allow applications to more easily manage the data flow.

7.8.1 QUIESCE and RESUME

Under a number of conditions, the target virtual machine may desire to halt the flow of incoming messages:

1. The application running in the virtual machine is terminating; it is still possible to process messages that are queued but no new messages should be sent to this virtual machine.

2. The application is receiving messages at a faster rate than it can handle them and wants to process its current queue of work before any new messages are sent.

The QUIESCE function merely sets a flag in a control block indicating that no further SEND functions should be allowed on the specified path (IUCV) or to the virtual machine (VMCF). It is a good practice to QUIESCE communications before terminating an application so that messages already sent by a source can be handled and the procedures for error recovery in the source may be simplified.

The RESUME function resets the flag that was set by QUIESCE. In addition, for IUCV the source virtual machine is presented with a external interrupt for IUCV to indicate that the path has been RESUMEd. When using VMCF, IBM recommends that the IDENTIFY function be executed to notify the

source virtual machines that the target is once again accepting messages; in practice, this requires that the target remember the userids of all source machines and the application running the source virtual machines understand the meaning of the IDENTIFY (the source may not know that communications were quiesced, so that case too must be handled).

7.8.2 CANCEL (VMCF) and PURGE (IUCV)

When the source virtual machine wishes to "take back" a message it has sent to a target, the IUCV function PURGE or the VMCF function CANCEL should be executed. In general, the message may be in any one of a number of states:

1. The external interrupt has not been presented to the target so the target does not know that the message exists.

2. The target has been informed of the message but has not executed a RECEIVE function.

3. The RECEIVE function is currently being executed.

4. The RECEIVE function has been executed and is waiting for the target virtual machine to REPLY.

5. The target machine is in the process of transmitting the REPLY.

6. The message processing is complete and the interrupt notifying the source virtual machine is pending.

The first and last cases above are the easiest to handle in terms of not interfering with the applications program running in the virtual machine. In these cases the MD is dequeued and released; a return code indicates to the application the fact that in the latter case the message has not been CANCELled but has, in fact, been completed. Note that in the last case, the MD is queued on the source virtual machine's VMBLOK; this case is always checked for first. If data transfer is in progress (RECEIVE or REPLY being executed), a VMCF CANCEL is rejected with an indication that the MD is busy and a IUCV PURGE does not find the MD since it is not on any queue at the time. For the remaining cases when the target knows of the message but has not fully processed it yet, VMCF protocol calls for simply dequeuing the MD and releasing it while IUCV protocol calls for flagging the MD as PURGEd but not actually dequeuing and releasing the MD until after the target attempts to process the message (the target receives an indication that the message was PURGEd and is no longer available).

7.8.3 REJECT

The REJECT function allows a target to terminate processing for a message for whatever reason it desires. After validation of the caller's parameter list, processing for this function is as follows:

1. Search the invoker's queue(s) for the MD.

2. Flag the MD as REJECTed and return the MD to the source virtual machine.

3. Enqueue an external interrupt (if necessary) for the source virtual machine.

Of course, the information passed back to the source indicates that the message was REJECTed and the communications protocols allow for the REJECT function to supply a "reason code" to the source indicating the cause of the rejection.

7.9 TERMINATING COMMUNICATIONS

The VMCF UNAUTHORIZE and the IUCV RETRIEVE BUFFER functions allow an application to terminate inter-machine communications. Note that these functions are also performed whenever a virtual machine is reset (LOGOFF, IPL, SYSTEM RESET, DEFINE STORAGE, etc.). The basic processing is to:

1. PURGE/CANCEL all completed MDs queued for the invoker.

2. REJECT all pending messages.

3. Clear pending external interrupts associated with the communications protocol. Release all control blocks and clear any pointers to them in the VMBLOK.

Because IUCV is a path-oriented protocol, the clean-up when communications are terminated can be more complete than is the case with VMCF (see the discussion of the SEVER function later in this chapter). Note that VMCF SEND messages queued on another virtual machine do not get cleared by an UNAUTHORIZE function. The actual data transfer operation for the SEND could actually occur for a different session of the source virtual machine than the one actually invoking the SEND function, as long as the source had performed a VMCF AUTHORIZE before the target executed a RECEIVE or REPLY function.

7.10 IUCV PATH FUNCTIONS

Much of the preceding discussion has been independent of IUCV or VMCF. This section discusses the specific functions available due to the path-orientation of IUCV. The path-orientation of IUCV introduces at least one critical difference in comparison with VMCF; CP knows which virtual machine is on each end of the path and the communications functions can more easily keep each virtual machine aware of the status of the other machine in terms of the communications link. One area really showing this difference is QUIESCE/RESUME processing. The VMCF protocol for notifying communicating source virtual machines that a target virtual machine has issued a RESUME (by having the target issue an IDENTIFY to each of the source virtual machines) can at best be termed a kludge. IUCV handles the problem automatically by notifying the source virtual machine immediately when the target issues either a QUIESCE or a RESUME on the path.

7.10.1 CONNECT

A virtual machine issues the IUCV CONNECT function when it desires to establish a communications path to a process running in another (or the same) virtual machine. Permission to utilize IUCV communications must be granted by the installation; the CP directory entry for the virtual machine contains the limit on the number of active IUCV connections it can have. Once this permission has been checked, processing for the CONNECT function proceeds as follows:

1. A currently unused PDENT is found for both the source and the target virtual machines. Some number of these unused entries are set up at the time IUCV communications is initialized with the DECLARE BUFFER function; more PDENTs can be dynamically allocated (by acquiring additional PDSEGs) as necessary within the directory specified limits on the number of active connections.

2. Initialize the PDENT on each end of the path to contain the address of the CCT for the virtual machine at the other end, the pathid of the other end of the path (remember, this is just the index to the PDENT for the other end of the path), and a flag indicating that this path is pending connection.

3. Stack an external interrupt for the target machine so it knows that a CONNECT request is pending.

At this point, the source virtual machine must wait on a response from the target; the only function the source can

perform on the path is a SEVER, which may be desireable if the target does not respond within a reasonable period of time.

7.10.2 ACCEPT

A virtual machine requests the ACCEPT function to complete the establishment of a communications path that has been initiated by the issuing of a CONNECT request from another virtual machine. After parameter validation, the ACCEPT routine performs the following:

1. Reset the connection pending flags at both ends of the path.

2. Set a flag at both ends of the path indicating that the path is valid.

3. Stack an external interrupt to inform the source virtual machine that the path is now ready for communications.

Normal IUCV message-sending functions may now be executed for the communications path.

7.10.3 SEVER

The SEVER function may be used to terminate communications along a communications path or as a response to a CONNECT request. After a communications path is established, a SEVER function must be executed on both ends of the path for all the resources to be reclaimed.

If the pathid in the SEVER parameter list is marked as being valid, the following actions are performed:

1. QUIESCE the path at both ends. This prevents either end from sending additional messages.

2. Invoke PURGE for MDs on the invoker's REPLY queue that are associated with the path being severed. These MDs represent messages that have been "delivered" and are waiting for the virtual machine to enable for the appropriate external interrupts or to execute the TEST COMPLETION function.

3. Dequeue any external interrupts associated with the path being severed.

4. Invoke REJECT for any MDs on the invoker's SEND or RECEIVE queues that are associated with the path being severed. These MDs represent messages sent to the virtual machine that have not been RECEIVEd (the SEND queue) or REPLYed to (the RECEIVE queue).

5. Invoke PURGE for any MDs on the target's SEND or RECEIVE queues that are associated with the path being severed. These MDs are messages sent from the source virtual machine but not yet completely processed by the target virtual machine.

6. Enqueue an external interrupt for the target virtual machine to notify it that the path is being severed.

7. Set a flag in the PDENT of the target indicating that the path has been severed.

The communications path has now been partially broken. Complete termination of the path is held up until the target executes a SEVER, indicating that it has performed any necessary clean-up functions.

If the pathid indicated in the parameter list is invalid, then the SEVER function must be in response to a pending CONNECT or a pending SEVER. All outstanding MDs are now released and the PDENTs are returned to their initial status. An additional check is performed if the last active path within a dynamically acquired PDSEG has been deactivated, in which case the storage occupied by the PDSEG is released..

7.11 CP SERVICES VIA IUCV

Although the preceding discussion assumed the communicating processes resided within virtual machines, IUCV allows communications paths to be established between CP service functions and virtual machines or between CP services (although no intra-CP usages exist at the current time).

To communicate with a CP system service, the name of the service must be specified in the CONNECT parameter list. The names of all available CP services are contained in a table located at label CPNMTBL in module DMKIUC. Note that the names of all the CP services start with an '*' and are therefore easily distinguished from a virtual machine user-id.

Besides the CP name table, another table in DMKIUA at label CPENTTBL lists the entry points to gain control when the CP system service is the target of a particular function. Entry points are provided for the following functions for each defined system service:

1. CONNECT.

2. MESSAGE (SEND).

3. SEVER.

4. QUIESCE.

5. RESUME.

These entry points are sufficient to provide full IUCV communications between CP and an application running in a virtual machine. Currently-defined CP system services are:

1. *CCS - Console Communications Services, used by VCNA/VTAM. The entry points to perform the IUCV functions are all in module DMKVCT.

2. *MSG - Message System Service, used to reroute a number of virtual console messages back to an applications program. The entry points to perform the IUCV functions are all in module DMKMSG.

3. *BLOCKIO - DASD Block I/O System Service, providing virtual machines with asynchronous device independent access to virtual DASD devices. Support for this function is in module DMKBIO.

Using the modules mentioned above, it should be a relatively easy task to establish a communications path between CP and an application running in a virtual machine, something requiring significant and complex modifications in the past.

7.12 SUMMARY

This chapter has presented an overview of the internal operation of the VMCF and IUCV inter-machine communication protocols along some understanding of their differences, both positive and negative. Even though it has been around for some time, VMCF has many limitations that preclude its use for many applications. IUCV appears to be IBM's choice for adding new functions (such as the block I/O support in VM/SP Release 3) and therefore would be the choice for most new applications you plan to develop.

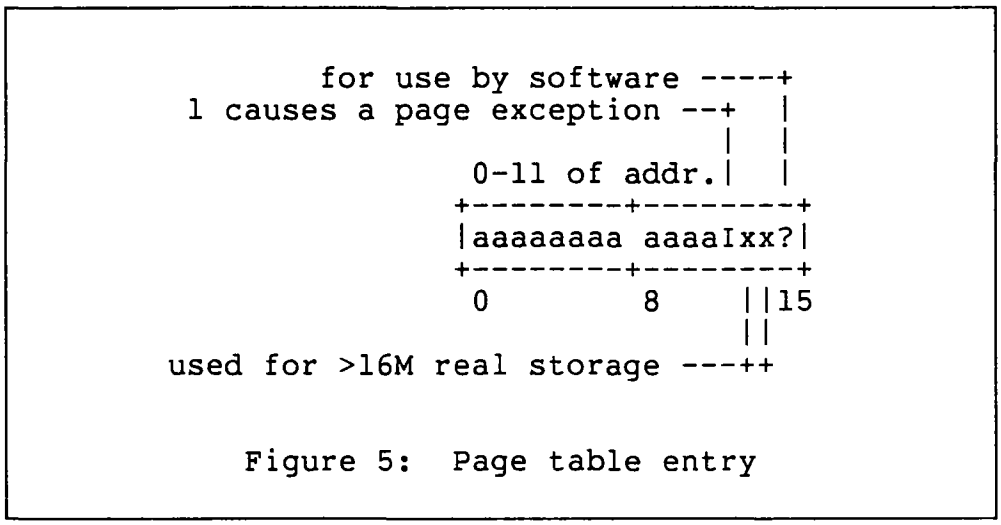


Figure 5: Page table entry

1.3.3.3 Translation

The address translation process proceeds as follows:

1. The low-order 12 bits of the address are saved, since they are merely the displacement into the given page.

2. The high-order 8 bits are used as an index into the segment table. If the "invalid" bit is on in the given segment table entry, or if the entry is beyond the defined range of the segment table, then an interrupt is generated.

3. The page table address is gotten from the segment table entry and the next 4 bits of the address are used as an index into the page table. If the "invalid" bit is on in the chosen page table entry, then an interrupt is generated. Otherwise, the 12 high-order bits from the page table entry are concatenated to the saved 12 low-order bits to form the real 24-bit memory address.

Naturally, this process takes time, even for the very fast electronic circuits that are employed. For that reason, many System/370 CPU models contain a "translation lookaside buffer" (TLB), which contains a number of virtual addresses and their corresponding real addresses. Whenever the previously described translation process is invoked, the results are saved in the TLB. Since most program addressing sequences tend to be sequential, there is a very good chance that another reference to the page will occur very soon, and the TLB entry can therefore be used to speed up the translation process. The TLB is designed so that it can be

NOTES

Chapter 8

STORAGE MANAGEMENT

8.1 INTRODUCTION

8.1.1 Overview

Recall from the CP architecture chapter that there are two kinds of real storage management in CP. There is user storage allocated in 4k pages and administered by the module DMKPTR. This storage is also called the dynamic paging area (DPA). There is another type of storage called free storage. Free storage is that portion of main memory set aside for CP control blocks. This storage is administered by the module DMKFRE. User storage administration is discussed first, followed by free storage administration.

8.1.2 References

8.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

2. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

3. R. Parmelee, et al., *Analysis of Algorithms for CP-67 Free Storage Management*, IBM Cambridge Scientific Center, July 1971.

4. E. Bozman, et al., "Analysis of Free-Storage Algorithms - Revisited," pp. 44, *IBM Systems Journal*, Vol. 23, No. 1, 1984.

8.1.2.2 CP modules

1. DMKFRE - manages CP working storage.

2. DMKPTR - manages the dynamic paging area.

8.2 REAL STORAGE CONTROL BLOCKS

The primary real storage control block is the CORTABLE. It is anchored in the PSA and has one element for every 4096 byte frame of real storage installed on the system. All of the CORTABLE entries are generated in one block of memory as a result of the expansion of the SYSCOR macro in DMKSYS; the first CORTABLE entry is labelled DMKSYSCS. Each entry is a fixed length, currently 16 bytes long. The information in a CORTABLE entry varies as the status of a page frame changes. Details of the information in the CORTABLE entry follow.

8.3 USER STORAGE MANAGEMENT - DYNAMIC PAGING AREA (DPA)

The part of CP's real storage not devoted to the PSA, the fixed nucleus, the trace table area, the free storage area, and the V=R area is called the dynamic paging area (DPA). This space is managed by the module DMKPTR. It is administered in page-sized blocks. The DPA is given to DMKPTR at CP initialization by DMKSTA. If the V=R area is unlocked by operator command, then it is logically added to the DPA. Practically speaking, DMKPTR administers the DPA by manipulating the CORTABLE.

In addition to the main entry point DMKPTRAN, there are several other entry points for specialized system functions. The more critical of these secondary entry points are DMKPTRFE and DMKPTRFT, used to get storage for and accept storage from DMKFRE. DMKFRE administers CP storage allocated in doublewords, primarily for control block purposes.

8.3.1 Introduction

In order to allocate storage effectively, DMKPTR maintains two lists of page frames. The free list is doubly-linked through the CORTABLE entries and is anchored at DMKPTRF1. This list represents all of the page frames that are available for immediate allocation. The flush list, sometimes called the user page list, is also doubly-linked through the CORTABLE entries and is anchored at DMKPTRU1. This list contains page frames that may be available for allocation, but it may be necessary to write the page out to DASD before the page frame can be made available for the free list. If a page frame is allocated to an active user, the CORTABLE entry for that frame contains pointers to the user's SWPTABLE, PAGTABLE, and VMBLOK. The flag byte in the CORTABLE keeps track of the status of the page frame. The definitions for the flag bits are given in table 10.

TABLE 10

CORFLAG flag definitions

Hex	Flag name	Definition
80	CORIOLCK	Frame locked; CORLCNT positive.
40	CORCFLCK	Frame locked for console function.
20	CORFLUSH	Frame on flush list.
10	CORFREE	Frame on free list.
08	CORSHARE	Frame contains a shared page.
04	CORRSV	Frame reserved.
02	CORCP	Frame owned by CP.
01	CORDISA	Frame disabled (not available).

Most calls to DMKPTR are made via the TRANS macro, which is the standard means by which CP routines obtain the real address of a specified page of virtual memory. The expansion of the TRANS macro contains a Load Real Address (LRA) instruction; if this instruction executes with a condition code of zero, then the needed page is already in main storage and the call to DMKPTR is avoided. This design eliminates much potential overhead in the memory management system. DMKPTR is called from within the TRANS macro only when the LRA instruction completes with a non-zero return code. The DMKPTR module can also be is invoked with the CALL macro. Its main entry point is entered with a normal save area.

There are several parameters that can be specified on a call to DMKPTRAN. The parameters and their meanings are listed in table 11.

TABLE 11

DMKPTRAN parameters

BRING	Read the page in if not already resident.
DEFER	Don't return to caller until page is resident.
SYSTEM	The page is in the system virtual memory.
IOERR	Return paging errors to caller; don't ABEND.
VFAULT	User virtual page fault.
LOCK	Lock this page in real storage.

8.3.2 DMKPTR operation

At the beginning of DMKPTR another LRA instruction is executed, just in case the entry was via direct call and not via the TRANS macro. The condition code returned by the LRA largely determines which path is taken through DMKPTR. Table 12 details the meaning of the condition codes returned from a LRA instruction.

TABLE 12

Load Real Address conditon codes

CC Bits	BC Mask	Meaning
0	8	Page is resident.
1	4	Segment exception.
2	2	Page exception.
3	1	Length violation.

When a condition code of 0 is returned by the LRA, then the code at the label RESIDENT is executed. This section of code picks up the CORTABLE entry address and branches to TESTLOCK, which checks on the lock status of the page frame.

If a condition code of 1 results from the LRA, then DMKSTRAN is called to handle the segment exception. The segment invalid bit is used to help CP keep track of idle segments for possible migration from drum to disk. DMKSTRAN will mark the segment "valid" and will then return to DMKPTR for re-execution of the LRA instruction.

If a condition code of 3 is returned from the LRA, then an addressing exception is reflected back to the caller since the virtual address was greater than the virtual machine's memory size.

If the LRA instruction results in a condition code of 2, the code beginning at label GETENTRY is executed. A condition code of 2 indicates a normal paging exception. The flow of control is described in the following section.

8.3.3 DMKPTRAN - page fault

If a user causes a page fault by referencing a page whose page table entry has the "invalid" bit set, then the resulting program check interrupt causes control to pass the DMKPRGIN, the first level program check interrupt handler. That routine calls DMKPTRAN with R1 containing the virtual address of the page causing the exception and R11 pointing to the VMBLOK of the virtual machine.

If the page is in main storage but is not connected to this user, then the appropriate pointers in the CORTABLE entry for the page frame are filled in, and the request is treated like a satisfied page-in.

Assuming the page is out of main storage and occupies a DASD slot, the code at label READPAGE will get a page frame from the free list and build a CPEXBLOK with a CPEXADD return address of DMKPTRFD requesting that the page be read into that page frame. It will then chain the CPEXBLOK from the anchor at DMKPTRRQ and exit to DMKPAG. Upon completion of the page-in, the code at label PAGIN will get control. The CORTABLE entry will be updated and the virtual interval timer will be updated, if the page brought in is the user's page 0.

Finally, the code at the label TESTLOCK will check to see if DMKPTR's caller requested the page frame to be locked in storage. If locking was not requested, DMKPTRAN will return to its caller. If locking was requested, the CORIOLCK flag in the CORTABLE entry will be set and the count field CORLCNT will be incremented. DMKPTRAN will then return to its caller.

8.3.4 Free list management

In addition to administering the page translation process, DMKPTR also keeps track of that portion of main storage currently available for paging activity. The free list of page frames initially describes all of the pages in the DPA. Pages are taken from the free list to satisfy all storage requests for page-sized blocks. Pages are added to the free list when its supply of pages becomes low. The threshold value for the minimum number of pages on the free list is the number of virtual machines in Q1 plus the number of virtual machines in Q2 plus 1. Candidates for pages for the free list come primarily from flush list. During replenishment, each page frame on the flush list is examined. If it has been neither referenced nor changed, it is immediately put on the free list. If it has been changed, it is scheduled for page-out, and after that process is completed, the page frame is placed on the free list. If there are no page frames on the flush list, the full CORTABLE is scanned looking for candidate pages. The best candidates are page frames that have not been referenced recently.

8.3.5 Flush list management

The flush list describes page frames whose owners are no longer in the multi-programming set, even though comments in DMKPTR refer to it, inaccurately, as the "user page list". When a virtual machine is dropped from queue, its resident pages are placed on the flush list by means of a call to DMKPTRRS. At queue drop time, no pages are paged-out to backing storage. DMKPTRRS is also called from page migration and virtual machine reset. If the user later references a page that is still in storage, the page is "reclaimed" from the flush list and returned to the user. If the page frame is needed before the user references it again, the page is written out to a page slot on DASD if changed. Pages are also placed on the flush list when there are temporarily no available DASD slots into which to write the page. As described above, when the amount of free storage is low, the flush list is searched for candidate page frames.

8.3.6 SELECT and friends

SELECT is the internal label of the logic in DMKPTR that checks to see if the number of page frames on the free list is below the critical threshold described earlier. If the threshold has been reached, the contents of the flush list are checked. The next frame on the flush list is added to the free list through a call to DMKPTRFT if unchanged. If

changed, then a page-out is scheduled to refresh the DASD copy of the page. After page-out, the frame is placed on the free list as before by a call to DMKPTRFT. After a frame is added to the free list or has been scheduled for a page-out, the threshold of free pages is checked again. This process is continued until the number of pages on the free list rises above the threshold.

If SELECT finds that there are no pages on the flush list, then the CORTABLE is scanned. This logic begins at label SELPAG. SELPAG uses the following logic for maintaining the reference history of the CORTABLE entries. SELPAG maintains a pointer to the last CORTABLE entry it has examined. The scan starts here. A page is chosen when its hardware reference bits are off. If the reference bits are on, they are reset to off so that when the next pass is made they may still be off. At the end, the CORTABLE search wraps to the beginning. When an unreferenced page is found, (and one eventually should be), then the address of the next entry in the CORTABLE is saved so the scan can start there next time. If the unreferenced page belongs to an in-queue user, SELECT counts it as a "steal". If there is no page frame available after two scans, the system ABENDS with a PTR007.

8.3.7 DMKPTRLK and DMKPTRUL

DMKPTRLK and DMKPTRUL are two entry points to manage CORIOLOK, the lock flag in the CORTABLE entry for each frame. DMKPTRLK is called to turn on the flag. If it is already on, the counter at CORLCNT is incremented. DMKPTRUL decrements this counter and if this call brings the counter to zero, it turns off the flag.

8.3.8 DMKPTRXX

This full word flag area, introduced by Lynn Wheeler for his scheduler and page migration support, is used to give system-wide indications of various conditions such as "heavy paging". The primary users of these flags are:

1. DMKSCH, the scheduler.
2. DMKDSP, the dispatcher.
3. DMKPTR, the real memory manager.
4. DMKPGM, the page migrator.

searched by very high speed hardware, and that compares favorably to the two extra memory references that are necessary for normal translation. The TLB and the segment and page tables must of course be kept synchronized, and so when CP changes a segment or page table entry it must also eliminate the corresponding entry in the TLB. That is normally done by purging the entire TLB with the PTLB instruction.

1.3.3.4 Interruptions

Three different program check interrupts can be caused as a result of DAT.

1. A segment-translation program check interrupt signals that an accessed segment table entry contained the "invalid" bit or that the addressed segment was beyond the range of the segment table. The instruction counter portion of the PSW is not updated to point to the next instruction; this allows the interrupted instruction to begin execution again.

2. A page-translation program check interrupt signals that an accessed page table entry contained the "invalid" bit or that the addressed page was beyond the range of the page table. The instruction counter is also not advanced in this case.

3. A translation-specification program check interrupt signals that some invalid configuration was detected in the segment table, the page table, or control registers.

1.3.4 CPU status

The *Principles of Operation* defines 3 different sets of CPU states.

1. The stopped state is entered when the operator presses the STOP key on the system console. The operating state is entered when the operator presses the START key or the RESTART key. Only in the operating state can the CPU execute instructions and accept interrupts.

2. The wait state is entered when a PSW is loaded with bit 14 set to 1. The CPU ceases to execute instructions but it can be interrupted if the appropriate PSW and control register mask bits are set to 1. The running state is entered when a PSW is loaded with

Scheduling and page migration in CP are discussed in other chapters. A complete definition of the DMKPTRXX flags is given in the page migration chapter.

8.4 CP CONTROL BLOCK REQUESTS - FREE STORAGE MANAGEMENT

All buffers needed for control blocks are obtained by calls to entry point DMKFREE and returned by calls to entry point DMKFRET. Table 13 lists the entry points associated with free storage allocation and deallocation. When initial free storage is exhausted, CP allows pages from the DPA to be assigned for temporary free storage usage. The assignment of a DPA page for free use is called "extending". Because of the extra work associated with extend processing, system programmers should carefully monitor free storage usage.

TABLE 13

DMKFRE entry points

DMKFREE	- Main entry to allocate a subpool block or a block from the free storage chain.
DMKFRERC	- Same as DMKFREE except set a return code if no storage is available.
DMKFRERS	- Move and merge subpool entries into the free storage chain. Invoked from DMKUSP (LOGOFF) and DMKTMR (once per hour).
DMKFRET	- Return a block to a subpool or free storage chain.
DMKFRETR	- Free storage initialization and replenishment. Called from CP initialization (DMKSTA) and extend processing (DMKPTRFR).
DMKFRETE	- Move CP control blocks out of the DPA. Put them on the free storage chain, never in a subpool. Called at LOGOFF (DMKUSP).

8.4.1 Free storage initialization

The free storage area is established by module DMKSTA called by DMKCPI at CP initialization. DMKSTA calls DMKFRETR to add the initial pages to the free storage list. The amount of initial free storage is dependent on the total main storage in the system, the existence of a V=R area, and the value of the FREE parameter on the SYSCOR macro in DMKSYS. If the value specified is greater than 25 percent of the real storage (less the V=R area), then the value is ignored. If the value is ignored or is defaulted, the amount of initial free storage is calculated as sum of three pages for the first 256K bytes of memory plus one page for every additional 64K bytes, not counting the V=R area. As a rule of thumb, IBM recommends one page of initial free storage per maximum simultaneous user.

8.4.2 DMKFREE method of operation

All free storage is requested in doubleword units. The subpools range in size from 3 to 33 doublewords, in steps of 3. A subpool is a linked list of blocks of same-sized memory. At CP initialization the number of blocks in subpools is zero. During garbage collection, blocks in subpools are merged back onto the free storage chain to reduce memory fragmentation. From time to time IBM has increased the sizes of subpools as control blocks have increased in size. The subpool entries are given out and returned in a last in - first out (LIFO) manner. IBM has stated that 99 percent of free storage requests are handled out of the subpools. If a module requests a block of storage whose size falls between two subpool sizes, DMKFREE rounds up the request to the next higher subpool size. If the associated subpool size is empty, the request is processed as described below for requests larger than 33 doublewords. If a larger block of storage is found, the request is satisfied by splitting the larger block from the low address side and returning the remainder to the free storage chain. If no equal or larger block is found, the larger subpools are searched in descending order down to the subpool size of the request. If a larger subpool entry is found, the block is split with the remainder going to a smaller subpool if appropriate. If no subpool has an entry to satisfy the request, extend processing is invoked as described below.

8.4.3 DMKFREE requests for larger blocks

If the requested storage size is larger than MAXSPSIZ, currently 33 doublewords, then the free storage chain is searched looking for the correct sized block. While searching the free storage chain, if an equal sized block is found from the initial free storage area, the request is satisfied. If the equal sized block is from extended storage, i.e. the DPA, its address is remembered but the search is continued. When the end of the chain is reached, if no larger or equal sized block is found out of initial free space, the DPA block is used. If a larger sized block out of initial free space is found, the request is satisfied by splitting off the requested size from the high end of the block. The remainder is left on the free storage chain.

8.4.4 Extend processing

If there is no block large enough, DMKFRE declares an extend condition, makes several special provisions, and calls DMKPTRFR to get a page from the DPA. The special provisions include 1) giving the system, via an SVC 16, an extra SAVEAREA because DMKPTR is invoked with a save area, 2) marking the PSA field XTNDLOCK, and 3) saving the BALRSAVE and FREESAVE areas in a field within DMKFRE called EXTNDSAV. This saving of the BALRSAVE area is necessary because routines in CP that are needed to page out a page use BALRSAVE. DMKPTRFR calls DMKFRERS to give a page from the DPA to DMKFREE for use. This call uses FREESAVE. The saving of the FREESAVE solves the reentrancy problem for DMKFRE. After the return from DMKPTRFR, 1) a save area is reserved again via an SVC 20, 2) BALRSAVE and FREESAVE are restored from EXTNDSAV, 3) the XTNDLOCK is turned off, and finally, 4) the register values on invocation of DMKFRE are reloaded from FREESAVE and DMKFRE is reinvoked. The only thing that CP can do during the call to DMKPTRFR is to process the request. Users can not be dispatched and most other functions are suspended. The suspension of all work until the free storage request is satisfied is what makes extending the system costly and to be avoided. While it may be obvious to a system programmer that he may lose control on a call to DMKFREE, it is probably not obvious that he may also lose control on a CALL to a resident module that is invoked by SVC 8. The SVC processing includes getting a SAVEAREA for the called program. If DMKSVC has no more SAVEAREAs to give out, it will call DMKFREE for a SAVEAREA, perhaps causing an extend. This subtlety has caused some grief among CP system programmers.

8.4.5 DMKFRET method of operation

DMKFRET is the entry point in DMKFRE that is called when a CP module is ready to return a block previously allocated by DMKFREE. DMKFRET is called with register 0 containing the number of double words to be returned and register 1 pointing to the block. The returning block is returned to a subpool or to the free storage chain. Any adjacent blocks are merged together to become a larger block. Because DMKFRET deals with returning blocks, it does not care about extend processing.

8.4.6 Subpool returns

When the size of a block falls between two subpool sizes, the request is rounded up to the next larger subpool size. The block is simply returned to the correct subpool by chaining it LIFO. If the returned block is greater than MAXSPSIZ, currently 33 doublewords, it is placed on the free storage chain, merged with adjacent blocks if appropriate, and then checked with the large block logic.

During the merge processing, blocks originating from the DPA are noted. After the merging is complete, if a block from the DPA is large enough, it is carved up on page boundaries and the whole pages are returned to the DPA via a call to DMKPTRFT. The remnants of the original block are chained into the free storage chain. In order to accommodate some modules in CP that call for large blocks with a short life span, DMKFRE checks to see if the previous caller of DMKFRET is the same as the current one. If the callers are the same, then blocks allocated by calls from DMKQCNFT, DMKQCOFT, and DMKVCNFT are not returned to the DPA, but rather are simply placed on the free storage chain. The rationale for this logic is that if some activity is generating many calls from one of these places, performance is better if the blocks are immediately available. Frequent extends are avoided.

8.4.7 Free storage garbage collection

As may be apparent, without some sort of garbage collection, storage would become fragmented over time. CP provides for regular garbage collection by several means. The primary garbage collection is done whenever any user issues the LOGOFF command. Module DMKUSP is called during logoff processing and performs the following activities:

1. Run down all of the allocation block chains for paging and spooling space. If a RECBLOK is found with no records allocated, then call DMKFRET to return it to free storage.

2. After all of the inactive RECBLOKs are returned, call DMKFRERS. This entry point runs through all of the subpool chains and returns all of the entries to the free storage chain, merging adjacent blocks as appropriate. DMKFRERS will also return to the DPA any whole pages it finds if the pages were not part of initial free storage.

3. Check to see if there are pages from the DPA being administered by DMKFRE (extended pages), and if so, then run down 1) the chains of spool file blocks (SFBLOKs), 2) the paging and spooling allocation blocks and associated RECBLOKs, and 3) the 3211 printer index work areas. If any blocks are found from the DPA, call DMKFREE to get a potential substitute block. If the substitute block is also from the DPA, then return the block and exit. As long as the substitute block is from initial free space, copy the contents of the old block into the new block and put the new block onto the current chain. Call DMKFRETE to return the old block to free storage. The reason that a special entry point must be used is that blocks returned via this entry point are not put in the subpools, but are rather merged directly into the free storage chain. Since the subpools are managed as a LIFO stack, if the old blocks were returned with a normal call to DMKFRET, they would be the first ones used when the next call was made to DMKFREE for a potential substitute control block.

A less complete garbage collection is done by a call once an hour from DMKTMR to DMKFRERS to merge the subpool entries into the free storage chain.

8.4.8 Miscellaneous

The chain of free storage is maintained in ascending address sequence. If CP is running on a system with ECPS active, all of the subpool manipulation in DMKFRET and DMKFREE is executed in microcode.

Free storage management is a problem in CP. The algorithms were developed when CP was much simpler. In general, if an installation experiences FREE or FRET problems, IBM will recommend using a trap to ABEND the system when a free storage inconsistency condition arises. The trap is called

the FRE013 trap because FRE013 is the ABEND code when the trap is executed. The trap keeps extra information, basically the user who requested the block and the size of request, for every block given out. If a block is given back on behalf of a user different from the one who requested it originally, CP will ABEND. The same ABEND occurs when the amount of space a caller returns differs from that originally allocated. Problems of a general class called "storage eaters" can be resolved with the help of the FRE013 trap. Unfortunately there is overhead involved with maintaining the additional information. Some installations have chosen to accept the additional overhead and have installed the trap on their normal system. They believe it is worth the overhead to detect storage problems induced either by IBM or by the installation itself. IBM has incorporated the FRE013 trap into release 4 of VM/SP, where it is called the "storage overlay trap".

More information about ABEND is given in the chapter on the CP trace table.

8.5 HPO CHANGES

The fourth reference listed at the beginning of this chapter describes some simulation work done to improve the existing FREE/FRET algorithm. Basically there are two subpool groups with different element sizes. Requests for storage from 2 to 128 doublewords are handled with a set of subpool chains that vary by multiples of 2 (2, 4, 6, 8, ..., 128). A second set of subpools handles requests for storage from 129 through 512 doublewords. Notice that 512 doublewords is 4096 bytes of storage. The increment in size between subpool chains in this second group is 32 (160, 192, ..., 512).

A second change involves the decision of what to do with a piece of returned storage. As you recall, with the SP algorithm all pieces of storage smaller than 33 doublewords were put back into the appropriate subpool in LIFO order. The HPO algorithm maintains a double set of subpools for each element size. The second set contains elements coming from the DPA (extended storage). Only when the first subpool is empty will the corresponding extended storage subpool be searched. This approach alleviates the "stickiness" associated with extended storage by reducing its frequency of use.

The HPO rewrite of DMKFRE also modifies the garbage collection algorithm used in DMKFRERS. First, a period of time must have expired before any collection will be done. The default time interval, located at label RSMIN, is 1 second. If a subpool element is in the extended storage subpool

chain, then it is put back onto the free storage chain. If the element is from initial free storage, then the element must have been unused for an amount of time before it will be returned to the free storage chain. For blocks from 2 to 128 doublewords, the idle time must be greater than the value at THRSHLD, 60 seconds. For blocks from 160 to 512 doublewords, the time value is shifted right by the value at BIAS. The EQU value of BIAS is 2, giving a minimum time of 15 seconds for these larger subpool sizes.

A small criticism seems appropriate. While the algorithmic changes described above seem reasonable, the simulation studies used to justify the changes are not convincing. No information was given as to the type of terminals being used. In particular, full-screen 3270 applications tend to use many large storage blocks for terminal I/O. That would not be the case if the terminals were line mode ASCII devices, and the resulting subpool usage could be quite different.

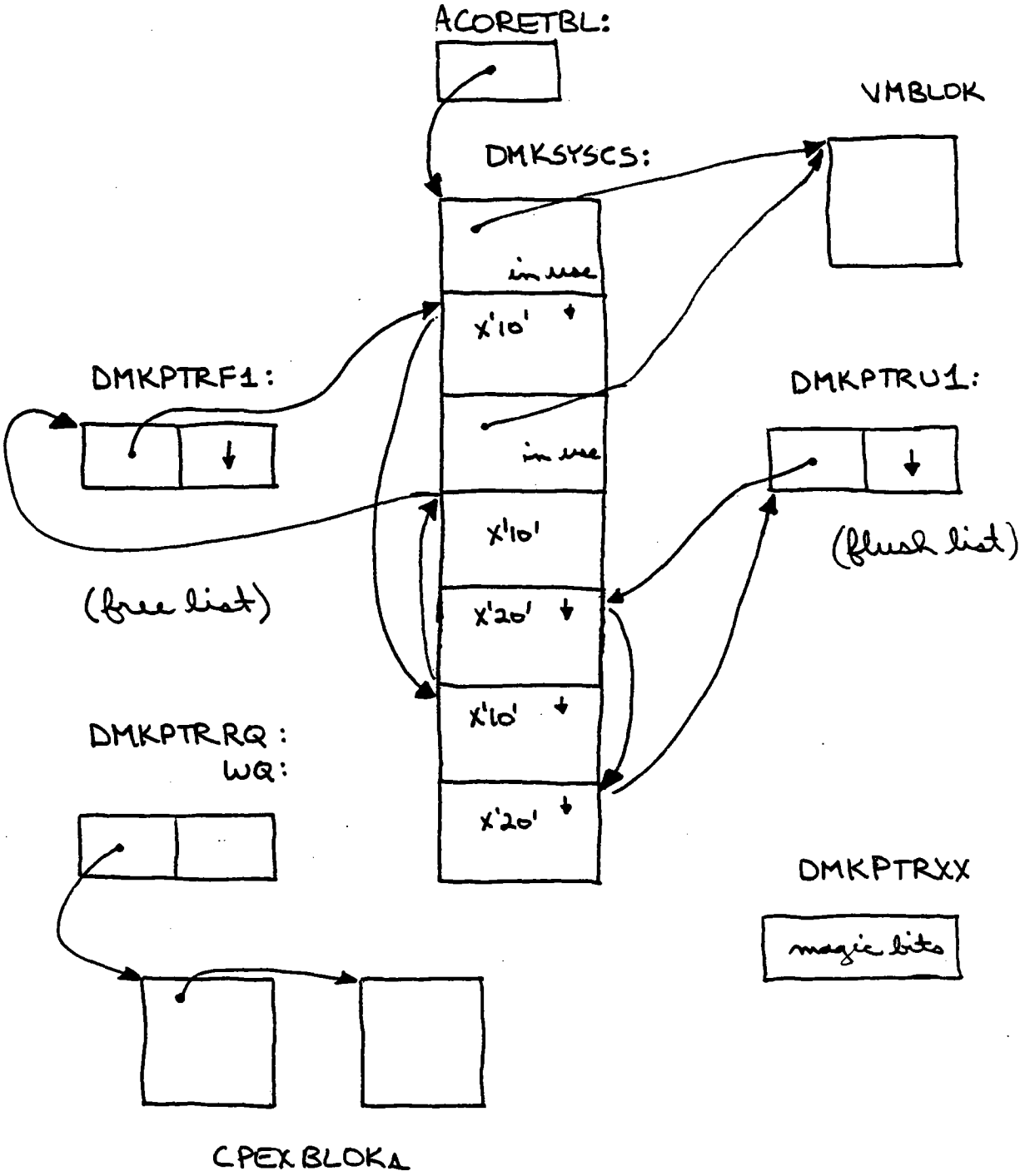
Note that the HPO algorithms are not used on a system where the ECPS assist for DMKFRE is operational. Two things can be concluded from this fact. The first is that if HPO is run on a 43xx systems with ECPS, the older, and faster, algorithms will be used. A second result is that you can not arm and disarm the ECPS call dynamically with the system up and running. With the base SP implementation, the ECPS microcode instruction could be NOPed on-the-fly for debugging purposes becaused the software and the microcode use the same subpool algorithm. Since the HPO software uses the existence of a NOP instruction to signal that it should use the "improved" algorithms, the system will die a horrible death if the NOP is suddenly changed to an operational ECPS micro instruction. You can, of course, turn off the ECPS micro instruction by modifying CP initialization before the free storage system is initialized.

The simulation studies showed that about three free storage pages were needed for every logged on user. This number agrees with the experience of other CMS-intensive installations. Therefore, to avoid extending, you should specify on the SYSCOR macro a number of pages approximately equal to three times the number of simultaneous users you expect on your system. This is about three times the amount of free storage that IBM recommends.

8.6 SUMMARY

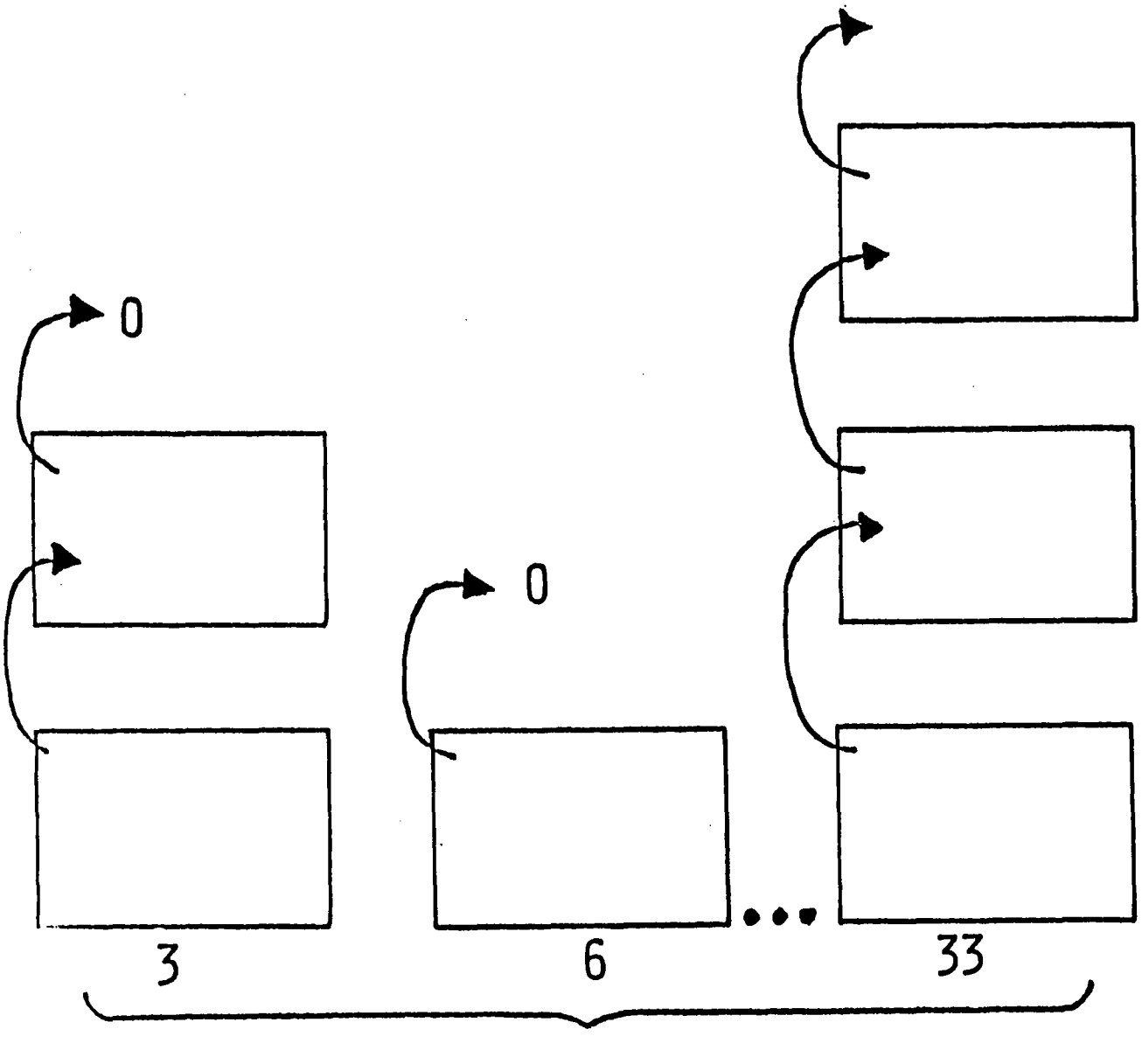
This chapter has dealt with the two different kinds of real storage that exists within CP. Each kind serves a different need. DMKPTR services requests by users and CP for main storage pages. DMKFRE services requests for control blocks allocated in doubleword blocks. Storage management is an important function that is critical to performance. CP has developed algorithms that meet most of the performance needs. The one weak characteristic is extend processing. Extending is done whenever DMKFRE can not satisfy a request for storage from the free area. DMKFRE must call DMKPTRFR to get a page of the dynamic paging area to continue servicing requests. While this processing is in progress, other work is suspended.

Installations can explicitly assign storage to different functions by specifying parameters on the SYSCOR macro during system generation. Free storage usage, in particular, should be closely monitored by a system programmer.



Dynamic Paging Area Overview

FREE STORAGE MANAGEMENT



NOTES

bit 14 set to 0. The CPU executes instructions at full speed or at reduced speed, depending upon the setting of the system console rate selector keys.

3. The CPU is in problem state when PSW bit 15 is 1 and in supervisor state when the bit is 0. Only some CPU instructions are allowed in problem state; the privileged instructions will be terminated with a program check interrupt.

1.4 I/O SUBSYSTEM

1.4.1 Overview

The System/370 I/O subsystem is quite complex but it is characterized by a few simple principles. In effect, the following process takes place:

1. A program builds a series of commands for an I/O device and issues a Start I/O (SIO) instruction to initiate the commands.

2. An independent processor, the I/O channel, takes the commands and begins executing them with the help of an I/O control unit and the requested I/O device. The CPU is freed to continue processing if it can. If it can do no more work, the CPU may enter the wait state.

3. When the I/O operation is complete, the channel causes an I/O interrupt and stores status bits into memory. The resulting PSW swap causes an I/O first level interrupt handler to start executing. Finally, the original requestor of the I/O is informed that the I/O has completed so that the requestor can continue processing.

1.4.2 Hardware devices

The hardware of the I/O subsystem can be organized into three types of components that are connected in a particular arrangement.

1. Attached to the CPU is the channel; a given CPU may have up to 16 channels. Each channel operates independently of the other channels, at least from a software point of view. The channel contains one or more subchannels, each of which is basically a processor that can execute a series of channel commands. There are three types of channels:

Chapter 9

PAGING

9.1 INTRODUCTION

9.1.1 Overview

The CP paging system is conceptually straightforward. CP is strictly a demand paged system. It never tries to page-in a virtual machine's pages until the virtual machine attempts to reference them. This ignorance of history has its strengths and weaknesses. At the invocation of the next command, the system does not try to anticipate which pages the virtual machine is likely to use.

9.1.2 References

9.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

3. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

4. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

9.1.2.2 CP modules

1. DMKFMT - performs standalone disk formatting. (Note that this is *not* a part of the CP nucleus.)

2. DMKPAG - constructs paging IOBLOKs.

3. DMKPAH - processes paging I/O interrupts.

4. DMKPGT - allocates DASD slots.

5. DMKPGU - deallocates DASD slots.

6. DMKPTR - requests page I/O operations.

7. DMKSYS - contains variables used by paging.

9.1.3 Preview

The paging subsystem is driven by the module responsible for real storage management, DMKPTR. If a page the virtual machine wants is not in any of the real storage management queues, DMKPTR requests that the page be read into memory. If real memory frames are in short supply, it is DMKPTR that decides which changed pages need to be written out and requests that it be done.

The paging subsystem relies on several parts of CP's architecture that we need to review. We will then examine the modules executed to perform a paging request and the associated control blocks. We will cover at some length the algorithms for page selection and I/O performance optimization. Finally we will follow a page-in operation from page fault through page posting.

9.1.4 System DASD areas

There are several kinds of DASD volumes known to CP. The only kind of importance to the paging subsystem is the so-called "CP-owned" volume. The volume serial names of CP-owned volumes are declared with the SYSOWN macro in the DMKSYS data module. Notice that there is a nomenclature problem here. Another kind of volume that CP understands is a so-called "system" volume. This kind of volume contains only user minidisks. Its name does not appear in the SYSOWN macro. The usage of the cylinders on a CP-owned volume is declared in the allocation map written on cylinder 0 by the standalone utility program DMKFMT. The kinds of usage that CP understands are:

1. PAGE - preferred paging.

2. TEMP - spool files and overflow paging.

3. DUMP - CP ABEND real memory dumps.

4. DRCT - system directory.

5. TDSK - temporary minidisks.

6. PERM - minidisks, CP nucleus, etc.

The first four cylinder types are page formatted, as described below. For most purposes page space, temp space, and dump space are handled in the same way by the paging subsystem.

9.1.4.1 Page space

Page space refers to that space so declared in the allocation record. The cylinders set aside for paging will be used only for paging slots. CP automatically recognizes when paging space is defined over fixed-head cylinders. The IBM 2305 is completely fixed-head. The IBM 3350 can have a feature making cylinders 1 and 2 accessed by fixed heads. There may be other drives that will be offered with a fixed head area. Since the use of these fixed head areas for paging can add significant performance to a system, CP manages these areas specially. The paging space not defined under fixed heads is called moveable head paging space. Both of these kinds of paging space are known as "preferred" paging space.

9.1.4.2 Temp space

The temp space mentioned above serves two functions. First and foremost it is the space in which spool files reside. It is, therefore, also known as spool space. However, it has a second major function. It is the space in which pages for the paging subsystem are placed when the preferred fixed head and moveable head spaces are full.

9.1.4.3 Dump space

A separate page-formatted dump area is a recent addition to CP. If there is no explicit dump space defined, CP will allocate space for system dumps out of the temp space. Unfortunately, the dump program must have contiguous DASD slots for a dump. If there has been high spool usage, there is a non-trivial probability that there are not enough contiguous DASD slots into which to write a dump. If this condition arises, the dump will fail. In order to solve this reliability problem, IBM recently added the ability to specify in the allocation record of a CP-owned volume an area to be used only for dumps. More about dumps is given in the chapter on trace table and dumps.

9.1.5 Paging hierarchy

The description of the varieties of paging space should suggest a hierarchy. CP does in fact manage the paging space as a hierarchy with fixed head (FH) preferred areas used first. Moveable head (MH) prefered space is used when the fixed head space is full. Finally the temp space is used when the moveable head space is full. This hierarchy allows CP's paging subsystem to optimize performance based on the kind of DASD devices it has to manage.

9.1.6 DMKFMT utility program

The DMKFMT utility program ("Format/Allocate") is a standalone IPL-able program that page formats cylinders of a volume and marks the usage of the cylinders (or blocks for FBA devices). The exact format of each track is device dependent. All modern devices have the track capacity to fit several pages on a track. In order to get good performance, DMKFMT will write filler records between the 4K page records if the track size is large enough. The existence of these filler records allows the device to head switch between page slots without losing a revolution.

9.1.7 SYSOWN macro

The SYSOWN macro in DMKSYS allows you to specify by name which volumes are CP-owned. All paging, temp, and dump space must be on volumes named in this macro. Each name on the SYSOWN macro generates a 6 byte field with the EBCDIC name and a 2 byte field. Initially the macro fills in the two byte field with a value of X'FFFF'. At CP initialization, if a volume with the entry's name is found on-line, the two byte field is filled in with the halfword displacement from the beginning of the real device blocks to the real device block that represents the address where this named volume was found.

A word of caution is needed. When you decide to add a new name to the SYSOWN macro, you should add the new name after all the existing volumes containing temp space. The reason for this restriction is that spool files have a compressed DASD slot address imbedded in them. By changing the absolute order of the volume names, you will change the slot addresses. Spool files on the volumes whose order has changed will be useless and a cold start of CP will be needed. A method of circumventing this problem if it is necessary to add a volume name in the middle of the existing SYSOWN names is to dump the spool to tape with the SPTAPE

command, shut down the system, and then IPL the new system with the altered SYSOWNed list. It is much simpler to just add the new name to the end of the SYSOWN list.

9.1.8 Allocation map

The allocation map, which is written on every DASD volume formatted and allocated by DMKFMT, is the master information for CP initialization to set up the paging, temp, dump, and other areas. For CKD devices the allocation map contains one byte for every cylinder on the device. The possible uses that can be specified and their associated values are given in Figure 23.

Flag Value	*Usage*
00	Temporary
01	Permanent
02	T-disk
04	Directory
0C	Directory - in use
08	Paging
10	Dump
FF	End of Volume

Sample Allocation Record

01	01	08	08	08	08	08	01	01	01	01	01	01	01	01	FF	na

Figure 23: Allocation record for CKD devices

Notice that a X'FF' is used to mark the first non-existent cylinder in the volume. Using this layout the allocation map can be fixed length. In the sample allocation record displayed in Figure 23, the first two cylinders are reserved for permanent space. The next five cylinders are designated paging space and the last eight cylinders are also permanent space. This allocation record does not describe any known device but is used only for illustrative purposes. The al-

location record is always written on cylinder 0, head 0, record 4 on CKD devices.

FBA devices have their allocation records written in blocks 3 and 4 of the volume. These two records are a logical extent map of the assigned page usage. Each entry is 12 bytes long, consisting of a flag of the same value as described for CKD devices and a start and end page number for the extent. Each page number requires four bytes of space.

9.1.9 Cylinder format

DMKFMT writes as many 4K page-sized records on a track as will fit. Specifically, each type of DASD is fitted with as many records as possible. Filler records are written to allow a head switch without the device losing a revolution. The 3350 cylinder layout is included in figure 24 as an example.

A cylinder's page-sized records are numbered sequentially beginning with 1. Using the 3350 as an example the first page slot on a cylinder has a record number of 1 and the last page has a record number of 120.

Since cylinder 0 has the volume label, the allocation record, and other information, it does not have the full complement of page slots. For example all of track 0 of cylinder 0 on a 3350 is used up by volume information. However, the 2305 cylinder 0 track 0 has one page slot.

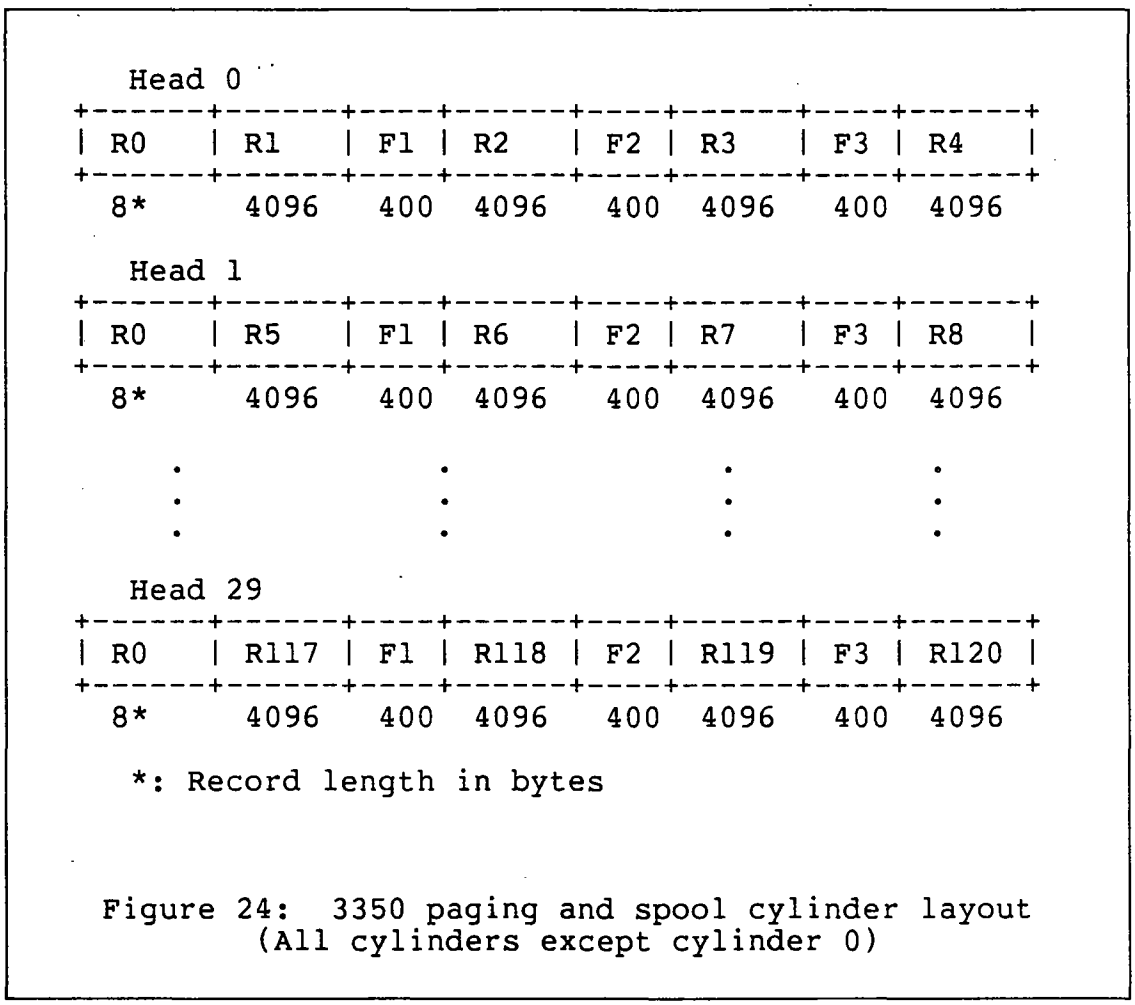


Figure 24: 3350 paging and spool cylinder layout (All cylinders except cylinder 0)

9.1.10 Internal compressed page addresses

Throughout the paging and spooling subsystems, CP keeps track of the external DASD slots with a compressed four byte address. This address takes advantage of the fact that the page slots on a cylinder are numbered in ascending sequential order. To uniquely identify each DASD paging slot, CP uses the format illustrated in Figure 25. For count key and data (CKD) devices, such as the 3350 and 3380, the slot address is of the form "CCPD", where "CC" is the cylinder number on the device, and "P" is the record number on that cylinder, starting with 1. The "D" is the position of the DASD volume in the SYSOWN list, starting with 0. For fixed block architecture (FBA) devices, such as the 3310 and 3370, the compressed address is represented by "PPPD", where "PPP" is the relative page number, starting with 0.

C C P D for CKD devices

P P P D for FBA devices

Figure 25: Four-byte internal compressed slot address

9.2 PAGING OVERVIEW

9.2.1 Modules

The modules of the paging subsystem are given in table 14.

TABLE 14

Paging subsystem modules and functions

DMKPTR -	requests page-ins and page-outs.
DMKPAG -	builds IOBLOKs with CCWs.
DMKPAH -	posts page-ins.
DMKPGT -	allocates DASD paging and temp slots.
DMKPGU -	deallocates paging and temp slots; allocates and deallocates system virtual addresses.
DMKIOS -	schedules I/O on a real device.

A user page fault in System/370 is signalled by a program interrupt code X'10' for a page exception or by code X'11' for a segment exception. Module DMKPRG gets control on all program exceptions. DMKPRG calls DMKPTR after determining that the exception is a paging or segment exception. If the page is still in memory on the free or flush list, the page is connected to the user's virtual memory and no further processing is needed. The page is said to be "reclaimed". If the page is not present in main storage, then DMKPTR schedules a page-in by building a CPEXBLOK and putting it on

the queue of page-in requests anchored at DMKPTRRQ. During DMKPTR's management of real storage frames, if a page-out is needed (because a changed page frame is required for some other purpose), DMKPTR builds a CPEXBLOK and chains it to the list anchored at DMKPTRWQ. After a CPEXBLOK has been put on either the read queue or the write queue, DMKPAG is entered via GOTO.

If there are other requests for the same page on the same device, as in the case of a page shared by several users, DMKPAG simply chains the CPEXBLOK to the existing IOBLOK. Normally DMKPAG must build an IOBLOK that contains the channel program to be executed to read or write the page. It chains the CPEXBLOK from the IOBLOK for later execution when the page-in or page-out completes. If there are other paging operations for the same device and cylinder, DMKPAG will slot-sort the requests with device dependent algorithms to enhance the performance of the paging subsystem. Finally DMKPAG calls DMKIOSQR to schedule the actual I/O.

After the I/O operation completes, the interrupt return address(IRA) from the page IOBLOK gets control. The entry point at the beginning of DMKPAH executes the code to unchain the CPEXBLOKs from the IOBLOK and call DMKSTK to stack them for later execution. After all of the CPEXBLOKs are stacked, the IOBLOK is put back on DMKPAG's list of pre-formatted IOBLOKs. Later the CPEXADD address in each of the CPEXBLOKs gains control; the return address is in DMKPTR, which will post the paging operation complete, allowing the user to continue execution.

9.2.2 Control blocks

9.2.2.1 SWPTABLE

SWPTABLEs are the control blocks that map a user's virtual memory into the DASD page slot space. There are 256 segments possible in a 16 megabyte address space. Each SWPTABLE maps one segment of user virtual memory. A segment is 64K bytes or sixteen 4K pages. The SWPTABLE header has a pointer to the VMBLOK, a segment table index number for this segment, a flag (SWPFLAG2) of status information about all of the pages in this SWPTABLE, and a pointer to the PAGTABLE for the segment. Each SWPTABLE entry for a page of user virtual memory has the page number within the segment, the virtual storage keys for the two 2K blocks making up the page, the CCPD of the page, and a flag to mark the status of the individual page. The definitions of the flag bits are given in table 15.

a) The selector channel contains exactly one subchannel. That means that the channel can execute exactly one channel program at a time, controlling exactly one device. Selector channels are used with tapes and channel-to-channel adapters.

b) The block multiplexor channel contains several subchannels, so that several channel programs can be active at a given time. However, at any one time only one of the subchannels can be transmitting a block of data over the channel, so the various subchannels must be able to disconnect from the channel during mechanical motion or other periods when the devices are idle. The overall effect is that records from several devices may be interspersed as the corresponding channel programs are executed. Block multiplexors can control disks, display terminals, and some "unit record" devices such as printers and card readers. If selector-type devices are attached to the block multiplexor channel, then the operation of those devices forces the channel to operate in selector channel mode for the duration of the channel program.

c) The byte multiplexor channel also contains several subchannels and can therefore also execute several channel programs at once. The subchannels share the channel by using it for just a few bytes at a time. The overall effect is that all subchannels appear to be executing and transmitting data simultaneously. Byte multiplexor channels can operate devices such as teleprocessing lines and some card readers and printers.

2. The control unit is attached to the channel and operates as an intermediary between the channel and the device; this is done mainly to conserve hardware and reduce cost. The control unit manages device positioning, which can often be performed without using the channel. A typical controller may support 8 tape drives or 8 or 16 disk drives. Most controllers can simultaneously position each of the attached devices and can transmit data to or from any one of the devices. A controller that is busy transmitting data with one device will return a "control unit busy" if asked to perform another data transmission. Some controllers are integrated with the associated device.

3. The third component is the device, the unit that actually performs the I/O operation. Devices are such things as tape drives, disk drives, terminals, card

TABLE 15

SWPTABLE flag definitions

Flag	Name	Definition
80	SWPTRANS	Page in transit
40	SWPRECMP	No need to deallocate CCPD*
20	SWPALLOC	Page scheduled for allocation
10	SWPSHR	Page is shared
08	SWPREF1	First 2K referenced
04	SWPCHG1	First 2K changed
02	SWPREF2	Second 2K referenced
01	SWPCHG2	Second 2K changed

* SWPRECMP also means "recompute the CCPD"

9.2.2.2 IOBLOK and extension

The standard CP IOBLOK is discussed in another chapter. Briefly, it has flag bytes, the cylinder number of this request, a pointer to the user's VMBLOK, a CAW, and an interrupt return address(IRA) to which this request will be unstacked. Extensions to the IOBLOK are made for all paging requests. The extension for CKD devices contains a standard DASD channel program and the associated cylinder, head, sector, and record number of the DASD slot. The end of the DASD channel program is a SENSE channel command word. This last CCW is changed to a Transfer-in-Channel(TIC) to branch to another IOBLOK's channel program if there are multiple requests for page slots outstanding.

A further redefinition of the IOBLOK is made if the paging request is to an FBA device. A normal FBA channel program is contained in this extent with the provision of chaining to another IOBLOK's channel program with a TIC CCW if appropriate. On earlier hardware, the operation code used to hold the place in the channel program if the TIC was not needed was a SENSE. The SENSE operation was used in order to "preserve track orientation" (which is meaningless at the end of a channel program). Unfortunately the SENSE operation to a 3880 control unit can cause a missed revolution. Therefore IBM introduced a PTF to change the ending operation code to a NOP.

9.2.2.3 ALOCBLOK and friends

The ALOCBLOK for CKD devices and its equivalent for FBA devices, the ALOFBLOK, are the major control blocks for information about the status of paging and spool slots. There is one ALOCBLOK per device, chained from the RDEVBLOK and threaded through all the chains anchored in DMKPGT for DASD devices with areas of declared usage. As an example of a complex case, if a 3350 has a fixed head area and its allocation record says that it has bands of cylinders for paging and temp usage, its ALOCBLOK may appear on three ALOCBLOK chains, one for each type of space (FH, MH, or temp). ALOCBLOKs are built by CP initialization or by module DMKVDG, the ATTACH command, when a CP-owned volume is attached to the system. Although the functions are identical, the formats of the ALOFBLOK and the ALOCBLOK are sufficiently dissimilar to force modules that deal with them to have two paths, one for FBA devices and one for CKD devices. If new function is added for DASD, then both legs of code must be updated.

9.2.2.4 RECBLOK

The RECBLOK is the control block that contains the status of a particular cylinder's contents. It is chained from the ALOCBLOK according to the use made of the cylinder (FH, MH, or temp). It exists on a chain with cylinder numbers closer to the middle of the device chained ahead of cylinders further from the center. There are two bytes reserved for the cylinder number that this RECBLOK represents. There is a counter of maximum pages and allocated pages on this cylinder. There is a RECFLAG that marks the usage of this cylinder and also records if the cylinder is full. Finally there is a bit map of the pages on the cylinder. A zero in the map signifies that the page is available for allocation. A one means that the page is already allocated. For FBA devices there is an extension to the base RECBLOK. This extension records the real first page number and last page number of the pages represented by this RECBLOK. The RECMAP is a bit map whose first bit represents the first page in the extent.

9.2.2.5 RDCBLOK

There is a new control block for FBA devices called the real device characteristics block, RDCBLOK. There is one block for each type of FBA device attached to the system. The anchor for the chain is in the PSA. The RDCBLOK records the physical characteristics of the device. The one character-

istic of the device that the paging subsystem uses is the number of pages per real cylinder. This number is calculated and used for building new RECBLOKs as appropriate. A pointer to a device's RDCBLOK is found in the RDEVBLOK.

9.3 OPERATION OF DMKPGT

DMKPGT's main preoccupation is allocating available DASD slots for paging and temp functions as close as possible to the center of the device. The program returns a page slot's CCPD to the caller.

DMKPGT has anchors for each type of device that can have an ALOCBLOK. In fact DMKPGT has an anchor for each device type within a device usage. There is at most one ALOCBLOK per device. Threaded through each ALOCBLOK are pointers to the next ALOCBLOK with the same kind of space. For example, there is a chain of ALOCBLOKs for 3350s with temp space, 3350s with page space, and possibly, 3350s with fixed head page space. Each device can be on each chain. There is an anchor for each usage type and device. For example, DMKPGT5F, DMKPGT5M, and DMKPTR5T are the anchors for 1) fixed head 3350 preferred paging, 2) movable head 3350 preferred paging, and 3) temporary 3350 space for spooling and overflow paging respectively.

All entry points in DMKPGT run through a three-level nested loop. The outermost loop looks down the three lists of ALOCBLOK anchors. Each anchor points to a circular chain of ALOCBLOKs that represent all the volumes of the same device type (such as 3350) that have been declared to have cylinders with the same function (FH, MH, or temp). The ALOCBLOK has several counters for the usage of the cylinders on the device it represents. One of the fields, ALOCMAX, is the maximum number of cylinders on the device. The next two counters count the number of cylinders allocated by type of use. ALOCUSEP is for paging cylinders used, and ALOCUSET is for temp cylinders used. These two counters are initialized to the value of the maximum number of cylinders on the volume minus the number of possible cylinders to be allocated for that particular use. After incrementing the appropriate counter a simple comparison between that counter and ALOCMAX indicates whether there should be additional cylinders available on the device for this type of usage.

The middle loop in DMKPGT searches down the ALOCBLOK chain looking for an ALOCBLOK with an available page slot. If no page is available, a new cylinder is allocated for the function being sought and the first RECBLOK for that cylinder is created. Whenever it is necessary to allocate a new cylinder, the ALOCBLOKs on the chain are relinked so that

pages are allocated first from the device with the most recently allocated cylinder. Moving the ALOCBLOK of the newly created RECBLOK to the head of the ALOCBLOK chain for this device type and page slot usage produces a rough round-robin allocation algorithm. The chain of RECBLOKs from each of the anchors in the ALOCBLOK is ordered by the distance the cylinder is away from the middle. Using this scheme, CP will always allocate the available slot closest to the middle of the volume.

The third and innermost loop is the search through the RECBLOKs anchored in the appropriate ALOCBLOK field. The ALOCBLOK contains four anchors for chains of RECBLOKs. The first is ALOCPGFH, for fixed head paging cylinders. The second is ALOCPGMH, for moveable head cylinders. The third is ALOCRECS, for spool usage of temp cylinders. The fourth is ALOCRECP, for general (non-preferred) page usage of temp cylinders.

Although there are several entry points in DMKPGT, they can be classified into two categories. The most often called entry points allocate one page at a time for the paging and spooling functions. Other entry points allocate contiguous pages on temp or dump space for system and 370X dumps. The entry points to DMKPGT are enumerated in table 16.

TABLE 16

Entry points of DMKPGT

DMKPGTPG	Allocate a slot of user virtual memory.
DMKPGTSG	Allocate a slot of user SPOOL space.
DMKPGTPM	Allocate a slot of temp space for paging. Used by page migration.
DMKPGTCG	Assign contiguous slots for 370X dumps.
DMKPGTDT	Allocate contiguous slots for system dumps. Called by DMKPGUDT.
DMKPGTDG	Allocate page slots for the SPTAPE command. Called by DMKPGUDG.

9.4 OPERATION OF DMKPGU

DMKPGU performs two major functions. Entry points DMKPGUPR, DMKPGUSD, DMKPGUSP and DMKPGUDU are called by other CP modules to disassociate a user virtual memory address from a given DASD paging slot. The CCPD of the slot to be deallocated is picked up from the SWPCYL field for the specified virtual memory address. The RDEVBLOK of the device is found from the 'D' part of the compressed address by a call to internal subroutine FINDEVIC. The ALOCBLOK for the device is pointed to by the RDEVALLN field. The RECBLOK representing the correct cylinder (the CC in CCPD for CKD devices and the PPP in PPPD for FBA devices) is located. The bit that represents that page is turned off in RECMAP. DMKPGUPR is called to deallocate one slot from a user's memory. DMKPGUSD is called to deallocate one slot of spool space. DMKPGUSR is called to deallocate a set of spool slots. The other entry points associated with this function allocate or deallocate multiple slots with one call. The system dump function and certain spool file functions call these entry points.

TABLE 17

Entry points of DMKPGU

DMKPGUPR	Deallocate a slot of user virtual memory.
DMKPGUSD	Deallocate a slot of user SPOOL space.
DMKPGUSP	Deallocate a slot of system virtual memory.
DMKPGUSR	Deallocate a set of SPOOL slots.
DMKPGUDU	Deallocate contiguous slots for system dumps.
DMKPGUDT	Allocate contiguous slots for system dumps.
DMKPGUDG	Allocate page slots for the SPTAPE command.
DMKPGUVG	Assign a system virtual address.
DMKPGUVR	Release a system virtual address.

The second major function of DMKPGU is invoked via calls to DMKPGUVG and DMKPGUVR. These entry points assign and return addresses in the system virtual memory. While CP does not run with DAT enabled, it does use virtual addresses when referring to DASD page slots, thereby allowing CP to use the same page I/O routines for its own use that it uses for virtual machine page I/O. This system virtual memory is used

extensively by the directory updating process and the SPOOL system (see those chapters for more detail).

At label PAGTABL there is a 160 byte field that is the bit map of system virtual addresses; this provides for a maximum of 1280 pages in the system virtual address space. When DMKPGUVG is called, the field at PAGTABL is searched for the first available bit. When the first zero bit is found, the virtual address the bit represents is constructed and returned to the caller. If no zero bit is found, then a PGU005 ABEND results.

The return of a page of system virtual memory to the available addresses is performed by DMKPGUVR. A process similar to DMKPGUVG, but in reverse order, is carried out. The virtual page address is transformed into the correct bit and byte offset in the PAGTABL field. The corresponding bit is turned off if it was on. A PGU006 ABEND results if the bit was already off.

CP is very good about consistency checking throughout the page allocation and deallocation process. CP will ABEND if any logical inconsistency is detected.

9.5 OPERATION OF DMKPAG

9.5.1 Introduction to DMKPAG

DMKPAG has two responsibilities. First it builds IOBLOKs needed to satisfy the paging request. Then it slot-sorts the existing paging requests to optimize the throughput of the paging subsystem.

9.5.2 Building IOBLOKs

DMKPAG gets control via GOTO to work off paging requests queued from DMKPTRRQ and DMKPTRWQ. Each of the CPEXBLOKs on these queues represents one user's request for one page not in real memory. DMKPAG maintains two stacks of preformatted paging IOBLOKs. IOBLOKs used by the paging system are larger than normal and contain the channel programs for the page request. When a stack is exhausted, DMKFREE is called to get storage for the IOBLOK. The first type of IOBLOK is used to satisfy paging requests for normal CKD and FBA devices. The second, recently developed, is used to request pages from what are called 'extended CKD' devices. Extended CKD devices are devices running with the speed matching buffer feature in an IBM 3880 storage director. Though the types vary in detail, the logic is the same. An IOBLOK must

be initialized and the channel programs filled with request dependent data. The CCPD of the requested DASD slot is transformed into sector, cylinder, head, and record numbers.

The requesting CPEXBLOK is chained to the in-transit queue anchor in the IOBLOK if no other IOBLOK exists for the same page. If another IOBLOK exists, this request is simply added to the CPEXBLOK chain anchored at IOBMISC.

If there is no other request for the same page, DMKPAG scans the pending IOBLOKs for the device looking for requests for the same cylinder. If there are none, DMKPAG queues an I/O request by calling DMKIOSQR. If other IOBLOKs for the same cylinder exist, DMKPAG sorts the requests by slot number using a device dependent algorithm. All records on a cylinder having the same angular displacement on the track are said to be in the same slot. A minimum optimization is to re-chain the channel program to bypass the SEEK of the second and subsequent requests to a cylinder. Most of the devices have record layouts for paging to allow optimization of this critical function. The IBM 3350 page format, for example, has 400 byte filler records inserted between page-sized data records to allow head switching without loss of a revolution.

For requests to the same relative slot (angular displacement) on the cylinder, there is a priority scheme to decide which request is scheduled first, as shown in table 18. The purpose of this scheme is to guarantee good response time to requests from users who are currently competing for cpu resources. After the request has been slot-sorted with other requests, DMKPAG starts again at the top to process the next page request or else exits if the read and write queues are empty.

TABLE 18

Relative priority of paging requests

User Status	Request Type	Priority
Q1	Read	1
Q2	Read	2
Q1	Write	3
Q2	Write	4
E1	Read	5
not E1	Read	6
E1	Write	7
not E1	Write	8

Note: Request is for same slot on same cylinder.
Q1 is the dispatcher's queue 1 list of VMBLOKs.
Q2 is the dispatcher's queue 2 list of VMBLOKs.
E1 is dispatcher's eligible list.

9.6 OPERATION OF DMKPAH

DMKIOT stacks the IOBLOK as a result of the interrupt at the end of the paging channel program. DMKPAHIO then gets control as the execution address specified in the IOBIRA. If there is an error with the I/O, DMKPAH reschedules I/O initialization. After five unsuccessful retries, DMKMCHST is called to terminate the system.

When the I/O is successful, DMKPAH stacks the in-transit CPEXBLOKs for later execution. DMKSTKCP is called for each CPEXBLOK chained from the IOBLOK. Each CPEXADD points to DMKPTR, which exits to its original caller with the page now resident (for a page-in request) or written out (for a page-out request). DMKPAH replenishes the supply of preformatted paging IOBLOKs by chaining the IOBLOK to the chain anchored at DMKPAGSK for FBA and normal CKD devices. For extended CKD devices the returning IOBLOK is chained to the list anchored at DMKPAGEX.

9.7 SUMMARY

The most important point to remember about this chapter is that CP is a demand paged system. Its paging subsystem works as an extension to its real memory management. Path lengths for non-error paths in the paging subsystem are very short.

The module names to remember are:

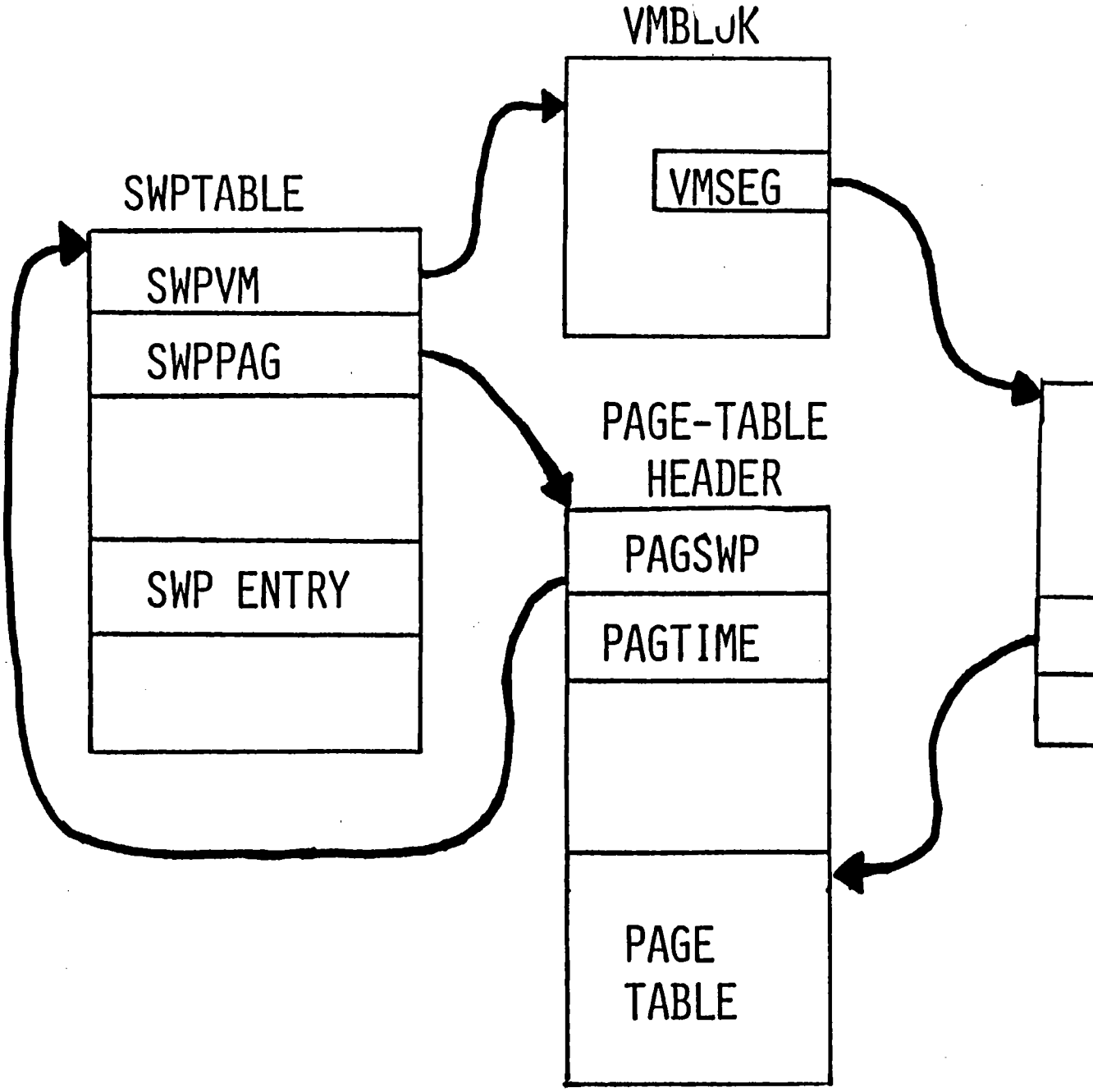
1. DMKPTR.
2. DMKPGT - DMKPGU.
3. DMKPAG - DMKPAH.
4. DMKIOS.

The control blocks associated with paging are:

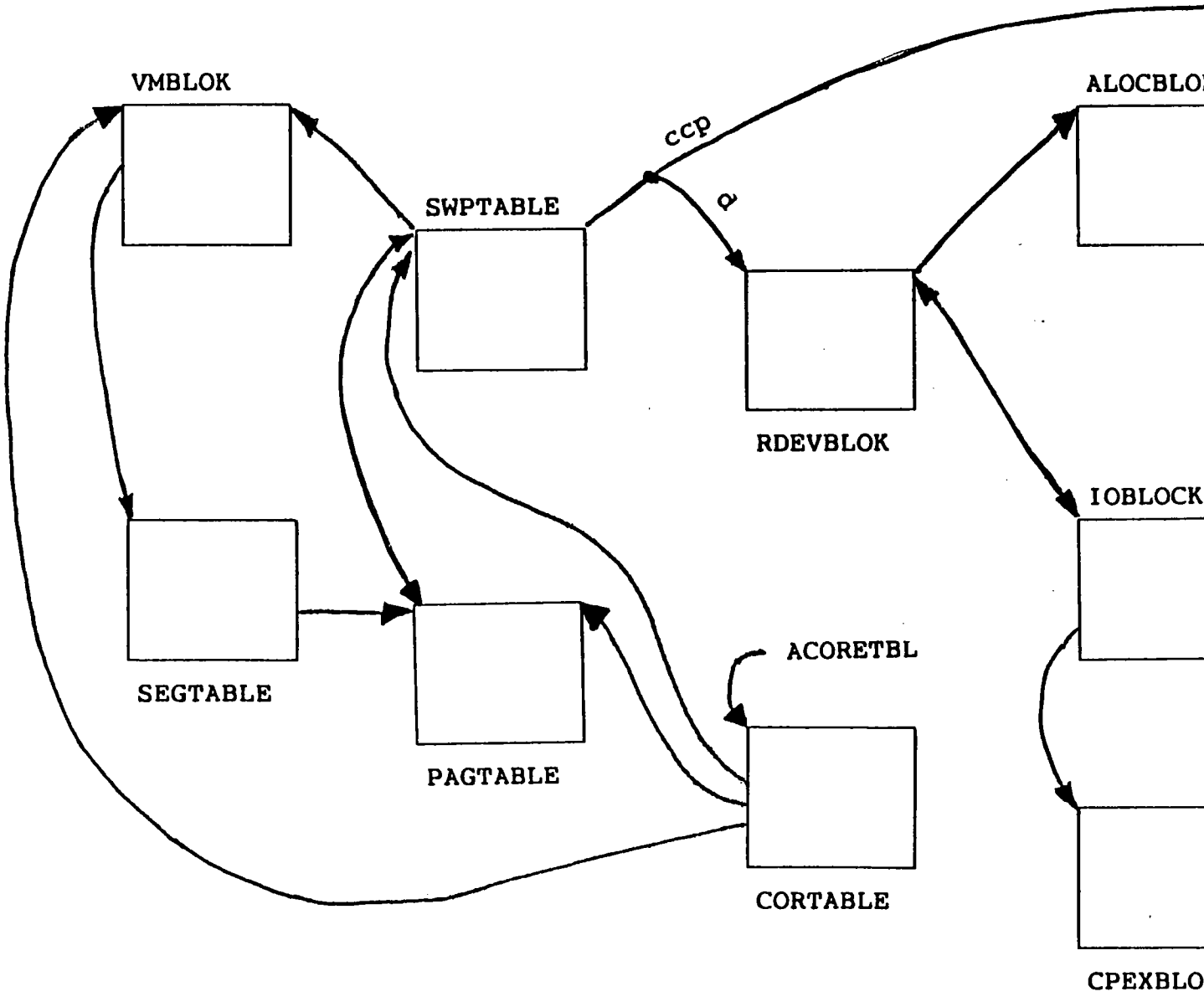
1. SWPTABLE.
2. CPEXBLOK.
3. IOBLOK with extensions.

The control blocks involved in DASD slot administration are:

1. ALOCBLOK and friends.
2. RECBLOK and extensions.



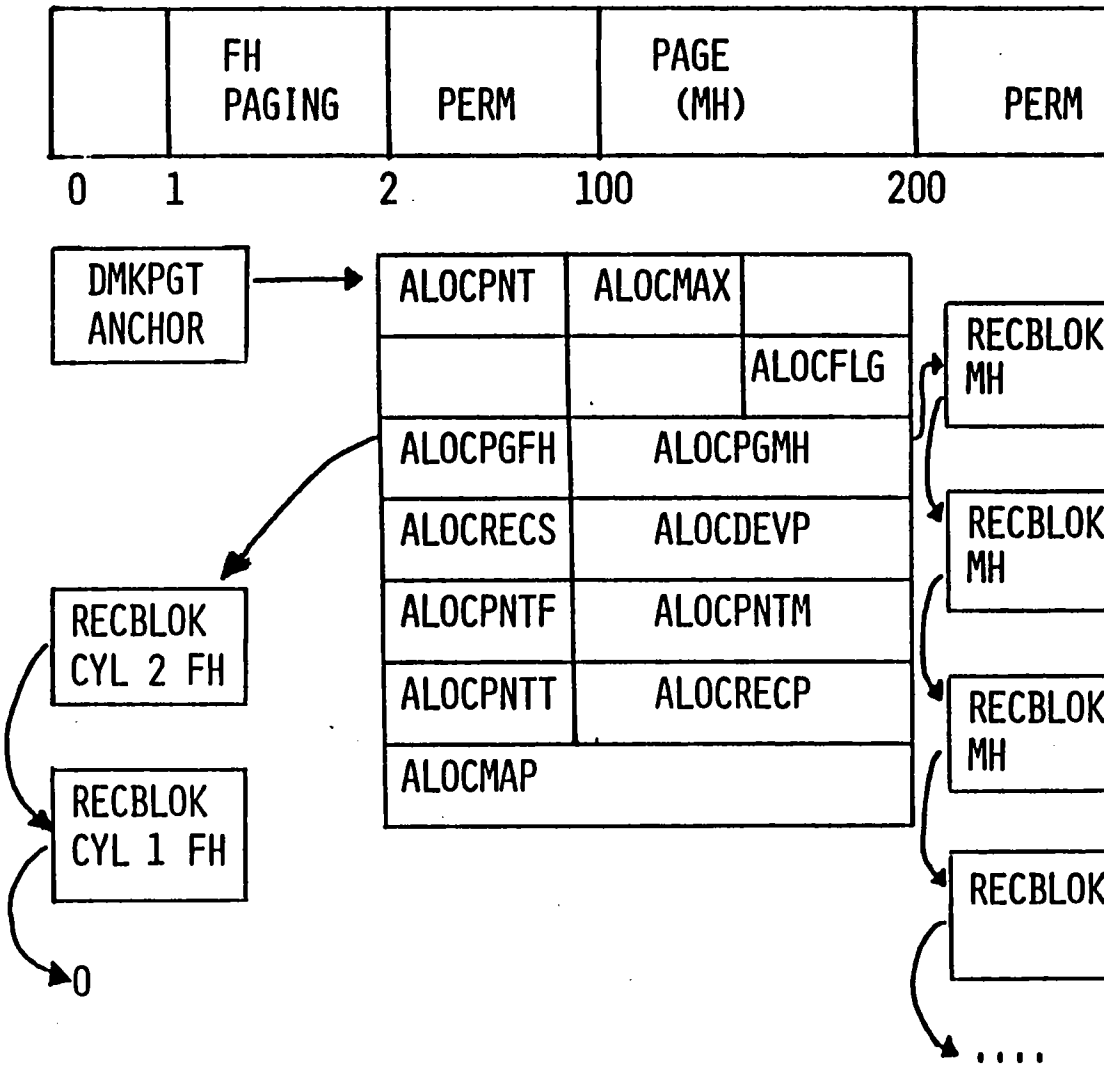
DYNAMIC ADDRESS TRANSLATION



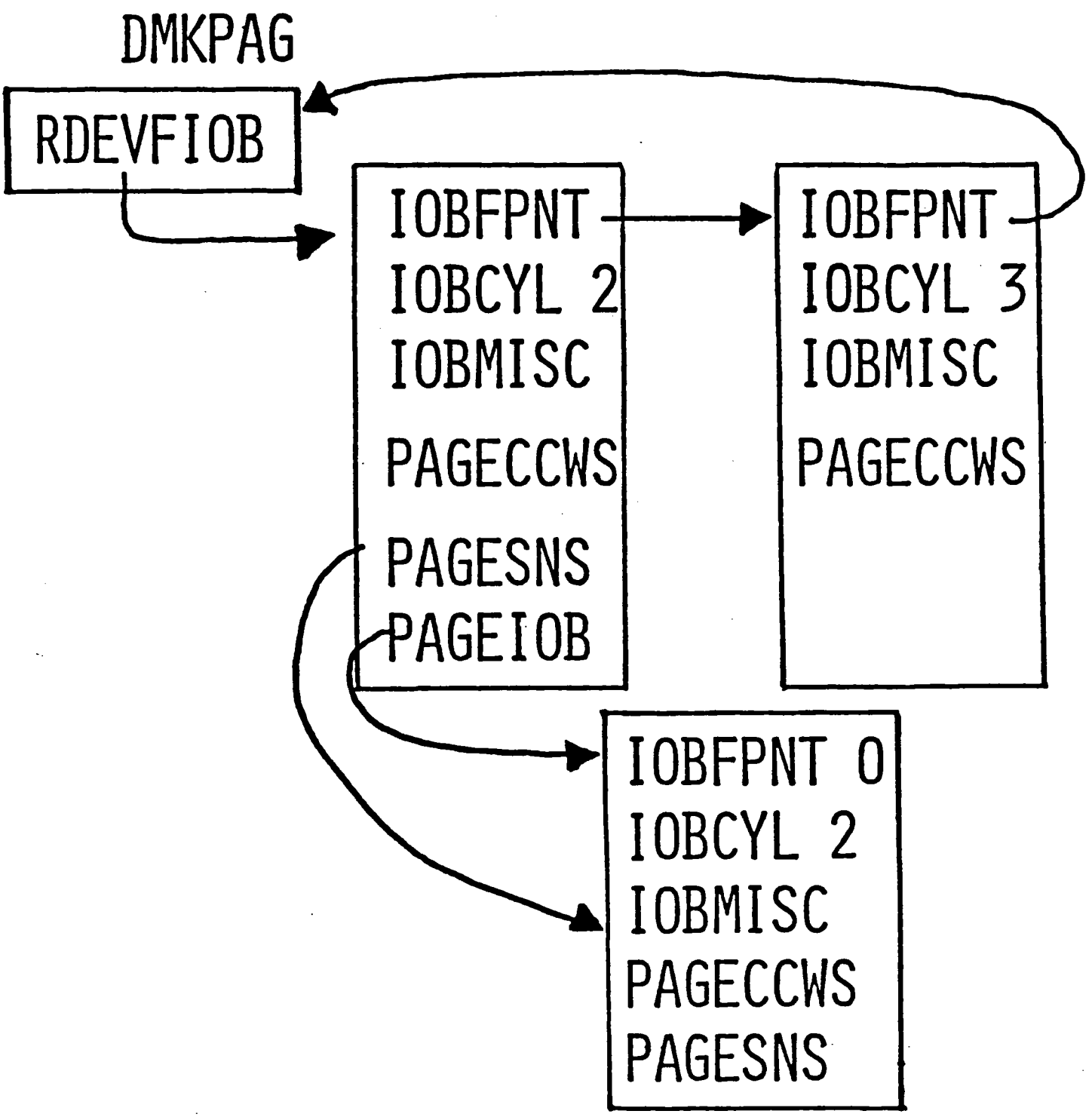
Paging Overview

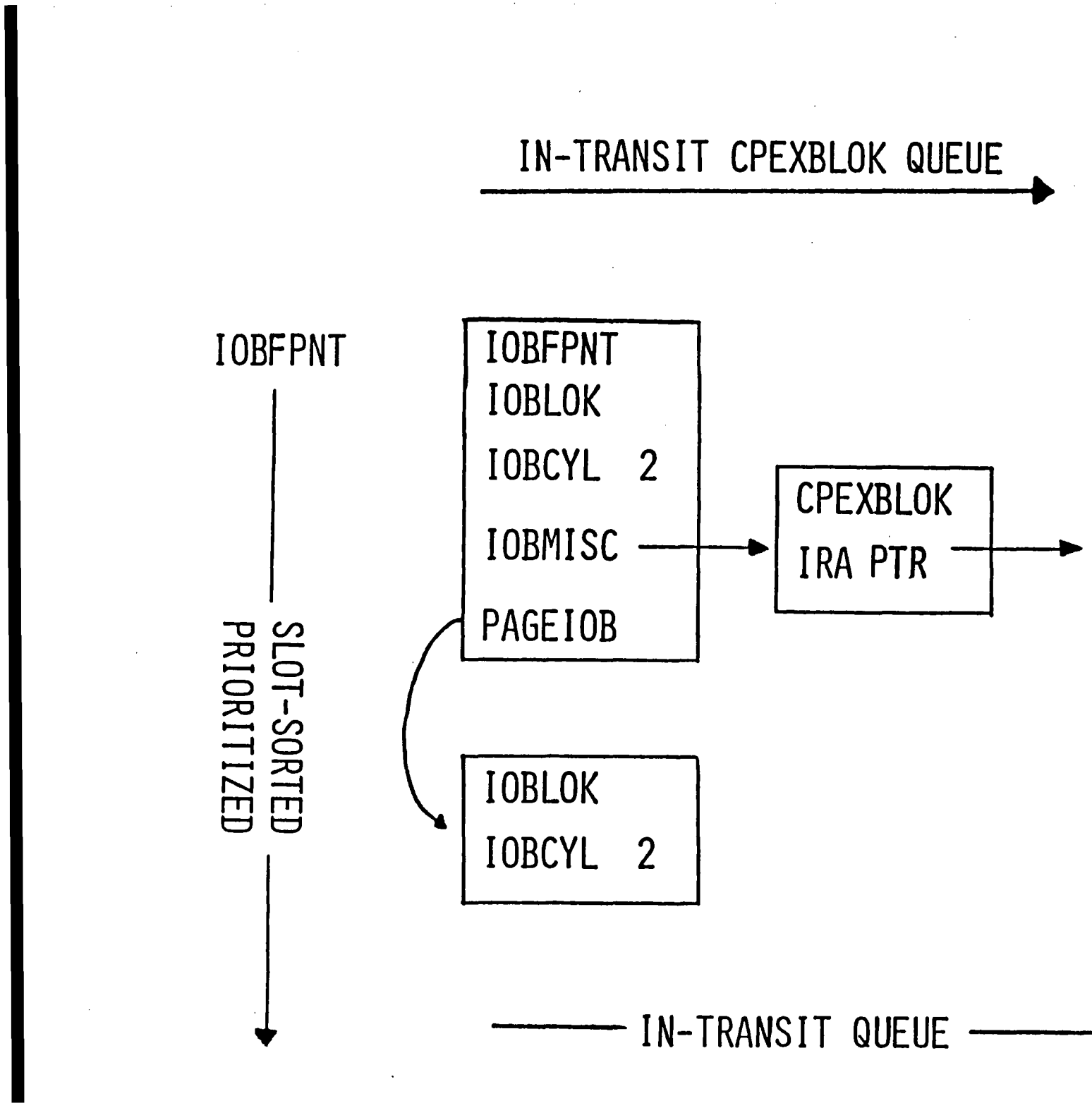
PAGE ALLOCATION CONTROL BLOCKS

COMPLICATED CASE



IOBLOK CHAINING 3




IN-TRANSIT CPEXBLOK QUEUE
IOBFPNT
SLOT-SORTED
PRIORITIZED
IOBFPNT
IOBLOK
IOBCYL 2
IOBMISC
PAGEIOB
CPEXBLOK
IRA PTR
IOBLOK
IOBCYL 2
IN-TRANSIT QUEUE

NOTES

readers, and printers. The channel-to-channel adapter (CTCA) can be used to connect two channels, allowing one to read and the other to write as a means of transmitting data between two computer systems. A device can perform one operation at a time; some operations involve data transfer and others involve only physical motion.

1.4.3 I/O addressing

Just as memory locations have addresses, so do components in the I/O subsystem. In general, I/O addresses are 3 hexadecimal digits or nibbles (12 bits):

1. The first nibble is the channel number, which ranges from 0 to the installed maximum, since some CPUs have less than the full set of 16 channels.

2. The second nibble is the controller number. Most channels allow at most 8 controllers to be attached (for various physical reasons), but their addresses usually may be chosen at will.

3. The third and last nibble is the device number.

The address scheme is slightly modified in a few cases, so that the control unit number may actually occupy anywhere from 0 to 5 bits:

1. Some controllers, such as the 3705, support a great many devices. They therefore respond to several controller addresses, even though the device is actually only one controller. The same is true of the 3274 display controller.

2. In the case of disk devices, an additional subdivision is introduced, the "string". A string is a group of 8 disks with device address 0 to 7 or 8 to F. One or more strings can be attached to the controller. In such a case, the upper 3 bits of the controller address nibble are used to address the controller and the remaining bit plus the upper bit of the device address combine to address the string. Some older disk devices, however, allowed only 1 string per controller, so that 5 bits were used as the controller address. This variation is of little interest to programmers but must be considered by anyone who sets up the hardware configuration.

Chapter 10

PAGE MIGRATION

10.1 INTRODUCTION

10.1.1 Overview

What happens to performance when the number of active pages in the system exceeds the capacity of the preferred paging area? Specifically, what happens when the fixed head areas overflow? The process that deals with this problem is called "page migration".

As long as the fastest devices have page slots available, CP automatically moves changed pages to the highest performance device type. This phenomenon is a side effect of the way DMKPTR and DMKPGT function. Before page-out of changed pages, DMKPTR calls DMKPGT to allocate a new DASD CCPD. Since DMKPGT always allocates pages on the fastest devices first, active pages will tend to move to these devices. This beneficial effect disappears when the fixed head page areas become full. Introduced as a part of the Wheeler PRPQ, and now incorporated into VM/SP, page migration deals with the problem of a full fixed head area by trying to identify inactive pages and moving them to non-preferred space, i.e. general paging and spool space. The identification of activity is done on a segment basis. Each page table, mapping one segment of virtual memory, is time-stamped whenever any page in the segment is processed by DMKPTR. Periodically CP scans all VMBLOKs in the system for unreferenced segments and moves those pages from preferred page space to temp space. In addition, during times of pressure on free storage, CP also moves the page tables and SWPTABLES of the migrated segments from memory onto DASD page slots. This process is called SWPTABLE migration. With the decreasing price of main storage and the corresponding increasing popularity of large real memory systems, the continued value of SWPTABLE migration may be doubtful. However, the continued need for page migration is assured as long as there are virtual memory operating systems and a lack of high performance, large capacity DASD devices.

10.1.2 References

10.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

3. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

10.1.2.2 CP modules

1. DMKPGM - controls page migration.

2. DMKSTR - controls SWPTABLE migration.

10.2 PAGE MIGRATION - DMKPGM

Remember the strange word of flags at DMKPTRXX introduced in the chapter on real storage management? Several flags of this word are used by page migration and SWPTABLE migration. Refer to table 19 for a complete definition of the flags.

TABLE 19

DMKPTRXX flags

Byte	*Flag*	*Definition*
0	80	Signals heavy paging.
	40	Signals another measure of high paging.
1	80	Indicates that the FH space is full.
	40	Indicates the MH space is full.
	02	Marks recalibration of the paging stats.
	01	Activates page migration.
2	80	Activates SWPTABLE migration.

All other flags are currently undefined.

Page migration, controlled by DMKPGM (PaGe Migration), is invoked either by the MIGRATE command (entry point DMKPGMUS) or by a CPEXBLOK stacked by DMKSTP (entry point DMKPGMEP). The MIGRATE command can be issued for a specific VMBLOK or for all VMBLOKs. If a userid is specified, only that VMBLOK's pages will be considered for migration. If no parameter is given, then all users' pages are considered. When DMKPGMEP is called then all users are candidates for migration. We will discuss the system-wide case. The user-specified case is a subset of the logic needed for general page migration.

10.2.1 Scanning order

As with several other processes in CP, page migration runs in a four-level nested loop. The outermost loop is controlled by a counter in SAVEWRK1+1. Page migration will loop normally three times on the VMBLOK scan trying to find sufficient inactive pages to move. "Sufficient pages" is defined to mean one-eighth of the total of fixed head pages in the system. "Sufficient moveable head pages" means one-eighth of the total of moveable head pages in the system. For the first VMBLOK scan, the time that a segment needs to be unreferenced is 12 minutes. On the second pass, the time is halved. The third pass uses an unreferenced interval of one-quarter the first interval. If the fourth pass is needed, the interval is zero minutes.

(The fourth pass through page migration is a last ditch effort to clear some preferred space. First the reference interval for a segment is set to zero. If there are no fixed head pages in the system, the moveable head preferred pages of all VMBLOKs are searched. Since this last action is so drastic, it is executed, at most, every 30 minutes.)

The second loop scans the linked list of VMBLOKs. To stop the search, page migration makes use of the fact that the system's own VMBLOK will be the first on the chain. During the scan of VMBLOKs, if the 12.5 percent vacant fixed head pages threshold is met, migration terminates. If the moveable head page areas were also full, 12.5 percent of these pages must be moved.

The third loop scans a user's segments. Scanning a segment means looking at the segment table entry and the page table header. For each VMBLOK the fourth and innermost loop looks at the individual page table entries (PTEs).

10.2.2 Selection criteria

Let us look at the criteria DMKPGM uses to determine whether to move a page. First the VMBLOK must meet the following conditions:

1. Migration is not inhibited (VMINHMIG).

2. User is not in the process of logging on or off (VMLOGON + VMLOGOFF).

3. The user is not in queue (VMINQ).

For a segment's pages to be considered for migration, all of the following must be true.

1. SEGINV flag must be on.

2. PAGSTMP + interval must be less than the current time.

3. Count of active segment table entries must be zero (PAGACT = 0). That is, the segment must not be actively shared.

For shared pages, the time criterion is ignored. Migration invoked by command ignores shared pages.

For each page in a segment table, the following conditions must hold in order for the segment to be migrated.

1. SWPRECMP not on.

2. SWPTRANS + SWPALLOC not on.

3. PAGINVAL flag in PTE on.

4. Real storage frame number in the PTE is 0.

The last 2 conditions are double-checks to be sure that the page is not in use.

If the operator has issued the "SET MHFULL percent" command, and the moveable head areas are full, preferred movable head pages are considered for movement. If either of these two criteria fails, then moveable head pages are not processed.

When a page finally passes all of the selection criteria, it is migrated by the process described below. Remember that the information needed to move the page is contained in the page table entry and its corresponding SWPTABLE.

1. Call DMKPTRFR to obtain a real storage page frame.

2. Build a CPEXBLOK with CPEXADD pointing to the internal label RDRETN. Place the CPEXBLOK onto the read request queue at DMKPTRRQ and go to DMKPAGIO to perform the page-in operation.

3. When RDRETN gets control after the page-in, call DMKPGTPM to allocate a slot on a slower speed paging device. In VM/SP the new slot is taken from temp (spool) space. In HPO systems at release 3.4 and higher, special space can be reserved solely for page migration.

4. Build a CPEXBLOK with CPEXADD pointing to the internal label WRTRETN. Place the CPEXBLOK onto the write request queue at DMKPTRWQ and go to DMKPAGIO to perform the page-out operation.

5. When WRTRETN get control after the page-out, release the old DASD slot by calling DMKPGUPR and update the SWPTABLE entry with the new slot's CCPD. Call DMKPTRFT to release the real storage frame.

This process is repeated for each page in the current segment.

10.2.3 Upward migration

During the VMBLOK scan, if no segments are identified as migration candidates, and if the VMBLOK has more than two pages on moveable head DASD, then each SWPTABLE is scanned for pages to mark so they will be moved upward in the paging hierarchy. The marking of a page to migrate upward is accomplished by turning on the SWPCHG1 flag for each moveable head page.

10.2.4 Unconventional techniques

Several techniques used in DMKPGM are non-standard in CP. First, one pageable module (DMKPGM) invokes a second pageable module (DMKSTR) via a stacked CPEXBLOK. Normally such logic is forbidden, but Lynn Wheeler knew that at DMKSCHTI+8 there is code that loads R15 from R0 and issues an SVC 8. Therefore this location is used as the address in the CPEXADD field. CPEXR0 is loaded with the adcon of DMKSTRSM, which will invoke segment table migration. A second violation of convention occurs when DMKPGM turns off the DMKPTRXX FHFULL and MHFULL flags during the call to DMKPTRRS. DMKPGM

knows that DMKPTRRS executes different logic based on these flags. After returning from DMKPTRRS, the flags are restored to their original settings. The last non-standard logic in DMKPGM to be mentioned is that, just before it exits, it moves 40 bytes of migration statistics into DMKSCHMS (Migration Statistics), so that the scheduler can conveniently have access to them. DMKSTRSM, SWPTABLE migration, moves more statistics to an adjacent area just before it exits.

10.3 SWPTABLE MIGRATION - DMKSTR

Swap table migration is started with a call to DMKSTRSM. DMKSTR is the module that also handles hardware-generated segment exceptions, program interrupt code X'11'. Swap table migration is really a misnomer, since both the page tables and the SWPTABLEs of a segment are moved from storage to DASD page space during DMKSTR processing. Lynn Wheeler, when designing SWPTABLE migration, created a mechanism that had broader uses than just migration. Normally a virtual machine has one "real memory", the segment table origin (STO) of which is contained in field VMSEG in the VMBLOK. Whenever the user is dispatched, the real control register 1 must be loaded with the STO address. For SWPTABLE migration, there is a field in the VMBLOK called VMSWPMIG. This field serves as a page table origin for a second one-segment "real memory" for the user. There is a fixed mapping of the virtual memory address represented by a particular SWPTABLE into a 256-byte slot in this memory space.

DMKSTRSM moves page tables and SWPTABLES of each migrated segment to a page in the memory space and causes that page to be written out of main storage. The SEGINV and SEGMIG flags are set on for each migrated segment.

When the user attempts to reference a page in a migrated segment, a segment exception is generated by the hardware. Because of the SEGMIG flag, DMKSTR can determine that the segment exception interrupt is due to migration. The page containing the page tables and SWPTABLEs for the segment is brought into memory, and the page tables and SWPTABLEs are moved to free storage and re-connected to the other control blocks. The virtual machine can then re-execute the interrupted instruction.

10.4 CONTROLLING MIGRATION PARAMETERS

The module that generates system-wide load and performance statistics is DMKSTP. It is invoked initially during CP initialization. At the end of processing, DMKSTP stacks a TRQBLOK with a timer set to expire in one minute if the multi-programming level is less than 0.05. If the multiprogramming level is above 0.05, then DMKSTP uses a time of five minutes. DMKSTP is responsible for determining when migration should be invoked. If there have been more calls to DMKPTR for an extended page than the current count of extended pages or if the amount of free storage consumed by page tables and SWPTABLEs is greater than 12 per cent of the total, then the appropriate DMKPTRXX flag is turned on to start SWPTABLE migration. The amount of free storage consumed by page tables and SWPTABLEs is approximated by multiplying the number of segments active in the system (TTSEGCNT in DMKPSA) by 23 and then by 8. Free storage is calculated by subtracting the address of DMKFREHI from the real memory size.

DMKSTP builds a CPEXBLOK for page migration (DMKPGMEP) according to the logic shown in table 20.

TABLE 20

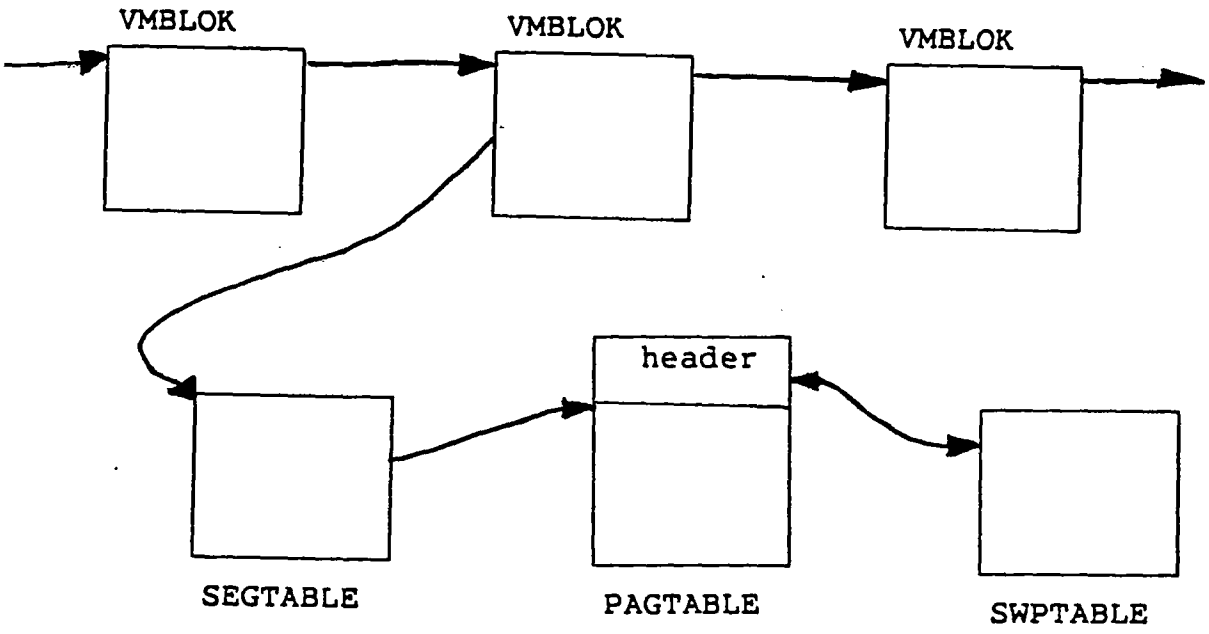
When to invoke page migration

```
|-If preferred paging full
|    or
|-If lots of paging activity (DMKPTRXX+0 = X'C0')
|     and
|      |-If SWPTABLE migration set
|      |   or
|      |-If count of FH pages > 1152 (2305-1).
and
  At least 10 minutes since last migration call.
```

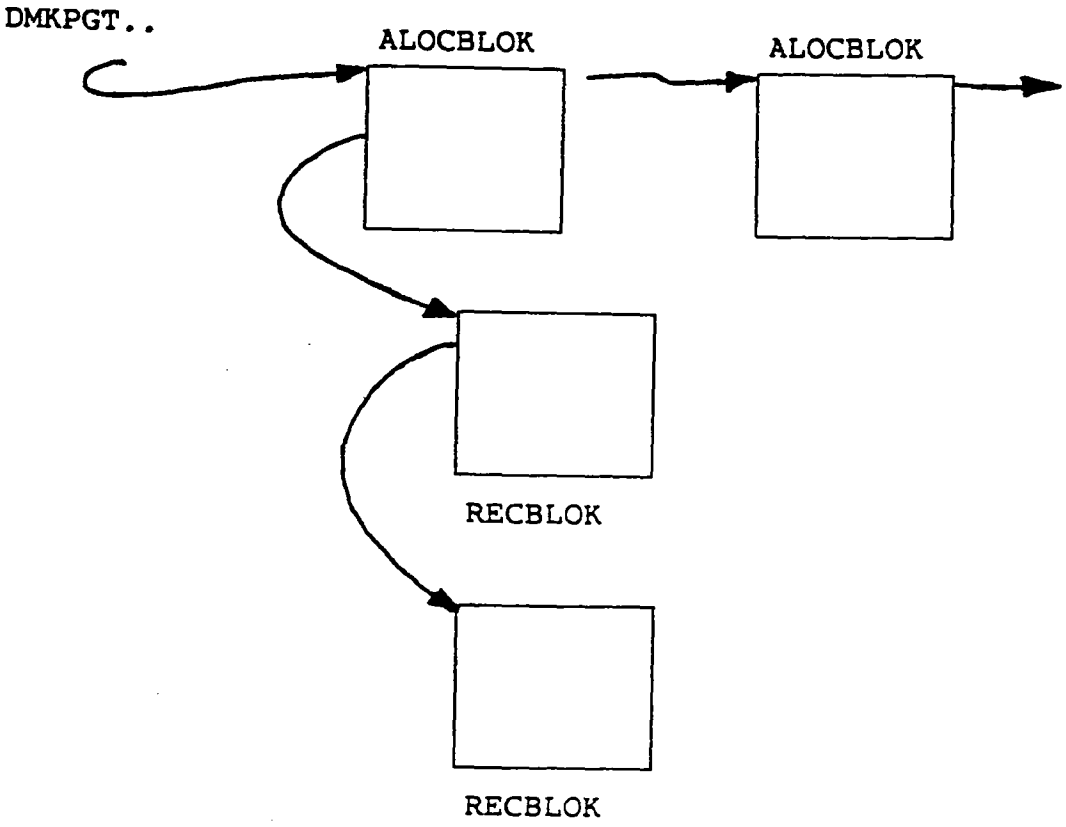
10.5 SUMMARY AND CRITIQUE

Page migration in CP performs a very important function. To maintain high performance, a system with a total page set greater than the capacity of the fastest paging devices needs a mechanism to move inactive pages to slower devices. DMKPGM and DMKSTM implement algorithms to accomplish this purpose. The code in DMKPGM and DMKSTM is non-standard and difficult to understand. The algorithms suffer from their primitive nature. The use of the segment time stamp is especially poor. A finer method of detecting inactive pages, not segments, is needed. Several installations have modified the timing constants associated with page migration to force it to use a shorter reference interval. The standard is 12 minutes for the first pass, 6 for the second pass and 3 for the third pass. Installations can use the "SET SRM PGMTLIM" to lower the initial interval. A second constant is the threshold of one-eighth the total FH pages in the system. When one-eighth the total are moved from fixed head areas, migration stops. This number is too high for heavily loaded paging subsystems. Many installations have lowered the number to one-sixteenth or one-thirtysecond. A third critical time is the interval between successive invocations of migration. The ten minute standard has been lowered by many installations to five or, even, one minute. In their HPO product, IBM has legitimized these revised time and threshold constants by incorporating them into the base system. More work in this area is needed.

IBM



ADDESSE



Page Migration Overview

NOTES

1.4.4 Hardware constructs

In order to manage I/O operations, several hardware constructs are used (these are also shown in figure 6):

1. Channel address word (CAW): Location X'48' in real memory is used to contain the real memory address of the channel program that is about to be started. In effect, this word in memory contains a parameter that is passed to the selected I/O channel.

2. Channel command word (CCW): The series of I/O commands is made up of doublewords that are called "channel command words". Each CCW contains an operation code, a data address pointer, a data length value, and several control flags. The entire series of CCWs is called a "channel program".

3. Channel status word (CSW): Location X'40' in real memory is a doubleword that is called the "channel status word". The CSW is a result value that is stored by the channel when it wants to communicate to the CPU the completion or the failure of a channel program. The CSW contains status flags, a residual length value, and a pointer to the end of the channel program.

4. I/O old and new PSW: Locations X'38' and X'78' in real storage are the old and new I/O PSWs. Whenever a channel wishes to indicate that an I/O operation is complete it causes an I/O interrupt. If the CPU is enabled for that interrupt (PSW bits and CR2), then the current PSW is stored into X'38' and a new PSW is fetched from X'78'. The new PSW should be disabled for I/O interrupts and contains the address of the program that will process the I/O completion.

Chapter 11

I/O PROCESSING

11.1 INTRODUCTION

11.1.1 Overview

CP I/O processing can be divided into three major areas:

1. The basic facilities used by all I/O operations.
2. The virtualization of I/O requests issued by a virtual machine.
3. CP's own I/O requests performed to real I/O devices.

This chapter will concentrate on the 'normal' I/O processing logic; error handling will be covered in less detail.

11.1.2 Review of 370 I/O processing

As you will recall from the introductory chapter, System/370 I/O is initiated by CPU instructions and carried out by the I/O subsystem consisting of the devices, control units, and channels. Each I/O device has one or more I/O addresses, and the address includes the address of the associated controller and channel. In general terms, typical I/O processing consists of the following operations:

1. An application program (the 'user' of some I/O device) constructs a channel program to perform some function on the device.
2. The application program typically calls some system program (an I/O supervisor, or 'IOS') that waits if necessary until the device is available for use.
3. The IOS sets the CAW to point to the channel program.
4. The IOS issues an SIO instruction to start the channel program running in the channel.

5. The IOS passes control back to the application program, which typically issues some 'wait' function.

6. At some later time an interrupt from the channel causes the current PSW to be stored and a new PSW to be fetched.

7. The new PSW causes the I/O interrupt routine in the IOS to run, and that routine causes the application program to be re-activated to continue processing.

From the point of view of a virtual machine, exactly that set of operations must appear to take place. From the point of view of CP, exactly that set of operations must actually take place whenever a real I/O device is used.

11.1.3 References

11.1.3.1 Publications

1. *Principles of Operation* (GA22-7000) contains a description of the general I/O architecture.

2. *Special Feature Description: Channel to Channel Adapter* (GA22-6983) describes the CTCA in detail.

3. For the various devices, see the appropriate "Component Description" manuals.

11.1.3.2 CP modules

1. DMKCCW - translates channel programs from virtual to real.

2. DMKCNS - supports real line mode terminal devices.

3. DMKDAD - performs 3380 error recovery.

4. DMKDAS - performs CKD error recovery.

5. DMKDAU - performs FBA error recovery.

6. DMKDGD - performs synchronous I/O for DIAGNOSE X'18'.

7. DMKDGF - handles interrupts for DIAGNOSE X'18'.

8. DMKGIO - performs synchronous I/O for DIAGNOSE X'20'.

9. DMKGRF - supports real local 3270 terminal devices.

10. DMKIOQ - finds paths and manages I/O request queues.

11. DMKIOS - starts all real I/O operations.

12. DMKIOT - handles all real I/O interrupts.

13. DMKISM - supports virtual ISAM channel programs.

14. DMKRGA - supports real BSC 3270 devices.

15. DMKRGB - supports real BSC 3270 devices.

16. DMKRGC - supports real BSC 3270 devices.

17. DMKRGD - supports real BSC 3270 devices.

18. DMKRSG - supports special 3800 functions.

19. DMKSCN - locates real and virtual I/O control blocks.

20. DMKTAP - performs real tape error recovery.

21. DMKUNT - translates channel programs from real to virtual.

22. DMKVCA - supports the virtual channel-to-channel adapter.

23. DMKVIO - simulates virtual I/O interrupts.

24. DMKVSC - performs CCW translation checks for V=R.

25. DMKVSI - processes a virtual I/O instruction.

26. DMKVSJ - processes a virtual I/O instruction.

11.2 BASIC CP I/O FACILITIES

There are three major components to CP's I/O processing routines:

1. DMKIOS, the I/O supervisor (some 3300 lines of code).

2. DMKIOT, the I/O interrupt handler (some 1500 lines).

3. Various control blocks, especially the IOBLOK and the RxxxBLOK and VxxxBLOK groups.

11.2.1 Control blocks

Several control blocks are used by CP I/O processing; the most important of these blocks are described below:

1. The IOBLOK is the basic unit of work for an I/O request; any routine in CP wishing to perform I/O must obtain an IOBLOK from free storage, construct in the IOBLOK various pointer and flag values, and then call the I/O supervisor with the IOBLOK address as a parameter. The IOBLOK contains such information as the address of the associated I/O device, the address of the VMBLOK for whom the I/O is requested, and the address of the channel program.

2. The RDEVBLOK is a portion of DMKRIO and represents a real I/O device in the system configuration. The RDEVBLOK contains flags indicating the current status of the device; it also contains pointers to the control units by which the device can be accessed. Additional pointers indicate which IOBLOK is currently active on the device.

3. The RCUBLOK is contained in DMKRIO and represents a real I/O control unit in the system configuration. The RCUBLOK contains flags that indicate the status of the control unit; it also contains pointers to the attached devices and to the channels by which the control unit can be accessed.

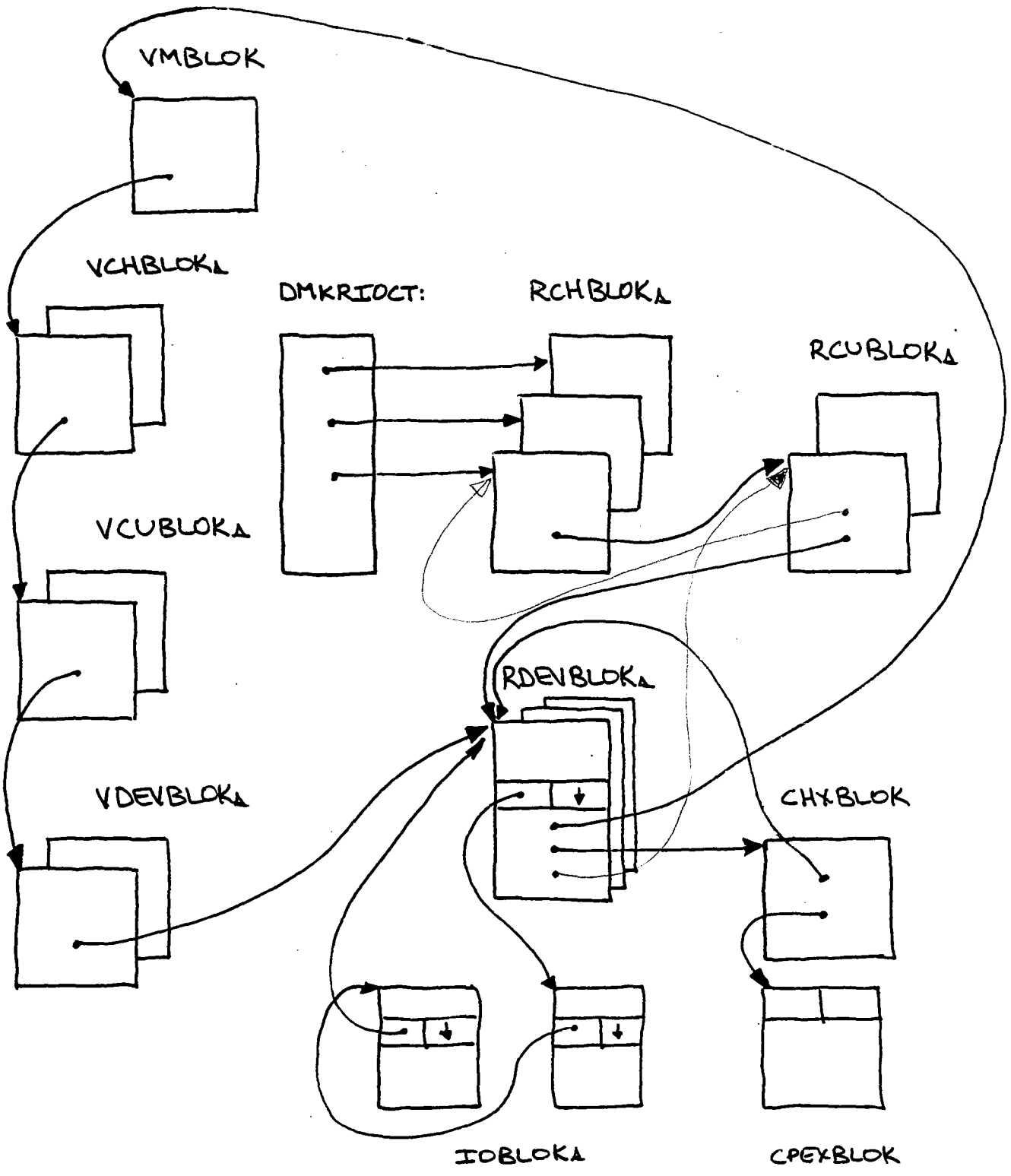
4. The RCHBLOK is contained in DMKRIO and represents a real I/O channel in the system configuration. The RCHBLOK contains flags that indicate the status of the channel; it also contains pointers to the attached control units.

5. The VDEVBLOK is constructed in free storage and represents an I/O device in a virtual machine's configuration. The VDEVBLOK is very similar to the RDEVBLOK and contains similar flags and pointers.

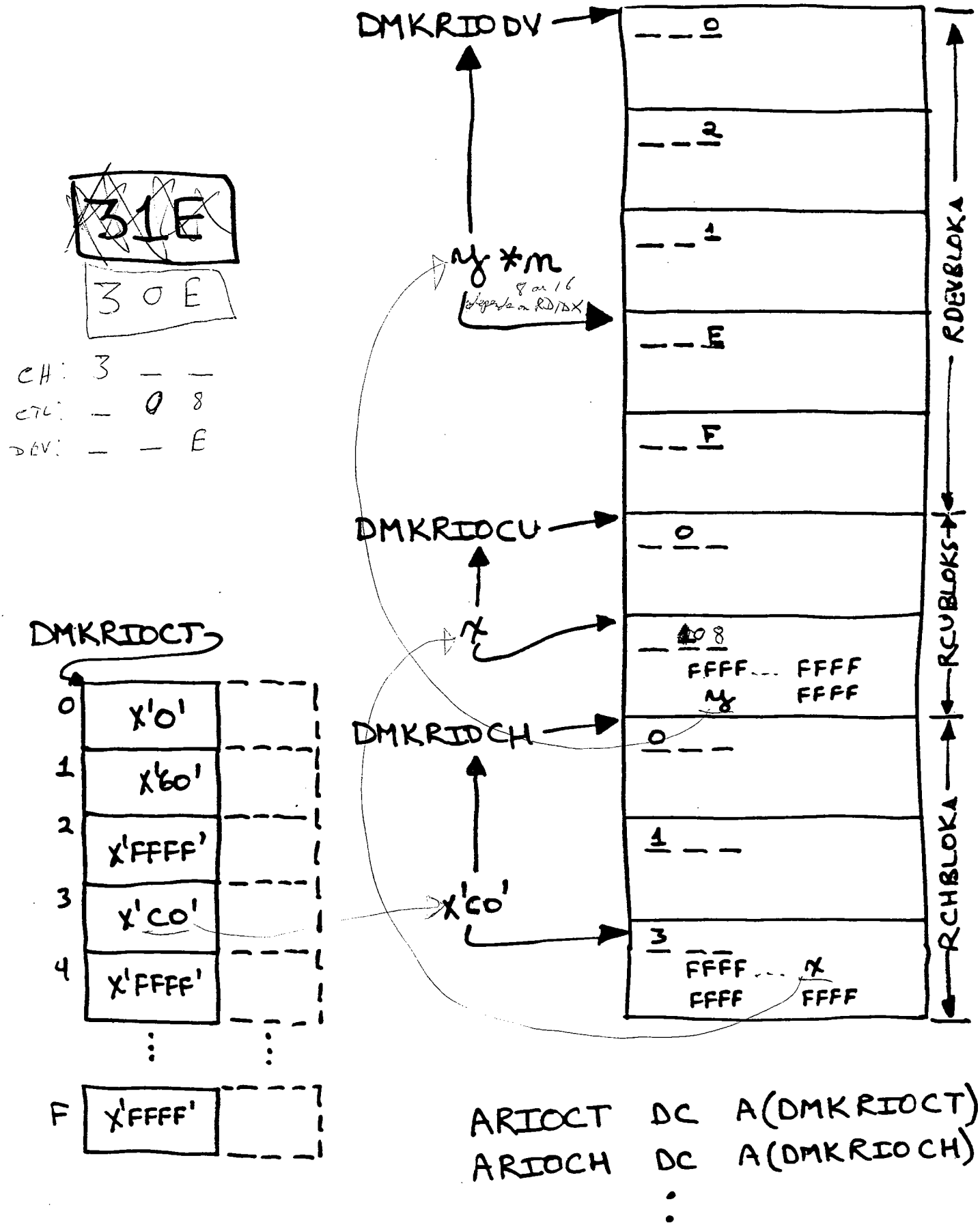
6. The VCUBLOK is constructed in free storage and represents an I/O control unit in a virtual machine's configuration. The VCUBLOK is very similar to the RCUBLOK and contains similar flags and pointers.

7. The VCHBLOK is constructed in free storage and represents an I/O channel in a virtual machine's configuration. The VCHBLOK is very similar to the RCHBLOK and contains similar flags and pointers.

To find the real I/O blocks for a given device, you can use the routine DMKSCNRU (scan for a real unit). That routine's logic is as follows:



I/O Processing Overview


DMKRIODV
0
2
1
E
F
RDEVBLOKS
DMKRIOCU
0
RCUBLOKS
8
FFFF ... FFFF
FFFF
DMKRIOCH
0
1
RCHBLOKS
3
FFFF ... FFFF
FFFF
8 or 16
DMKRIOCT
0 X'0'
1 X'60'
2 X'FFFF'
3 X'C0'
4 X'FFFF'
F X'FFFF'
X'C0'
31E
30E
CH: 3 _ _
CTL: _ 0 8
DEV: _ _ E
ARIOCT DC A(DMKRIOCT)
ARIOCH DC A(DMKRIOCH)

1. Multiply the 4-bit channel number by 2 and use the result to index into the channel lookup table at DMKRIOCT. Each halfword in that table contains the displacement of an RCHBLOK from the first RCHBLOK. Add the halfword value to the address of DMKRIOCH. The resulting RCHBLOK address is placed into R6.

2. Multiply the 5-bit control unit number by 2 and use the result to index into the RCHBLOK's table of control units, starting at RCHCUTBL. Each halfword after RCHCUTBL is the displacement of an RCUBLOK from the first RCUBLOK. Add the halfword value to the address of DMKRIOCU. If this is a subordinate RCUBLOK, then get the pointer to the primary RCUBLOK. The resulting RCUBLOK address is returned in R7.

3. Multiply the 3-bit device number by 2 and use the result to index into the RCUBLOK's table of devices, starting at RCUDVTBL. Each halfword after RCUDVTBL is the displacement of an RDEVBLOK from the first RDEVBLOK, except that the displacement is in terms of doublewords or quadwords. Multiply the displacement by 8 or 16, according to the bit RDIDX in CPSTAT5, and add the result to the address of DMKRIODV. The resulting RDEVBLOK address is returned in R8.

CP arbitrarily generates an RCUBLOK for each group of 8 RDEVBLOKs, no matter how the real devices and control units are configured; this structure simplifies much of the work in the I/O supervisory routines. Since some RCUBLOK status information must apply to the entire corresponding control unit address range, the first RCUBLOK of such a range is considered the "primary" RCUBLOK; all other RCUBLOKs in the range contain a pointer to it. As shown above, DMKSCNRU always returns the address of the primary RCUBLOK.

11.2.2 DMKIOS (and DMKIOQ)

DMKIOS is the portion of the I/O supervisor that queues and starts I/O requests. Its two main entry points are DMKIOSQV for virtual machine I/O and DMKIOSQR for CP I/O; both entry points result in the same basic logic being run, the main difference being that virtual machine I/O processing requires changing the status of the VMBLOK. At service level 305, DMKIOS was split, with certain subroutines moving to a new module, DMKIOQ. The descriptions below will continue to use the old subroutine names, with the new ones given in parentheses. The main logic flow through DMKIOS is as follows:

1. If the real device is already busy (RDEVSTA4: RDEVBUZY), then queue the IOBLOK onto the RDEVBLOK and return to the original caller.

2. Otherwise, call the internal subroutine IOSTRTDV:

   a) Mark the device 'scheduled' (RDEVSTA4: RDEVSCHD).

   b) Call the internal routine IOSFINDP (DMKIOQFP) to get a path to the device, consisting of an available channel and an available control unit. If no path is operational, then return to the caller of IOSTRTDV to simulate CC=3. If all paths are busy, then queue the IOBLOK onto the RCHBLOK or RCUBLOK and exit to the original caller.

   c) Place the IOBLOK address into RDEVAIOB and set RDEVBUZY and clear RDEVSCHD in RDEVSTA4.

   d) Set RCHBUSY for a selector channel and set RCUBUSY for a shared control unit.

   e) Set the real CAW from IOBCAW and clear the real CSW and RDEVCSW.

   f) For CP I/O or for virtual machine DIAGNOSE or SIOF, issue a real SIOF. For virtual machine SIO, issue a real SIO.

   g) If the result is CC=3 (hardware not operational), then:

      i) Unbusy all the RxxxBLOKs.

      ii) If there is no alternate path, then return with IOBCC3 and IOBFATAL error indications.

      iii) If there is an alternate, then try again on another path (point b above).

   h) If the result is CC=2 (channel busy), then:

      i) Queue the request on the RCHBLOK.

      ii) Return to the caller if this is a byte multiplexor operation.

      iii) Find an alternate path (point b) for a selector or block multiplexor.

   i) If the result is CC=1 (CSW stored), then:

      i) Unbusy the RCHSTAT and RCUSTAT.

ii) Stack the IOBLOK (call DMKSTKIO).

iii) If there was a unit check in the CSW, then schedule a SENSE IOBLOK. Otherwise, start up any queued I/O for this device or controller or channel.

j) If the result is CC=0 (success), then:

i) Set RDEVSTA4: RDEVBZCH to show that the device is busy on the channel.

ii) If this is a virtual machine SIO request, then clear VMIOWAIT.

iii) If this is not for the active running virtual machine, then call DMKSCHDL to inform the scheduler of the virtual machine's change of state.

iv) If this is a byte multiplexor, or a selector, or a block multiplexor with no queued requests, then return to the caller. Otherwise, issue a TCH instruction to cause a "channel available" interrupt as soon as I/O interrupts are re-enabled by the dispatcher. The interrupt will cause DMKIOT and DMKIOS to re-issue the queued request, which will then have a very good chance of receiving CC=0 instead of CC=2. This saves some real time by allowing the block multiplexor channel to be given as many requests as possible.

3. Return to the original caller.

The internal subroutine IOSFINDP (DMKIOQFP) mentioned above has the following logic flow:

1. For the case that there is only one path defined for the device, and if both the channel and the controller are available (operational and not busy), then return to use the path. If either the channel or the controller is not operational, then return for CC=3 processing. Otherwise, one of them is busy, so queue the IOBLOK onto the busy component and exit to the original caller of IOSTRTDV.

2. For the case that there are alternate paths to the device, and if no path is operational, then return for CC=3 processing. If an available path can be found, then return to use that path. Otherwise, queue the IOBLOK onto the first busy path component

and create associated mini-IOBLOKs and queue them onto all other operational paths; return to the original caller of IOSTRTDV.

The major subroutines IOSQDEV, IOSQCU, and IOSQCH (DMKIOQQD, DMKIOQUS, and DMKIOQSK) all use common logic to queue an IOBLOK onto an RxxxBLOK:

1. For non-DASD, add the IOBLOK to the FIFO chain anchored at RxxxFIOB and update the field RxxxLIOB.

2. For DASD, also add the IOBLOK to the chain anchored at RxxxFIOB, but insert fixed head requests FIFO before all moveable head requests and insert moveable head requests in order by increasing seek addresses.

11.2.3 DMKIOT

DMKIOT, the I/O interrupt portion of the I/O supervisor, gains control directly from the I/O new PSW whenever an I/O interrupt occurs. Although various unusual and error conditions make DMKIOT rather complex, its logic is quite simple for the common case of channel end plus device end:

1. Call DMKSCNRU to the RxxxBLOKs associated with the interrupting device.

2. Save the CSW into RDEVCSW and into IOBCSW.

3. In turn clear RCHBUSY, RCUBUSY, RDEVBZCH, and RDEVBUZY.

4. Call DMKSTKIO to stack the completed IOBLOK so that execution can resume at the address contained in IOBIRA.

5. Go to DMKIOSRS to restart any queued I/O requests using this path:

   a) If there is an IOBLOK queued on the RDEVBLOK, then start its I/O operation and GOTO to DMKDSPCH.

   b) If there is an IOBLOK queued on the RCUBLOK, then start its I/O operation and GOTO to DMKDSPCH.

   c) If there is an IOBLOK queued on the RCHBLOK, then start its I/O operation and GOTO to DMKDSPCH.

   d) If nothing is queued on this path, then GOTO to DMKDSPCH.

6. The major subroutines IOSDQDV, IOSDQCU, and IOSDQCH (DMKIOQDE, DMKIOQDU, and DMKIOQDH) all use common logic to remove the next IOBLOK from the RxxxBLOK:

   a) For non-DASD, remove the next item from the chain anchored at RxxxFIOB.

   b) For DASD, remove the next IOBLOK if it is a fixed head request. If it is a moveable head request, then remove the IOBLOK whose seek address is next. Use the RDEVCYL value for the current seek address and the RDEVSKUP flag to indicate in which direction the disk arm is currently being moved.

11.3 SUPPORT OF VIRTUAL MACHINE I/O REQUESTS

Each I/O request from a virtual machine must be handled differently, depending upon the exact type of virtual or real I/O device being used. The following is a description of the processing that CP must perform for several types of devices.

11.3.1 Device independent support

The initial handling of virtual machine I/O is common to all device types and involves these processes:

1. initial checks
2. channel program translation
3. conversion to real I/O
4. status and interrupt reflection

11.3.1.1 Initial checks

When the virtual machine I/O supervisory routines issue an I/O instruction, such as SIO, a privileged operation program check interrupt occurs on the real machine since the virtual machine is actually running in problem state. The following steps are initiated as a result:

1. DMKPRG gets control from the interrupt and, after noting that the virtual machine is in virtual supervisor state, passes control to DMKPRVLG.

2. DMKPRV notes that the instruction is an I/O instruction (the opcode is in the range X'9C' through X'9F') and passes control to DMKVSIEX.

3. DMKVSIEX begins the actual simulation of the SIO instruction. It first computes the device address and then calls DMKSCNVU to locate the virtual I/O blocks for that virtual device. A failure to find the blocks results in the termination of the SIO simulation by setting the CC to 3 ("device not operational").

4. If the virtual channel block indicates that a previous operation is still active, then the SIO is terminated with CC = 2 ("busy"). If the previous operation's completion status has not yet been returned to the virtual machine, then the SIO is terminated with CC = 1 ("CSW stored"), and DMKVIO is given control to build the new CSW.

5. DMKVSI then locates the virtual page 0, or causes it to be paged in if it is not already resident, and examines the virtual CAW (location X'48').

6. For certain virtual devices, DMKVSI passes control at this point to special processing routines that will be described later: DMKVCN for virtual consoles and DMKVSP for spooled unit record devices.

7. For all other devices, DMKVSI builds an IOBLOK with the interrupt address pointing to DMKVIOIN.

11.3.1.2 Channel program translation

Virtual machine channel programs are subjected to a virtualization process that can affect all three major components of the channel program:

1. The CCW opcodes are in some cases modified since certain sequences cannot be supported for virtual machines.

2. The address portions of the CCWs must be converted to real memory addresses within the real machine since the I/O channels require real memory addresses. The addressed pages must also be paged into real memory and locked there for the duration of the I/O operation.

3. The data portions must be modified in certain cases.

DMKVSI calls DMKCCWTR to perform the CCW translation (for the V=R virtual machine, DMKVSI first calls DMKVSCVR to see if the translation is needed for this channel program, and if not then the call to DMKCCWTR is skipped). The result is a new channel program that contains real memory addresses and the appropriate opcodes. Certain special simulation for ISAM channel programs is also performed before control is returned from DMKCCWTR.

11.3.1.3 Conversion to real I/O

DMKVSI continues to process the virtual SIO by performing these steps:

1. Increment VMIOCNT, the count of virtual machine I/O operations. Put the virtual machine into the IOWAIT status (VMRSTAT: VMIOWAIT), unless this is an SIOF instruction and block multiplexing is enabled in virtual CR0 and ECMODE is on and I/O tracing is not active.

2. If this is a virtual CTCA, then call DMKVCAST to finish the SIO.

3. For all other devices, call DMKIOSQV to queue the IOBLOK on the real device.

4. GOTO to DMKVSJ to clear VMEXWAIT and return to the dispatcher via the fast entry point. The virtual machine will now either run or not run, depending upon the settings of the status bits such as VMIOWAIT.

11.3.1.4 Status and interrupt reflection

Virtual machine I/O operations must result in an I/O interruption exactly like on a real system. For that reason, CP must generate I/O interruption codes and cause the virtual machine to perform PSW swaps as necessary to complete the virtualization processes.

1. When the channel program ends, DMKIOT gets control via the real I/O interrupt and stacks the IOBLOK to the dispatcher, which later uses the IOBIRA address to pass control to DMKVIOIN.

2. DMKVIOIN locates the VDEVBLOK, VCUBLOK, and VCHBLOK for the corresponding virtual device and gets the CSW from IOBCSW.

3. The status is copied from IOBCSW to VDEVCSW and the appropriate 'interrupt pending' bits are set in the VxxxBLOKs.

4. DMKUNTRN is called to untranslate the CSW address field.

5. The appropriate 'busy' indicators are cleared in the VxxxBLOKs.

6. DMKUNTFR is called to fret the real CCW chain and to perform any special data conversion such as minidisk relocation for "read home address".

7. The appropriate virtual 'interrupt pending' bits are set in the VCUBLOK and VCHBLOK, as is the summary bit VMPEND: VMIOPND.

8. If VMDSTAT: VMTIO was set, then the virtual machine is taken out of VMEXWAIT.

9. The IOBLOK is FRETted and control is passed to DMKDSPCH.

11.3.2 Device dependent support

Certain types of virtual devices require special handling, which is described briefly below.

11.3.2.1 Virtual DASD

Certain channel program opcodes cannot be allowed for minidisks. For example, DMKCCWTR changes a "write home address' command to a "read home address" with the skip bit since the real home address area must contain the real cylinder and head addresses and not the virtual (minidisk relative) addresses. DMKCCWTR must also modify seek addresses since they must contain the real cylinder or block addresses and not the values relative to the minidisk. Each seek address must also be checked to insure that it does not go beyond the extent of the minidisk. In some cases, input data must also be converted. The count field of record 0 and the home address, for example, are relocated from real cylinder to virtual cylinder by DMKUNTFR.

11.3.2.2 Virtual console

(Virtual console processing in DMKVSPEX is covered in detail in the chapter on terminal support.)

11.3.2.3 Virtual unit record devices

(Virtual unit record device support in DMKVSPEX is described in the chapter on spooling.)

11.3.2.4 Dedicated devices

For dedicated devices all opcodes are allowed. Channel program translation must of course still perform its address conversion and page locking functions.

11.3.2.5 Virtual CTCA

The virtual channel-to-channel adapter is supported by DMKVCA; entry points exists for each of the simulated I/O instructions. The CTCA status is maintained in the CHXBLOK, which contains a mirror image, the CHYBLOK, which describes the "other" of the two connected virtual selector channels. DMKVCA uses a complex arrangement of multiple CPEXBLOKs to perform the appropriate I/O interrupt signalling that the real CTCA performs. For control CCWs, DMKVCA causes attention interrupts to appear on the "other" channel, and for data transfer CCWs DMKVCA itself performs the data movement between the associated virtual machine buffer areas.

11.4 SUPPORT OF CP-GENERATED I/O

In addition to the I/O operations that CP initiates in order to simulate I/O requests from virtual machines, there are also I/O operations that CP initiates on its own behalf.

CP-generated I/O is performed very much like VM-generated I/O; the major difference is that there is no VMBLOK whose status might have to be changed during the I/O processing. The entry point DMKIOSQR is called to execute or queue a CP I/O request, and the flag bit IOBCP indicates that the I/O request originated in CP. The IOBIRA field points to the routine that will process the I/O completion. In some cases, such as within the paging subsystem, additional data areas are placed behind the IOBLOK so that they are pre-

served and are addressable; such techniques are possible because the caller of DMKIOSQR is responsible for obtaining and releasing free storage for the IOBLOK. CP performs its own I/O to several different types of devices.

11.4.1 Real DASD

CP must perform its own I/O to DASD devices such as disks and drums for the following purposes:

1. to save and restore copies of memory pages that have had to be paged out due to memory contention.

2. to maintain the directory of users and their virtual machine configurations.

3. to spool input cards and output lines and cards while driving real card readers, line printers, and card punches.

All of this DASD I/O is performed as paging, which is described in detail in the chapter on paging.

11.4.2 Real unit record devices.

(Real unit record device support is covered in detail in the chapter on spooling.)

11.4.3 Real terminals

(Real terminal support is covered in detail in the chapter on terminal support.)

11.5 MISCELLANEOUS TOPICS

In conclusion, we will discuss several miscellaneous topics:

1. TIO loop handling.

2. DIAGNOSE X'14', X'18', X'20', and X'58'.

3. Special V=R processing.

11.5.1 TIO loop handling

Special processing is performed by CP to handle the case of a virtual machine executing the sequence of a TIO instruction followed by a BC instruction back to the TIO. If normal instruction simulation were to be followed, and if the tested device were busy, then a loop would result and system throughput would suffer. The virtual machine would also be charged for the CPU consumption suffered in the loop and especially in the simulation of the TIO instructions. As a result, special code is present in DMKVIO, DMKVSI, and DMKVSJ to detect the TIO case and modify the simulation process:

1. Whenever DMKVSI is invoked to simulate any I/O instruction, it first clears the VMTIO flag in VMDSTAT. As DMKVSI progresses with its simulation, the VMTIO flag is set under the following conditions: VCHBUSY is on, VDEVBUSY is on, the virtual device is neither a CTCA nor a dedicated terminal line, and the instruction is TIO.

2. When DMKVSI exits via DMKVSJ, if VMTIO is on and the virtual device is a terminal (CLASTERM, implying a start-stop terminal), then VMIDLE is forced on. For all device types when VMTIO is set, control is passed to the dispatcher, which will not run the VMBLOK since its VMEXWAIT is still set from the TIO simulation.

3. When an I/O interrupt occurs and control ultimately passes to DMKVIO, if VMTIO is set then VMEXWAIT is cleared. This will allow the next instruction, perhaps a BC back to the TIO, to be executed. If the device is a terminal (CLASTERM), then VMIDLE is also cleared so that the VMBLOK once again may be dispatchable.

The net effect of this logic is that a TIO to a busy device results in suspension of the virtual machine until the device gives an interrupt. That effectively prevents a subsequent BC instruction from looping, assuming of course that normal programming conventions are followed. (The special tests for CTCA and dedicated terminal lines appear to be included to prevent suspension when various IBM telecommunication access method routines are in use with those devices.)

11.5.2 DIAGNOSE X'14', X'18', x'20', and X'58'

The DIAGNOSE codes X'14', X'18', X'20', and X'58' are all provided as alternative methods for virtual machine I/O under certain special conditions; in many cases CP can do a more efficient simulation of I/O processing when the DIAGNOSE instructions are used since they are almost always synchronous and rarely cause virtual I/O interrupts. DMKHVC gets control as a result of the program check interrupt that occurs when a virtual machine attempts to issue the DIAGNOSE instruction. DMKHVC calls the following routines:

1. For X'14', DMKDRD is called to process a spool file request. For the read request, DMKDRD builds an IOBLOK and calls DMKIOSQV. DMKDRD goes to the dispatcher to wait until the I/O completes.

2. For X'18', DMKDGD is called to process a standardized CKD DASD read or write request. After translating the channel program via DMKCCW, DMKDGD builds an IOBLOK and calls DMKIOSQV. DMKDGD goes to the dispatcher to wait for I/O completion, at which time control is passed to DMKDGF via the unstacked IOBLOK.

3. For X'20', DMKGIO is called to process a general channel program for any device type. The channel program is translated by DMKCCW and an I/O operation is requested via DMKIOSQV. DMKGIO goes to the dispatcher to wait until the I/O completes.

4. For X'58', DMKHVC sets flags indicating a DIAGNOSE and goes to DMKVSI to initiate the standard processing for SIO. In certain cases this DIAGNOSE executes asyncronously, just like SIO.

Note that these actions are all similar to normal virtual SIO handling, except that VMEXWAIT and VMIOWAIT are not cleared until the I/O operation completes on the real device. At that time VMEXWAIT and VMIOWAIT are cleared and the virtual machine is again dispatchable. Except for some special cases associated with the X'58' code, no I/O interrupts are reflected to the virtual machine; the virtual PSW condition code indicates the result of the I/O operation. DIAGNOSE X'58' logic is discussed in greater detail in the chapter on terminal support.

11.5.3 Special V=R processing

If the command SET NOTRANS ON is given, and if the virtual machine is running in the V=R area, then DMKVSI calls DMKVSCVR before calling DMKCCWTR to translate the channel program. DMKVSCVR returns with a code indicating whether or not CCW translation should be performed for the channel program. Translation will be required if any of the following conditions are met:

1. Any CCW attempts to perform I/O to virtual page 0.
2. Any CCW attempts to perform I/O beyond the V=R area.
3. The device is not a dedicated device.
4. SIO tracing is active.
5. The device is a dialed terminal line.
6. There exists an alternate path to the real device.
7. The device is dedicated as read-only.
8. Device status is pending.

This process is non-trivial, but it is far less work than would be performed by DMKCCWTR for normal translation.

NOTES

Chapter 12

TERMINAL SUPPORT

12.1 INTRODUCTION

12.1.1 Overview

The system console is a special device in many ways. It is the one device that is present on every System/370. It is an I/O device, but it also has other functions, such as stopping and starting the CPU. It is the major communications facility between the programming system and the operator.

Support of the virtual machine's system console is much like other virtualization processes performed by CP. From the point of view of the virtual machine, the console appears to be a standard System/370 console device; from the point of view of the user, the console is probably some kind of terminal device. CP performs the appropriate transformations between the virtual and real devices. In this chapter we will discuss the virtual system console as an I/O device and we will examine how CP handles various real terminal devices. The chapter on console functions has a discussion of the virtual system console control facilities.

As a part of VM/SP, CP supports a "logical" 3270 terminal that is actually implemented as a program interface and is used by the VM/Passthru program product. The logical terminal is processed by CP very much as if it were a real device; only at the lowest level is special processing performed.

12.1.2 References

12.1.2.1 Publications

1. *IBM 4341 Process Functional Characteristics and Processor Complex Configurator* (GA24-3672) contains a description of the real consoles: 3278 and 1052.

2. *IBM 3270 Component Description* (GA27-2749) describes the local and remote 3270 terminals.

3. *IBM System/360 Component Description IBM 2702 Transmission Control* (GA22-6846) describes the start-stop terminals.

4. *General Information - Binary Synchronous Communications* (GA27-3004) contains a good introduction to binary synchronous communications.

5. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

6. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

12.1.2.2 CP modules

1. DMKBSC - remote 3270 error recovery.

2. DMKCFM - console commands.

3. DMKCNS - slow-speed terminal I/O.

4. DMKGRA - local 3270.

5. DMKGRC - local 3270.

6. DMKGRF - local 3270.

7. DMKGRH - 3066 (console for 370/168).

8. DMKHPS - logical device (DIAGNOSE and I/O requests).

9. DMKHPT - logical device (external interrupts).

10. DMKQCN - real console write.

11. DMKQCO - real console read.

12. DMKRET - RETRIEVE key for 3270 and 3101.

13. DMKRGA - remote 3270.

14. DMKRGB - remote 3270.

15. DMKRGC - remote 3270.

16. DMKRGD - remote 3270.

17. DMKSND - SEND command processing.

18. DMKTTY - special ASCII and 3101 handling.

19. DMKVSI - virtual SIO handling.

20. DMKVSJ - virtual SIO handling.

12.2 VIRTUAL MACHINE CONSOLE DEVICES

CP supports two types of virtual consoles, the 3215 and the 3270. The user can select which type of console support he wants by using the TERMINAL CONMODE command. Of course, the selected mode must be supported by the programming system that is being run in the user's virtual machine.

12.2.1 3215 mode

By default, the virtual console is a 3215, which is identical to the old System/360 1052 operator's console; the 1052 and the 3215 are typewriter-like devices that support the following CCW opcodes:

1. Write (X'01' and X'09'): These commands write a line to the console. X'01' leaves the typing element at the end of the line, whereas X'09' performs a carriage return at the end of the line.

2. Read (X'0A'): This command unlocks the keyboard and reads the characters that are typed on the keyboard AS THEY ARE TYPED. Since this command must be issued by the channel before the characters can be typed, the operator must push a "REQUEST" key, which causes an ATTENTION I/O interrupt. The programming system must respond by issuing the X'0A' read command. When the operator pushes the "END OF BLOCK" key (or its equivalent), the read command is completed.

3. Alarm (X'0B'): This command causes the console alarm to sound.

Most modern terminal devices do not require the operator to push some sort of request key before typing, and so 3215 mode processing may seem unusual; that support gives rise to many of the strange features of VM console support, such as the little-known SET AUTOREAD command in CMS. IBM operating systems continue to support the ancient 1052 and its newer 3215 version, even though the modern CPUs do not include such consoles.

12.2.2 3270 mode

By issuing the TERMINAL CONMODE 3270 command, a user can request that CP simulate a virtual 3270 console. The virtual machine must be running a programming system that supports the 3270 as a console. The real terminal must be a 3270 for this mode to be valid, since CP does not simulate 3270s on other types of terminal devices. The following is a summary of the channel commands that are commonly used in 3270 mode:

1. Write (X'01') and Erase/Write (X'05'): These commands write control information and output characters to the 3270. X'05' erases the screen before writing. The first character is a control character that can lock or unlock the keyboard or sound an alarm.

2. Read modified (X'06'): This command reads all modified fields from the screen. Note that this command is usually issued in response to an attention I/O interrupt from the 3270, which in turn is the result of the operator pushing one of the "action" keys such as ENTER or a PF key.

12.3 I/O SIMULATION REVIEW

General I/O simulation has already been covered in the chapter on I/O processing. For console SIO, much of the same logic is used, with the exception of the final simulation modules; the logic flow is as follows.

1. When the virtual machine issues a SIO instruction to begin a channel program for its console, a privileged operation program check occurs. The resulting PSW swap gives control to DMKPRGIN.

2. DMKPRGIN verifies that the virtual machine is in virtual supervisor state and goes to DMKPRVLG to simulate the privileged instruction.

3. DMKPRVLG checks the instruction's opcode and sees that it is SIO. It therefore goes to DMKVSIEX to simulate the execution of the virtual SIO instruction.

4. DMKVSIEX checks the status of the virtual console and then goes to DMKVCNEX to process the console SIO request.

5. DMKVCNEX works through the channel program and passes control via CALL to DMKQCNWT for writes and to DMKQCORD for reads.

6. DMKQCNWT and DMKQCORD first construct a CONTASK that describes the console I/O operation. They then pass control to various device-dependent routines to complete the SIO simulation processes. When those routines have finished, DMKQCN and DMKQCO return control to DMKVCN.

7. DMKVCN finishes the SIO simulation by setting the appropriate status bits and then going to DMKDSPCH. The virtual machine will be dispatched later following the SIO instruction, which will appear to have executed normally, just as on a real System/370.

12.4 REAL TERMINAL I/O

All of the processing that was described in the preceding section is independent of the type of terminal upon which the virtual console is being simulated. DMKQCN and DMKQCO must invoke device-dependent routines to build the actual channel programs that are needed to perform terminal I/O. In the followings sections we will first review several important control blocks and then we will examine the logic flow for several common types of terminals.

12.4.1 Control block review

Several control blocks are used extensively in console I/O processing:

1. The RxxxBLOKs describe the status of the real terminal devices.

2. The VxxBLOKs describe the status of the virtual console and its virtual I/O paths.

3. The BSCBLOK is used as an I/O work area for remote 3270 terminals.

4. The NICBLOK is used to logically subdivide the RDEVBLOK for remote 3270 terminals.

5. The CONTASK describes a single real terminal I/O operation. It contains flags, a VMBLOK address, a channel program, and any output data.

12.4.2 Slow-speed terminals

Slow-speed terminals are all those terminals that are not 3270s. In general, such terminals perform their I/O a line at a time and have no full screen support. The IBM 2741, 1050, 1052, 2150, 3101, 3210, 3215, 3767, and 7412 as well as ASCII ("TTY-compatible") terminals are all supported by DMKCNS. (DMKCNS derives its name from the fact that in release 1 of VM/370, no other terminals were supported as virtual machine consoles; all the support code was in DMKCNS.) Much of the logic is common to the different device types, but we will examine each one separately.

12.4.2.1 2741 and 3767

The IBM 2741 and 3767 typewriter terminals are supported identically by CP. These terminals transmit in a 6-bit code that is neither ASCII nor EBCDIC but is instead related to the positions of the characters on the 2741 typing element. Starting at the point where DMKQCN has prepared an output CONTASK, the logic flow is as follows:

1. DMKQCN stacks a CPEXBLOK to DMKCNSIC and then goes to DMKDSPCH.

2. DMKCNSIC first translates the output data to 2741 line code and then constructs a channel program to write the translated data with a possible added carriage return and a number of "idle" characters (to cover the carriage return time).

3. If no I/O operation is currently active on the device, then the channel program is started by a call to DMKIOSQR; DMKCNS then exits by going to DMKDSPCH.

4. If there was I/O active, and if it was a "prepare", then DMKCNS itself issues an HDV ("halt device") instruction to terminate the prepare. DMKCNS exits by going to DMKDSPCH to wait until the HDV completes. At that time, DMKIOT stacks the completed IOB. The dispatcher will later pass control to DMKCNS, which then starts the new I/O operation by calling DMKIOSQR; once again DMKCNS exits by going to DMKDSPCH.

5. Similar processing takes place if the active I/O is a read and the new I/O is a priority write; that is, the read is halted, the write is started, and the read is re-issued after the write has completed.

6. Once the write has completed, DMKIOT gets control and stacks the completed IOBLOK, resulting in the dispatcher later going to DMKCNSIN, which unstacks the CONTASK from the RDEVBLOK and then calls DMKQCOET.

7. DMKQCOET spools the output in case console spooling is enabled and then FRETs the CONTASK storage. If "return" was requested in the CONTASK, then DMKQCO builds a CPEXBLOK that will clean up and exit back to DMKVCN, as described above. DMKQCO then returns to DMKCNS.

8. DMKCNS starts the next available CONTASK, or, if none is available, it issues a SIO to begin a "prepare" channel program, which in effect waits until the user hits the "ATTN" key. DMKCNS then goes to DMKDSPCH.

For a console read request, starting at the point where DMKQCO has prepared an input CONTASK, the logic flow is as follows:

1. DMKCNS builds a channel program consisting of a write of a control character ("circle-C"), a read for the desired number of characters, a skip-read to consume any extra input, and a nop to force concurrent ending status.

2. This channel program is started as was described above for output CONTASKs.

3. When the read channel program completes, DMKIOT stacks the completed IOBLOK. As a result, DMKCNSIN later gets control to handle the interrupt.

4. DMKCNSIN examines the input buffer to see how many characters were read and whether or not the read ended with the standard control character sequence.

5. If the read completed normally, then the data is translated to EBCDIC and a new CONTASK is started to write a "circle-D" control to the terminal to lock its keyboard. DMKCNS calls DMKQCOET to finish processing the read CONTASK and to move the input data to the original user's buffer area. Control is given to DMKDSPCH, which will later dispatch the virtual machine.

6. If the read completed abnormally, DMKCNS will try again if data check was found or will reflect an attention interrupt by calling DMKQCOET as above.

12.4.2.2 TTY and 3101

The teletypwriter and IBM 3101 ASCII terminals are handled identically, except when the user has selected TERMINAL MODE 3101, in which case some special processing is performed for the 3101. In the standard TTY mode, input and output CONTASKs are processed by DMKCNS just as they are for 2741s, except that the control character sequences are different and the data is transmitted in ASCII.

For 3101 mode, DMKCNS calls DMKTTY (!) to generate special sequences.

1. For output, DMKTTY tries to compress multiple blanks into a special sequence to save data transmission time.

2. If the 3101 screen has filled (that is, the TERMINAL SCROLL count has been reached), then DMKTTY writes a "***MORE***" prompt and waits for the user to hit the ENTER key. DMKTTY then erases the prompt and allows the new output to be written.

3. For input CONTASKs, DMKTTY checks to see if a 3101 PF key was pushed, and if so handles the generation of the proper input data sequence, using the PF definitions that were given by the user via the SET PF command.

12.4.3 Local 3270 terminals

Locally attached 3277, 3278, and 3279 terminals are supported by DMKGRF, with some assistance from DMKGRA and DMKGRC. (When local 3270 support was added to VM/370, DMKGRF was so named to distinguish the new 3270 "graphics" terminals from the slow-speed terminals; over the years, DMKGRF has been split into several other modules.) Output CONTASKS are processed as follows, for the general case in which the screen does not become full:

1. DMKQCN passes control to DMKGRFIC, which is the device-dependent routine for local 3270 terminals.

2. DMKGRFIC must contend with many special conditions, such as keeping track of the current output line number and handling the generation of screen message such as "MORE" and "HOLDING".

3. DMKGRFIC builds an IOBLOK that contains a channel program to write the output to the 3270 and calls DMKIOSQR to queue up the IOBLOK for writing to the 3270 as soon as possible.

4. DMKGRF then goes to DMKDSPCH to wait for the write to complete.

5. When the write completes, DMKIOT gets control and stacks the complete IOBLOK. As a result, the dispatcher later goes to DMKGRFIN, which cleans up the CONTASK and calls DMKQCOET to return the contask to DMKQCN. DMKGRF then goes to DMKDSPCH.

For a read request, a different logic flow is needed since a response from the user is wanted.

1. DMKQCO goes to DMKGRFIC as described above.

2. DMKGRF sets up an IOBLOK and a TRQBLOK (timer interrupt queue block). The TRQBLOK is used for two purposes: (1) to act as a normal TRQBLOK to handle the 50 second and 10 second timeouts associated with the 3270 screen getting full, and (2) to serve as a general work area. The TRQBLOK points to an interrupt handling routine in DMKGRF.

3. If necessary, DMKGRF generates a data stream and a channel program to write the "CP READ" or "VM READ" messages into the status area. The usual sequence of DMKIOSQR, DMKDSPCH, and DMKIOTIN processing will follow.

4. DMKGRF exits to DMKDSPCH to wait for the user to enter the input and press an "action" key such as ENTER or a PF key.

5. When the user enters the input, an attention I/O interrupt is generated. DMKIOT gets control and builds and stacks an ATTN IOBLOK, which later results in the dispatcher passing control to the interrupt handling routine in DMKGRF.

6. DMKGRF sets up a channel program in the IOBLOK to issue the X'06' (read modified) channel command. DMKGRF calls DMKIOSQR to queue the read and then exits to DMKDSPCH.

7. When the read completes (a very short time later), DMKGRFIN is given control via the stacked IOBLOK and returns the input data to the virtual machine's buffer area as defined in the original virtual channel program.

8. Control is then passed to DMKDSPCH, which will eventually simulate a virtual I/O interrupt using the status information that has been moved from the IOBLOK to the VDEVBLOK.

Since the real 3270 keyboard is usually unlocked, it is possible for the user to enter input and press an action key at almost any time, even when the virtual machine does not have a read request active; that is in fact the normal condition for CMS command mode processing. In this case, DMKGRF reads the data as shown above and places the data into a special 1-line buffer. It then causes a virtual attention I/O interrupt to be stacked for the virtual machine console. When the virtual machine issues a SIO with a read request, DMKVCN can then simulate the read by using the data from the special buffer.

12.4.4 Remote 3270 terminals

Remote 3270 terminals represent a complex subsystem in themselves, and so we will discuss three aspects of their support: the hardware itself, the control blocks, and the program logic.

12.4.4.1 Hardware

The hardware associated with remote 3270 terminals consists of the following:

1. Attached to the channel is a 2701, 2703, 3704, 3705, or 3725 control unit capable of supporting the binary synchronous communications technique (BSC).

2. Attached to the TP control unit is a modem, then a TP line, and finally a second modem at the remote end.

3. Attached to the second modem is a 3271, 3274, 3275, or 3276 cluster controller. The cluster controller has a two-character address that is unique on the TP line. (Note that VM/SP supports only 1 cluster controller per TP line, although modifications have been produced to extend the support to allow multiple clusters per line in a "multi-drop" configuration.)

4. Attached to the cluster controller are various terminals or printers, such as the 3277, 3278, 3279, and 3287. Each such device has a two-character address that is unique on the cluster controller.

Unlike local 3270s, remote 3270s cannot cause an attention interrupt when the user presses an "action" key; this limitation is caused by the rules of BSC. As a result, CP must poll each cluster controller from time to time to see if any attached terminal has had an action key pushed. The

polling period must be chosen to give good apparent terminal response and yet not incur too much system overhead. Terminal response is also affected by the sharing of a single TP line for several terminals on the cluster controller and by the fact that input data streams must be broken into multiple blocks of up to 256 bytes each.

12.4.4.2 Control blocks

Since the TP line is defined in DMKRIO as a single device with a unique 3-digit hex address and RDEVBLOK, some additional control block structure is needed for CP to be able to support a user at each of the many attached remote terminals. The RDEVBLOK is reserved for doing low-level I/O to the TP line and contains pointers to two other control blocks:

1. The BSCBLOK is a general status and work area that is used by many of the BSC channel programs. The BSCBLOK is gotten from free storage when the operator starts the TP line.

2. The NICBLOK represents either the cluster controller or an attached terminal or printer. The NICBLOKs are generated with the CLUSTER and TERMINAL macros in DMKRIO and are grouped together for each TP line. The NICBLOK corresponds to the RDEVBLOK for a local 3270 in that it contains the anchors for queued terminal I/O and other information unique to a particular virtual machine.

Additional control information is carried in the CCWs that are used to perform I/O on the TP line. Byte 5 of the CCW is undefined by the I/O hardware, and so CP places into each CCW a code value that defines the purpose of the CCW.

12.4.4.3 Program logic

The following is a list of the routines that are directly involved in remote 3270 support; in each case, the major entry point names are given:

1. DMKGRAIN: I/O second level interrupt handler.

2. DMKGRATM: timer second level interrupt handler.

3. DMKRGBIC: process queued CONTASKs.

4. DMKRGBRE: perform a remote I/O operation.

5. DMKRGBFL: perform a polling operation.

6. DMKRGBSN: scan NICBLOKs for queued work.

7. DMKRGC : decode an input stream.

8. DMKRGDxx: extended data stream processing.

When DMKQCN and DMKQCO have queued CONTASKS onto the NICBLOK, they then go to DMKRGBIC, which is the main entry point for remote support. The major processing in DMKRGBIC is as follows:

1. If the TP line already has active I/O, then just exit to DMKDSPCH.

2. If there are no queued CONTASKS for any NICBLOK on this cluster, then just exit to DMKDSPCH.

3. If a write CONTASK is found, then build an IOBLOK, generate a write channel program, call DMKIOSQR, and exit to DMKDSPCH.

4. If a read CONTASK is found, start a polling operation and exit to DMKDSPCH.

All TP line IOBLOKs contain an IOBIRA field that points to DMKRGAIN; when the I/O completes, DMKIOTIN stacks the IOBLOK, causing DMKRGAIN to get control. The major processing is as follows:

1. Give any input data stream to DMKRGC for decoding.

2. Handle write completions.

3. Obey the various BSC protocol rules (ACK, NAK, etc.).

Much of CP's handling of remote 3270s involves polling the clusters to ask if any attached terminal has input to be read. The major polling loop is as follows:

1. Various routines pass control to DMKRGBSN, which executes the following logic:

   a) Scan the NICBLOKs for this cluster. If any queued CONTASKs are found, then go to the appropriate handler.

   b) If no CONTASK is found, and if BSCSCAN is set, then clear BSCSCAN and go to DMKDSPCH. This begins a polling delay period.

c) If BSCSCAN was not set, then build an IOBLOK containing the polling channel program. This channel program will terminate either with a NOP (code 7) if no terminal has any pending input or with a READ if some terminal did have pending input. Call DMKIOSQR to start the channel program and go to DMKDSPCH.

2. At I/O interrupt time, DMKRGAIN examines the terminating CCW. If it is the code 7 NOP then schedule a timer interrupt for the polling delay (.5 seconds), set BSCSCAN, and go to DMKRGBSN (above). If some other termination occurred, then go to the appropriate handling routine, which in this case will cancel the outstanding timer interrupt request.

3. If the polling timer interrupt occurs, then DMKRGATM gets control and goes to DMKRGBSN.

12.4.5 Logical device support

Logical devices provide a means for a virtual machine to work jointly with CP in such a way as to simulate a local 3270 terminal. The standard user of logical devices is VM/Passthru, but other programs may also use the interface. Console processing for logical devices is like that for local 3270 devices, except that the routines DMKHPSQR and DMKHPSDG take over the functions that have been described for DMKIOSQR (execute a channel program) and DMKIOTIN (handle an I/O interrupt). DMKHPS provides the interface between CP and what we will refer to as the "LDVM" (Logical Device Virtual Machine). We can view this support in the same way we viewed remote 3270 support, in terms of hardware, control blocks, and program logic.

12.4.5.1 "Hardware"

While there is no real hardware associated with logical device support, the following components are analogous to the 3270 hardware:

1. The LDVM corresponds to the 3274 controller. This virtual machine acts as a concentrator and interface between CP and the real users. Typically those users have terminals that are ATTACHed or DIALed to the LDVM, but there is no requirement that there be any real terminals or even users at all; the LDVM must simply be able to simulate the actions of real 3270 terminals.

2. Corresponding to the 3274 channel cables is a special software interface between CP and the LDVM. This interface is bidirectional, with DIAGNOSE X'7C' performing data transfer as well as LDVM/CP signalling and the "service processor external interrupt" (code X'2402') providing CP/LDVM signalling.

12.4.5.2 Control blocks

There are several control blocks associated with logical device support:

1. The X'2402' service processor external interrupt stores an interrupt code into the word of memory at location X'80' in the LDVM's virtual memory. This word is re-defined to contain the 2-byte device address, a 1-byte flag, and a 1-byte reason code, all of which are used by CP to indicate what action the LDVM should take next.

2. The field VMVMPS in the system VMBLOK contains a pointer to the "system communication block", for which there is no defined control block DSECT. This block contains 8 pointer words, each containing the address of the VMBLOK for the LDVM associated with a group of logical devices. The first pointer word is for the LDVM for devices X'4E00' through X'4FFF', the second is for X'4C00' through X'4DFF', and so on. Note that 8 LDVMs can be supported, and each can control 512 logical devices. This and SNA support are the only cases in which CP uses a 16-bit device address rather than the usual 12-bit address (for SNA support, all terminals have the address X'DEAF').

3. The field VMVMPS in the LDVM VMBLOK contains a pointer to the "CP diagnose console interface control block", which is mapped by the VMPSCOM dsect. This control block corresponds to a single logical device and the various VMPSCOMs are chained in a singly-linked list in device address order. The first half of the VMPSCOM contains pointers to data and CCWs being processed. The second half contains an RDEVBLOK so that many CP routines can behave as if the logical device were a real 3270. This RDEVBLOK is unusual in that it is located in free storage (not in DMKRIO), contains the X'4000' bit in the device address field, and has no associated RCUBLOK or RCHBLOK.

12.4.5.3 Program logic

In order to fully understand the program logic you should study the description of DIAGNOSE X'7C' in the *System Programmer's Guide*, since that explains the functions that are available to the LDVM. The following description will involve only the interactions between CP and the LDVM and will ignore the internal LDVM logic for supporting its real devices and users. We will examine the process by which a logical device LOGON proceeds; that will include initializing, CP writing, and CP reading after an attention interrupt.

1. Initialization begins when the LDVM issues DIAGNOSE X'7C' with the INITIATE function (Ry = 1). Control is passed through privileged operation simulation to DMKHPSDG for decoding of the DIAGNOSE parameters. DMKHPSDG INITIATE logic performs the following:

   a) Construct the VMPSCOM and master VMPSCOM if not yet present.

   b) Fill in the RDEVBLOK fields with their initial values for an unused local 3270 device.

   c) Call DMKSTKIO to stack a stand-alone DE IOBLOK to DMKGRFIN, which will in turn perform the standard functions for a new 3270 session (namely the writing of the VM logo).

2. When CP writes output to the logical device, the following two-step process takes place:

   a) After DMKGRF has constructed the write channel program, it will notice the X'4000' bit in the device address and rather than calling DMKIOSQR will instead call DMKHPSQR, which will queue an external interrupt X'2402' for the LDVM. The dispatcher will notice the queued external interrupt and will call DMKHPTEX to place the appropriate information into location X'80' of the LDVM; that information will show that the logical device has a write operation pending.

   b) The LDVM will receive the X'2402' external interrupt and will examine location X'80' to determine what kind of service is needed for which logical device. In this case, the LDVM will decide to ACCEPT the data that CP is trying to write to the logical device. The LDVM will perform the ACCEPT function by issuing a DIAGNOSE X'7C' with Rx = device address, Ry = 2, Rx+1 = a buffer address, and Ry+1 = the length of the buffer. DMKHPSDG will again get control and will do the following:

i) Find the VMPSCOM block for the logical device.

ii) TRANS in the LDVM page(s) containing the specified buffer.

iii) Move the CCW opcode (write or erase/write) from the CONTASK to the buffer. Move the data (the VM logo) to the buffer.

iv) Place the data length into the LDVM Ry.

v) Go to the dispatcher to let the LDVM resume execution following the DIAGNOSE instruction. The LDVM can then appropriately dispose of the output data (the VM logo).

3. Input processing is also a two-step process, which begins when the LDVM signals CP that input data is available for CP to read:

a) The LDVM simulates the ENTER key (to clear the VM logo) by issuing DIAGNOSE X'7C' with Rx = device address, Ry = 3, Rx+1 = the address of any input data, and Ry+1 = the length of that data. DMKHPSDG performs the following actions:

i) Find the VMPSCOM block for this logical device.

ii) TRANS in the LDVM page(s) containing the input data.

iii) If a CP read IOBLOK is already queued on the RDEVBLOK, then move the data into CP's buffer area and stack the IOBLOK with CE+DE for standard I/O completion.

iv) If no CP read IOBLOK is queued, then get free storage, copy the data, and create and stack an ATTN IOBLOK to DMKGRFIN.

b) When the ATTN IOBLOK is unstacked and control passes to DMKGRFIN, normal processing results in the building of a read IOBLOK, but DMKHPSQR is called (instead of DMKIOSQR):

i) If LDVM data is already available, then move the data into the CP buffer, release the temporary data buffer, and stack the IOBLOK with CE+DE.

ii) If no data is available, then keep the IOBLOK queued on the RDEVBLOK and stack an external interrupt X'2402' to the LDVM with the reason code at location X'83' showing the type of read (read buffer or read modified).

Logical device support is very neatly written and well structured. While it currently supports only 3277 and 3278 logical devices, there is no architectural reason why additional device types could not be added. The interface between CP and the LDVM is clean, simple, and efficient.

12.5 FULL-SCREEN PROCESSING

The previous descriptions of 3270 support have all assumed that TERMINAL CONMODE 3215 was in effect. That means that the virtual machine operation system uses 3215 channel programs and that CP translates those channel programs into 3270 operations; the virtual machine treats the 3270 as a line mode device.

The original support in VM/370 for the local 3270 terminals included a special facility by which programs could use the 3270 as a display terminal; this facility was used by the CMS EDIT command and was available for general application program use. Display terminal mode was later expanded to full-screen mode for use with the XEDIT editor; the new support allowed application program control of the entire 3270 screen as well as the PF keys. VM/SP also added support for CONMODE 3270, so that the virtual machine operating system can use the terminal as a 3270 while CP continues to use it as a line mode device.

12.5.1 Display terminal mode

12.5.1.1 Application program use

The initial full-screen support in release 2 of VM/370 added DIAGNOSE code X'58', by which the application program can specify the address of a channel program to be issued for a given device, usually the virtual console. The channel program can consist of one or more CCWs whose opcode is X'19' and whose "unused" byte (bits 40-47) contain the following:

1. X'FF' will just clear the 3270 screen immediately. X'FE' will clear the screen and will also insure that all previous terminal I/O has completed. In both of these cases no output data is written to the screen.

2. For other values, bit 40 (X'80') causes "MORE" status to be entered before the write is performed, even if the screen is already cleared. Bits 42-47 contain the line number where the data should start being written; the first line on the screen is line 0.

This facility allows writing into the top N-2 lines of the screen, with the loss of whatever was in those lines, or into the input area, with no change to the output area. The X'FF' or X'FE' flag value can be used in the first write to switch the screen without entering "MORE" status and therefore requiring the user to hit the PA2 key. Note that there is no special support for reading from the terminal. All the 3270 action keys (ENTER, PA2, PF1, etc.) perform their normal functions as defined by CP's 3270 line mode support.

12.5.1.2 CP program logic

This form of DIAGNOSE X'58' is handled very much like a normal console SIO instruction, except that the virtual machine remains in VMEXWAIT and VMIOWAIT until the I/O operation has been performed and no virtual I/O interrupt is generated. DMKVCN does not call DMKQCNWT as it does for line mode output; instead it calls DMKQCNWF and passes a parameter containing the screen line number at which the output is to be written:

1. If the output is to the input area (the line number is 2 less than the screen size), then the data is written to the screen just like line mode output. This operation does not interfere in any way with line mode output and does not affect the line count or the "MORE" status.

2. If the output is to the output area (anything above the input area), then "MORE" status will be entered if the screen is not already clear; the output data will be written after the user hits the PA2 key. Any subsequent line mode output will cause "MORE" status to be entered if there is data in the output area.

(The CP logic for entering the "MORE" status can be compared to a 3-position switch: the "left" position is line mode and the "right" position is display or 3270 mode. The switch is placed into the "center" position by anything that clears the screen, such as the CLEAR and PA2 keys or the DIAGNOSE X'58' codes given above. If the switch is moved from one side to the other, then a "MORE" condition will result. If the switch moves from either side to center or from center to either side, then the change will take place immediately and without a "MORE" condition.)

12.5.2 Full-screen mode

The XEDIT editor introduced in VM/SP requires total control of the 3270 terminal screen and keyboard. XEDIT must be able to write the full screen and must be able to intercept all of the action keys. Other application programs have similar requirements. As a result, DIAGNOSE X'58' was enhanced in VM/370 release 6 with BSEPP by providing two new CCW opcodes, X'29' for writing and X'2A' for reading. These opcodes allow the user to write and read standard 3270 data streams without modification by CP.

12.5.2.1 Virtual machine use

It is the responsibility of the application program to generate proper data streams for the user's 3270 terminal. The program must also be able to intercept the ATTN interrupt that results from the user pushing an action key; the program must be able to interpret the data stream that it then reads from the 3270 terminal. Since CP still uses the terminal as a line mode device for its own output, the application program must be able to recognize special status codes that indicate that CP had taken over the screen for its own output; the program must be able to refresh the previous screen contents in this case. The detailed description of DIAGNOSE X'58' in the *System Programmer's Guide* is complex, but it does fully and accurately represent the various status conditions. You should also examine the code in XEDIT or other full-screen applications if you want to see how the conditions are handled.

12.5.2.2 CP program logic

CP support for full-screen operations is basically quite simple; CP merely passes data unmodified between the virtual machine and the terminal. Complexities arise when CP must use the terminal for its own output. Two status bits are maintained in the field TRQBFLG2 in the TRQBLOK associated with the terminal:

1. CRTFSSA indicates that the application program is using the terminal in full-screen mode. This bit is set when an X'29' opcode is found with the 'ERASE/WRITE' flag set. CRTFSSA corresponds to the 'system available' lamp on the old 3277 terminal.

2. CRTFSII indicates that the terminal is in "input inhibited" status; the keyboard is locked and the user cannot enter data or push any action key. This bit also corresponds to a lamp on the 3277.

These two bits represent the state of the terminal and indicate whether or not CP can use the terminal for its own output. The four possible conditions are summarized in the following list:

1. SA=0 and II=0 indicates that CP is in control of the terminal. CP output and virtual machine line mode output will be written as it becomes available. A full-screen write from the virtual machine will place the terminal into "MORE" status until the user pushes the PA2 key.

2. SA=1 and II=0 indicates that a full-screen ERASE/WRITE has been issued and that only full-screen operations will be performed. Since II=0, input is not inhibited and full-screen reads may therefore be issued. CP output will be queued in memory waiting for II to be set to 1.

3. SA=1 and II=1 indicates that full-screen mode is still in effect but that the keyboard is now locked. This is the case after a full-screen read has been processed. If CP has output waiting, it will take control of the terminal immediately and start writing the output data. The next full-screen operation will end with a special CSW status, X'8E' (attention, channel-end, device-end, and unit-check). This special status tells the application program that the screen had been stolen and that the application should issue first a non-full-screen dummy read and then a full-screen ERASE/WRITE to restore the previous screen contents. As a result of the dummy read, the screen will be placed into "MORE" status so that the user can read the CP output; when he pushes the PA2 key, then the restored full-screen data will appear.

4. SA=0 and II=1 is a temporary condition that only exists between the attention interrupt from an action key and the full-screen READ command from the virtual machine.

12.5.2.3 Full-screen application example

Below is an example of some application program coding that could be used for full-screen processing. This is only the very basic code and does not include attention interrupt handling and screen image construction routines. This is given only as a general example and is not even guaranteed to assemble properly, much less perform any useful purpose.

```
START  LA   R0,ZERASE      Clear the screen to allow
       BAL  R14,IOREQ        full-screen operations.

*
*  This is the main loop.  Write a new screen image
*  and wait for the user to do something.  Decode the
*  attn interrupt and ignore everything but 'enter'.
*
WRITE  BAL  R14,BUILDSCR   Go build the screen image.
       LA   R0,ZWRITE      Write the screen using
       BAL  R14,IOREQ        erase/write, and handle
       BE   READCP           any cp screen robbery.
       BAL  R14,IOWAIT     Wait for an attention.
       LA   R0,ZREAD       Now go read the
       BAL  R14,IOREQ        new command, and
       BE   WRITE            re-write if stolen.
       CLI  ZAID,X'7D'     If this is not 'enter',
       BNE  WRITE            then just ignore it.
*** Here we can process the 'enter'ed data.
       B    WRITE          When done, go do it again.

*
*  In case cp took over the screen before a write,
*  issue a dummy read to clean things up.
*
READCP LA   R0,READUM      Issue a garden-variety
       BAL  R14,IOREQ        read to re-enter full
       B    WRITE            screen mode.

*
*** The 'BUILDSCR' and 'IOWAIT' routines
*** are left as exercises for the reader.
*



*
*  data areas and ccws.
*
READUM CCW  X'02',0,X'30',1  Dummy for stolen screen.
ZWRITE CCW  X'29',*-*,X'20',*-*  Full-screen write
       ORG  ZWRITE+5           with a flag:
ZSCFLG DC   X'80'              80: erase/write.
       ORG  ,
ZREAD  CCW  X'2A',ZAID,X'20',30  Full-screen read
       ORG  ZREAD+5            with a flag:
       DC   X'80'              80: read modified.
       ORG  ,
ZERASE CCW  X'19',0,X'20',1    Initial write to
       ORG  ZERASE+5           clear the screen
       DC   X'FF'              (avoid 'MORE'.)
       ORG  ,
CONADD DC   A(x'009')          Console address.
```

```
*
*  Subroutine to perform 3270 full-screen i/o.
*  R0 points to the channel program.  return with
*  cc = be to retry an error (cp stole the screen).
*
IOREQ  SSM   *+1             Be sure we have quiet.
       L     R1,CONADD       Be sure that any
IOR010 TIO   0(R1)             previous operation
       BC    6,IOR010          has completed
       BC    1,GONEAWAY        correctly.
IOR020 DIAG  R0,R1,X'58'     Start the channel program
       BC    8,IOR030          and continue if started.
       BC    4,IOR040        Check for any status bits.
       BC    2,IOR020        Loop if it was busy.
       BC    1,GONEAWAY      Quit if console is gone.
IOR030 TIO   0(R1)           Wait for the 'sio'
       BC    2,IOR030          to complete.
       BC    1,GONEAWAY      Quit if console is gone.
IOR040 CLI   X'45',X'00'     For channel errors
       BNE   GONEAWAY           we can only quit.
       CLI   X'44',X'0C'     If it completed normally,
       BE    IOR060              then we are all done.
       CLI   X'44',X'08'     If only channel end,
       BE    IOR050              go wait for device end.
       CLI   X'44',X'8E'     If cp stole the screen,
       BE    0(R14)              return with error cc.
       TM    X'44',X'B0'     For attn, cue, or busy,
       BNZ   IOR020              restart the diagnose.
       TM    X'44',X'0C'     If neither ce nor de,
       BZ    IOR020              then try it once again.
IOR050 TIO   0(R1)           Wait until device end
       BC    2,IOR050            finally comes in.
       BC    1,GONEAWAY      Quit if console is gone.
IOR060 MVC   ZCOUNT,X'46'    Save the residual count.
       LTR   R14,R14         Set a non-error condition
       BR    R14                 code and return.
```

12.5.3 3270 SIO processing

DIAGNOSE X'58' is defined to be a 3215 operation; it is valid only for TERMINAL CONMODE 3215. In some guest operating system environments it is better to have the virtual machine console run with TERMINAL CONMODE 3270; the guest system will then use SIO and 3270 channel programs for the console. CP support for full-screen mode has been extended to allow the use of SIO and the standard CCW opcodes for ERASE/WRITE, READ MODIFIED, etc., instead of the X'29' and X'2A' opcodes

for DIAGNOSE X'58'. Most of the same logic is applicable, since the same problem has to be solved: CP must be able to steal the console and use it in line mode while the virtual machine uses it as a 3270.

An additional feature is provided by TERMINAL SCRNSAVE ON; CP itself will save the contents of the screen before stealing it. The contents will be restored by CP when control of the terminal is switched back to the virtual machine. This support is needed because it may not be possible to change the guest operating system to handle the special X'8E' status (after all, the guest thinks that it has a normal 3270 console). If TERMINAL SCRNSAVE OFF is given, then CP will not save the screen contents before stealing the screen. In that case, CP will simulate the pressing of the CLEAR key, since that is the signal to many guest operating systems that the console screen must be rewritten.

12.6 SECONDARY USER FACILITY

CP provides support for a second userid to be associated with a virtual machine. The secondary userid is active whenever the virtual machine is running disconnected; that is, whenever the primary userid is not logged on to a terminal.

Whenever DMKQCN or DMKQCO try to perform terminal I/O and find that there is no terminal, they examine the VMSECUSR field. If it is non-blank, then it contains the userid of the secondary user. The I/O request is re-directed to that userid with an appropriate prefix to indicate the original userid. The CP command "SEND" may be used to simulate an input operation for a secondary userid; the support for SEND is contained in DMKSND.

NOTES

DMKRGAIN uses magic codes in CCWs (like HASP) to keep track of a CCW context.

01	@EOT	CC+SILI	2	1
09	@addr. seg.	CC+SILI	3	7
03	0	SILI	7	1
02	@INPUT BUFFER	SILI	10	263

Chapter 13

SPOOLING

13.1 INTRODUCTION

13.1.1 Overview

The spooling subsystem is the portion of CP that moves data between real and virtual unit record devices: printers, punches, and readers. Four major functions make up the spooling subsystem.

1. Virtual spooling support provides the simulation of printers, punches, and readers for each virtual machine.

2. Real spooling support provides I/O control for the real unit record devices attached to the computer system.

3. The system spool area provides DASD storage for keeping the spooled data available for long periods of time and across system IPLs.

4. Spooling commands allow the virtual and real spooling devices to be interconnected in many different ways. They also support the manipulation of spooled data files.

In this chapter we will start with the structure of the system spool area and then proceed to input spooling and output spooling. At the end, we will discuss the spooling commands and some other related topics.

13.1.2 References

13.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

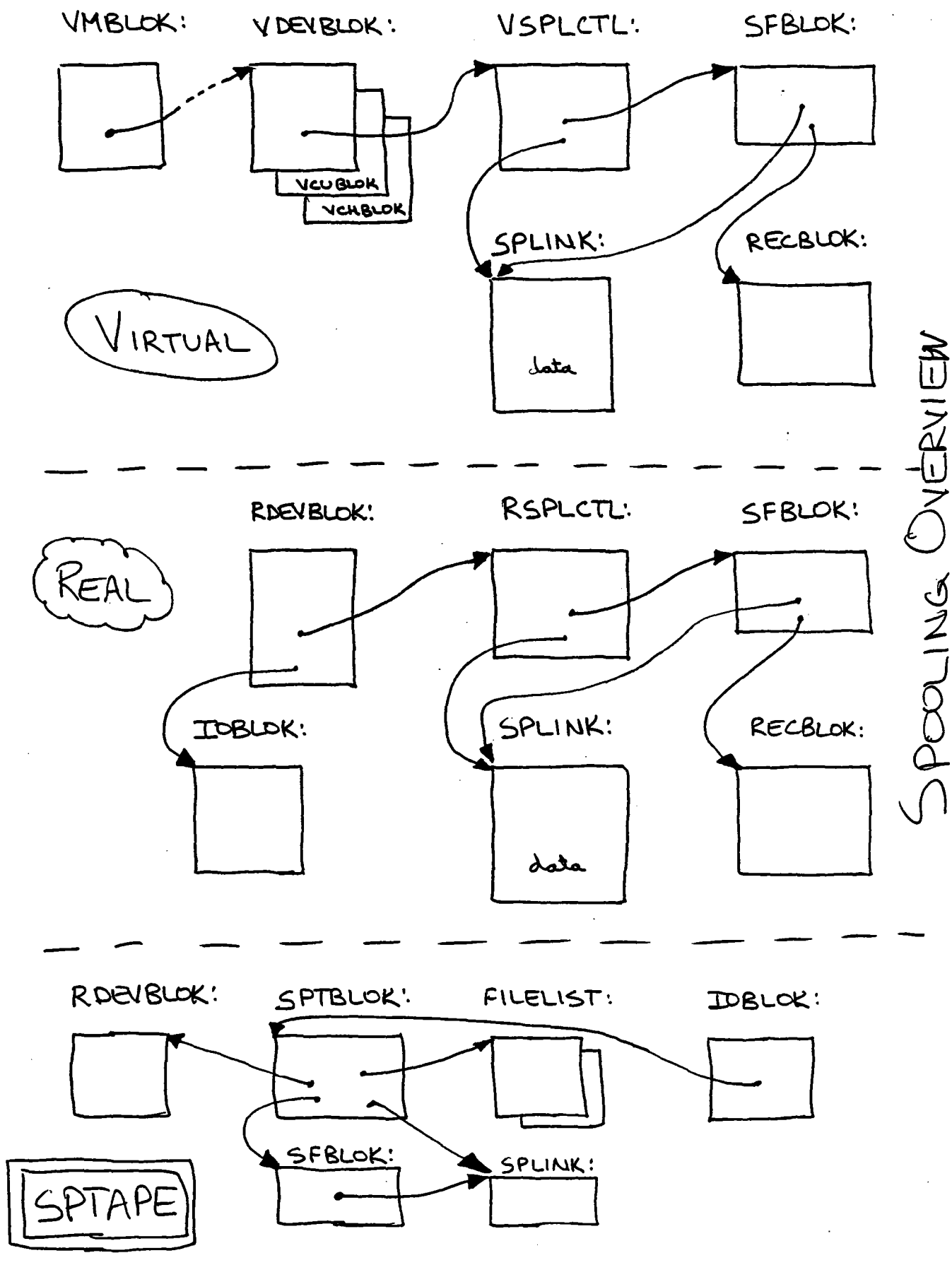
2. *IBM Virtual Machine/System Product: CP Command Reference for General Users* (SC19-6211).

3. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

4. For descriptions of the real (and virtual) spooling devices, see the appropriate Component Description manuals.

13.1.2.2 CP modules

1. DMKCPB - supports the NOTREADY, RESET, and READY commands.
2. DMKCPS - supports the SHUTDOWN command.
3. DMKCQR - supports the QUERY FILES.
4. DMKCSB - supports the LOADBUF and LOADVFCB commands.
5. DMKCSO - supports the real spooling commands.
6. DMKCSP - supports the SPOOL command.
7. DMKCSQ - supports the CLOSE, FREE, and HOLD commands.
8. DMKCST - supports the TAG command.
9. DMKCSU - supports the CHANGE command.
10. DMKCSV - supports the ORDER, PURGE, and TRANSFER commands.
11. DMKDEF - supports the DEFINE command.
12. DMKPIA - contains the 3289-E character sets.
13. DMKPIB - contains the 3262 character sets.
14. DMKRSP - is the real device spooling manager.
15. DMKRSQ - supports real spooling for the 3800.
16. DMKSPL - is the spool file manager.
17. DMKSPS - supports the SPTAPE command.
18. DMKSPT - supports the SPTAPE command.
19. DMKTCS - supports real 3800 separator and overlay.
20. DMKUCB - contains the real 3211 UCB images.


VMBLOK:
VDEVBLOK:
VSPLCTL:
SFBLOK:
VCUBLOK
VCHBLOK
SPLINK:
RECBLOK:
data
VIRTUAL
RDEVBLOK:
RSPLCTL:
SFBLOK:
REAL
IOBLOK:
SPLINK:
RECBLOK:
data
RDEVBLOK:
SPTBLOK:
FILELIST:
IOBLOK:
SFBLOK:
SPLINK:
SPTAPE
SPOOLING OVERVIEW

21. DMKUCC - contains the real 3203 UCB images.

22. DMKUCS - contains the real 1403 UCS images.

23. DMKVSP - is the virtual device spooling manager.

24. DMKVSQ - supports virtual output spooling.

25. DMKVSR - utility subroutines for DMKVSP.

26. DMKVST - supports printer output for CP commands and spooled console.

27. DMKVSU - utility subroutines for DMKVSP.

28. DMKVSW - continuation of DMKVSP.

13.2 SYSTEM SPOOL

The system spool area consists of two major parts, pointers and control blocks in real storage and data buffers on DASD. The major control blocks are the SFBLOK and the RECBLOK.

13.2.1 Pointers and control blocks

The system spool space has its origin in the CP nucleus, DMKPSA; the words ARSPPR, ARSPPU, and ARSPRD are the chain anchor pointers. They point to DMKRSPPR, DMKRSPPU, and DMKRSPRD, which in turn point to the first SFBLOK for each of the device types. Each SFBLOK points to the next SFBLOK in the chain. The SFBLOKs remain in real storage at all times while CP is running; they are copied to the system checkpoint area to allow CP to restart after a shutdown.

Each spool file is represented by its own SFBLOK, which contains the following information:

1. SFBPNT : memory address of the next SFBLOK.

2. SFBSTART: DASD address of the first data buffer.

3. SFBLAST : DASD address of the last data buffer.

4. SFBOWNER: userid of the current owner.

5. SFBORIG : userid of the originator.

6. SFBRECS : address of the RECBLOK.

7. SFBFNAME: the assigned file name.

8. SFBDATE : the creation date.

9. SFBTYPE : virtual device type.

10. SFBFILID: file id number.

11. various other items.

The RECBLOK is used to describe which DASD pages have been assigned to a spool file. Since each RECBLOK describes one cylinder, multiple RECBLOKs may be chained from the SFBLOK. The RECBLOK chain exists only when a spool file is active on a real or virtual device.

13.2.2 Data buffers

The actual data portion of the spool file is maintained in page-sized buffers residing on DASD. The buffers are resident in real storage only when data is being moved between the buffer and a real or virtual I/O device. Each buffer is described by the SPLINK DSECT in SPOOL COPY and contains the following:

1. the DASD address of the next and previous buffers.

2. the number of data records in the buffer.

3. the TAG information; the TAG is a NOP (with data but without the SKIP bit) and is located at the beginning of the first buffer.

4. the data records and the CCWs needed to process them.

The CCWs consist of a data transmission CCW (READ or WRITE) followed by a TIC to skip over the data itself. Since bytes 4 through 7 of the TIC CCW are unused by the I/O subsystem, the data area begins at byte 4; the TIC is therefore sometimes referred to as a "half-TIC". At the end of the buffer is a NOP CCW that is used to cause a real I/O device to return concurrent channel end and device end. The CCWs comprise a single chained channel program. Figure 26 shows the format of a spool file buffer. Note that SPNXTPAG and SPPREPAG are DASD pointers; they are in the format "CCPD" or "PPPD". The fields SPFILID and SPTIME are used as file validation indicators during checkpoint starts.

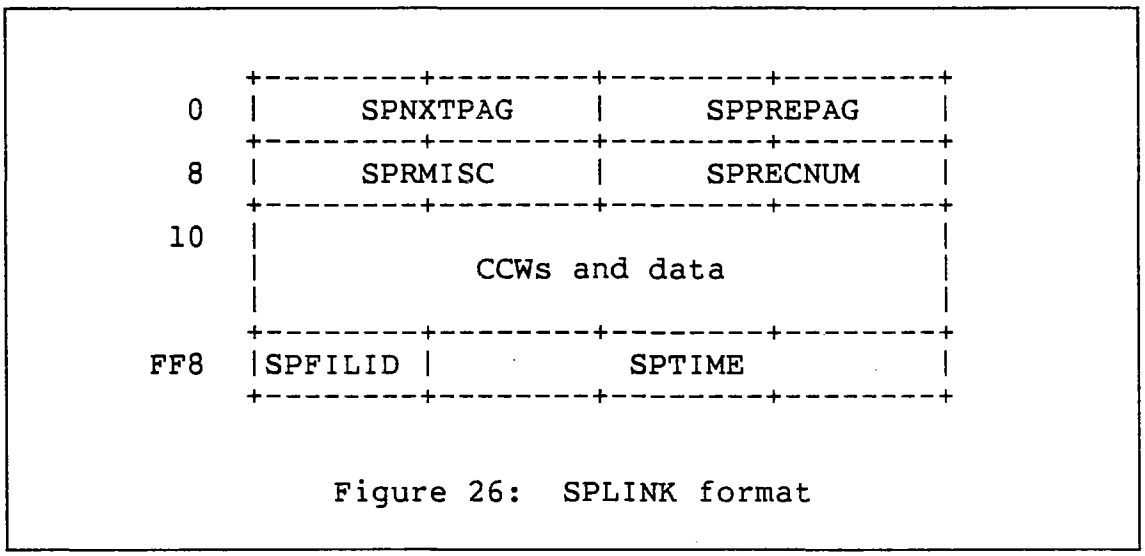


Figure 26: SPLINK format

13.3 INPUT SPOOLING

Input spooling consists of reading card images from a real reader and saving those images into the system spool. When the destination virtual machine issues a read request, the card images are then read from the spool and presented one at a time to the virtual reader.

13.3.1 Real reader I/O

Real reader I/O begins when the system operator places a card deck into the real reader and pushes the START button. The resulting standalone device end interrupt causes the following operations to take place:

1. Control is passed to the real spooling manager DMKRSPEX via an unstacked IOBLOK.

2. DMKRSP calls DMKSPLOR to open a reader file; this results in the allocation of an SFBLOK and a real memory page with a CP virtual address. The buffer page is filled with a channel program that will read 42 card images into the buffer.

3. DMKRSP then calls DMKIOSQR to read the first card into a temporary buffer. That card is scanned to find the destination userid and any optional parameters such as the file class.

4. DMKRSP calls DMKIOSQR to issue a SIO to the reader to read the next 42 cards, which will fill the buffer page.

5. When the I/O completes, DMKRSP calls DMKPGTSG to obtain a DASD slot to contain the buffer and calls DMKRPAPT to write the buffer into the assigned DASD slot in the system spool.

6. If there was a previous page in the spool file, then it is read by calling DMKPGUVG and DMKRPAGT. The forward pointer is updated and the page is re-written by calling DMKRPAPT and DMKPGUVR.

7. The reading process continues until physical end of file. The reader spool file is then chained onto the DMKRSPRD chain. If the destination virtual machine is logged on, then a standalone device end interrupt is sent to the machine at its lowest virtual reader address.

13.3.2 Virtual reader I/O

13.3.2.1 Review of virtual I/O

The general process of virtual I/O simulation has been discussed in another chapter. To review:

1. A virtual machine SIO results in a program check interruption, which causes DMKPRGIN to start running.

2. Since the virtual machine is in virtual supervisor state, DMKPRGIN goes to DMKPRVLG.

3. Since the opcode is SIO, DMKPRVLG goes to DMKVSIEX.

4. Since the virtual I/O device is a virtual reader, DMKVSIEX goes to DMKVSPEX.

13.3.2.2 Virtual reader SIO

DMKVSPEX simulates the SIO to the virtual reader by the following process:

1. If there is not already an open reader file, then:

   a) Get a VSPLCTL work area from free storage and link it to the VDEVBLOK.

b) Run the chain anchored at ARSPRD to find an available reader file for this user and this reader class.

c) Call DMKPGUVG to assign a virtual page in CP's own address space.

d) Call DMKRPAGT to read the first buffer page into that virtual address.

2. Call DMKVSRGC to get the next virtual reader CCW. Check it for a valid opcode.

3. Call DMKVSRMD to simulate the read operation:

   a) Call DMKPTRAN to page in the virtual machine's buffer, as addressed by the current virtual CCW.

   b) Call DMKPSASC to check for storage protection violations.

   c) Move the data from the spool buffer page to the virtual machine's buffer.

4. Advance the spool buffer pointer to the next spool CCW:

   a) Using the imbedded TIC, move to the next CCW in the spool buffer.

   b) If there is nothing left in the buffer, get the forward chain from SPNXTPAG and call DMKRPAGT to read the next DASD buffer into the virtual buffer area.

   c) Accumulate a chain of RECBLOKs for later use in freeing the DASD slots belonging to the spool file.

5. Finish by setting the appropriate "interrupt pending" bits in the VxxxBLOKs and constructing the virtual CSW. Go to DMKDSPCH.

13.3.2.3 Virtual reader close

When the end of the reader spool file is found, or when certain spooling commands are given, the spool file is closed:

1. For files that are to be kept, call DMKCKSPL to checkpoint the file.

2. For files that are to be purged, unchain the SFBLOK and call DMKSPLDL to delete the file.

3. DMKSPLDL puts the SFBLOK on a chain of 'delete' SFBLOKs, anchored from DMKRSPDL. If the delete task (DMKSPLDR) is not active, call DMKSTKCP to stack a CPEXBLOK that will start the delete task. Return to DMKVSP and from there exit to DMKDSPCH.

4. DMKSPLDR processes each SFBLOK chained from DMKRSPDL either by calling DMKPGUSR for the whole file or by calling DMKPGUSD for each buffer of the file. It then FRETS the SFBLOK.

13.4 OUTPUT SPOOLING

Output spooling includes both virtual and real processing for printers and punches. Console spooling is also a part of output spooling, but we will discuss it as a special topic at the end of this chapter.

13.4.1 Virtual printer I/O

The processing for virtual printers and virtual punches is almost identical; the same routines are used except when device differences force special handling. For that reason, we will discuss only virtual printer processing in this section.

13.4.1.1 Review of virtual I/O

The initial processing of a virtual printer I/O operation is exactly like the processing of virtual reader operations as described above, so please refer to that description.

13.4.1.2 Virtual printer SIO

DMKVSPEX simulates the SIO to the virtual printer by the following process:

1. If an open printer spool does not already exist, then call DMKSPLOV to:

   a) Obtain an SFBLOK and a VSPLCTL block from free storage.

b) Call DMKPGUVG to assign a virtual page in CP's address space.

c) Call DMKPGTSG to assign a page slot on a spooling DASD device.

2. Call DMKVSRGC to get the next virtual printer CCW and check it for valid opcodes. Also, if necessary, call DMKPGTSG to assign a new page slot on spooling DASD.

3. For CCWs that include data (or for NOPs with a length greater than 1):

   a) Call DMKVSRMD to move the data from the virtual machine memory to a work buffer.

   b) Add the write CCW and the trailing half-TIC.

   c) Truncate any trailing blanks in the data portion and adjust the CCW data length appropriately.

4. Call DMKVSQPD to move the CCW, the half-TIC, and any data to the current spool buffer page. If that page fills, then call DMKRPAPT to write the page to the spooling DASD device.

5. If there are more virtual CCWs, then repeat these steps.

6. At the end of the virtual CCWs:

   a) Set the appropriate "interrupt pending" bits in the VxxxBLOKs.

   b) Build the new virtual CSW.

   c) Set VMIOPND and clear EXWAIT in the VMBLOK.

   d) Go to DMKDSPCH to let the virtual machine continue.

13.4.1.3 Virtual printer close

The virtual printer is usually closed by the CLOSE command or by the CLOSE operand on the SPOOL command; these both result in a call to DMKVSPCO. The virtual printer is also closed if an invalid virtual printer CCW opcode is received; this is handled by DMKVSPEX calling its own internal subroutine PRTEOF, which is described below.

1. DMKVSPCO consists of the following logic:

a) If the virtual device is busy, then stack a CPEXBLOK that will cause control to return when the device is no longer busy.

b) Call the internal subroutine PRTEOF, described below.

c) EXIT to the caller.

2. The actual close process is handled by DMKVSP's internal subroutine PRTEOF:

a) If the spool file is still empty, then purge it.

b) Update the VMLINS counter.

c) TRANS in the last spool buffer, update its SPNXTPAG pointer, fill in the verification information at the end of the buffer, and write the buffer back to DASD via DMKRPAPT. Update the SFBLAST pointer.

d) Call DMKSPLCV to finish the virtual close process:

i) Finish the SFBLOK construction.

ii) Call DMKRPAGT to read the first buffer, set the TAG information, and call DMKRPAPT to re-write it.

iii) If the spool file is to be transferred to a virtual reader, then send a message to the recipient and reflect a DE interrupt to a suitable virtual reader.

iv) If the spool file is to be processed by a real spooling device, then add the SFBLOK to the appropriate chain and call DMKCSOSD to wake up the real devices.

e) Call DMKRPAGT and DMKPGUVR to release the virtual buffer page.

f) FRET the VSPLCTL block.

g) Return to the caller (the CLOSE or SPOOL command or DMKVSP itself).

13.4.2 Real printer I/O

Real output spooling support is similar to real input support in many ways. DMKRSP is the driving routine and it uses standard spool buffers with integral channel programs. Since real printer and real punch support use the same logic in almost all cases, we will discuss only printer support. The support consists of three major portions:

1. Starting the real device.
2. Writing the next spool buffer to the device.
3. Closing the device at the end of the spool file.

13.4.2.1 Starting the real printer

The real printer is started either by the operator issuing the START command or by the closing of a virtual printer file. The following actions take place:

1. DMKCSOST (START command) or DMKCSOSD (virtual printer close) calls DMKSTKIO to stack an IOBLOK containing a stand-alone DE interrupt. The interrupt address (IOBIRA) is DMKRSPEX, which is therefore given control when the IOBLOK is unstacked by the dispatcher. DMKRSPEX recognizes this initial DE interrupt and begins the processing of starting the printer.
2. DMKRSP gets storage for the RSPLCTL block and calls DMKPGUVG to allocate a virtual page in CP's address space.
3. For each of the classes that is enabled for the device, DMKRSP runs the chains of printer SFBLOKs, looking for the first one that matches and is available for processing. Checks are also made for proper forms requirements.
4. DMKURSTA is called to display a message on the operator's console.
5. DMKSEPSP is called to write out a printer separator page.

13.4.2.2 Write the next buffer

For subsequent device end interrupts on the printer, DMKRSPEX takes the following actions:

1. If that was the last buffer in the spool file, then call the internal routine "PRTEOF" described below.

2. Test for certain operator commands having been given, such as DRAIN or FLUSH, and if so perform those actions.

3. Call DMKRPAGT to page-in the next buffer from the spool file.

4. Accumulate a chain of RECBLOKs that indicate the pages belonging to the spool file.

5. Relocate the addresses in the CCWs to point to the data portions.

6. Call DMKIOSQR to start the new channel program to the device.

7. Go to DMKDSPCH while the I/O is active. The ending interrupt will begin this process again from the top.

13.4.2.3 Close the real printer

The internal subroutine "PRTEOF" gets control after the last spool file buffer has been written to the printer. This routine closes the printer as follows:

1. Call DMKSEPTL to write the trailing separator page.

2. Call DMKSPLDL to delete the spool file.

3. If another spool file should be printed, then start again at the SFBLOK search logic described above.

4. Call DMKRPAGT and DMKPGUVR to release the virtual buffer page.

5. Fret the RSPLCTL and the IOBLOK.

6. Go to DMKDSPCH.

13.5 SPOOLING COMMANDS

In this section we will briefly discuss the logic of each of the various spooling commands. The commands can be divided into two major groups:

1. The class G commands, which affect virtual spooling devices or the user's own spool files.

2. The class D commands, which affect real spooling devices or all users' spool files.

The processing modules described below are all pageable and are called by DMKCFM, the console function main routine.

13.5.1 Virtual spooling

The class G spooling commands, which affect virtual spooling devices and spool files, are processed as follows:

1. CHANGE: DMKCSUCH

   a) Run the chain of SFBLOKs and make changes as requested. Copy the changed fields into the first buffer page of the file.

   b) Call DMKCKSPL to checkpoint the changed SFBLOKs.

   c) If appropriate, call DMKCSOSP to start the real output devices.

2. CLOSE: DMKCSQCL

   a) Call DMKVSPCR to close a virtual reader.

   b) Call DMKVSPCO to close a virtual printer or punch.

3. DEFINE: DMKDEFIN

   a) Call DMKVDSDF to get the necessary VxxxBLOKs.

   b) Set initial values into the VDEVBLOK.

4. DETACH: DMKVDDDE

   a) Call DMKVDREL to perform the release function, which involves calling DMKCFQRD to reset any pending virtual interrupts and DMKVSPCx to close any active spool file.

b) Nullify the VDEVADD field to show that the virtual device does not exist. Nullify the device pointer in the VCUBLOK and possibly the controller pointer in the VCHBLOK.

5. LOADVBUF: DMKCSBVL

   a) Find the VDEVBLOK and get the VFCBBLOK or build a new one.

   b) Move the FCB image from DMKFCB.

6. NOTREADY: DMKCPBNR

   a) Find the VDEVBLOK and set VDEVNRDY.

7. ORDER: DMKCSVOR

   a) Find the desired spool file's SFBLOK.

   b) Find the user's first SFBLOK.

   c) Chain the desired SFBLOK in front of the first SFBLOK.

8. PURGE: DMKCSVPU

   a) Find the next SFBLOK to be purged.

   b) Call DMKSPLDL to schedule the actual purge operation, which will be carried out by DMKSPLDR.

   c) Repeat for all requested spool files.

9. QUERY: DMKCFJQU, DMKCQPRV, DMKCQGEN, DMKCQREY, DMKCQHQF, and DMKCQHQU

   a) For each of the various operands given below, call the named routine to actually search the SFBLOK chains and display the number of files or a summary of each file.

   b) For QUERY UR (class D), call DMKCQPRV.

   c) For QUERY UR (class G), call DMKCQGEN.

   d) For QUERY FILES, call DMKCQREY and go to DMKCQHQF.

   e) For QUERY RDR, PRT, PUN, READER, PRINTER, or PUNCH, call DMKCQHQU.

10. READY: DMKCPBRY

a) Find the VDEVBLOK and exit if busy.

b) Clear VDEVNRDY and set up a pending DE virtual interrupt.

c) Set VMIOPND.

11. SPOOL: DMKCSPSP

a) Set new values into the VDEVBLOK as requested.

b) For the CLOSE option, call DMKVSPCO.

c) For the FOR and TO options, call DMKUDRFU to validate the userid.

d) For the TO option, call DMKSTKIO to stack an IOBLOK with a virtual DE for the target virtual machine's reader.

12. TAG: DMKCSTAG

a) For TAG QUERY device, find the VDEVBLOK and the VSPXBLOK and display the tag information.

b) For TAG QUERY file, find the SFBLOK, get the first buffer page via DMKPGUVG and DMKRPAGT, and display the tag information.

c) For TAG device, find the VDEVBLOK and the VSPXBLOK and save the new tag information.

d) For TAG file, find the SFBLOK, get the first buffer page via DMKPGUVG and DMKRPAGT, save the new tag information, and re-write the buffer via DMKRPAPT.

13. TRANSFER: DMKCSVTR

a) For each spool file to be transferred, change SFBUSER.

b) If transferring to a user, send a message to the recipient and stack an IOBLOK with a DE for the user's first appropriate virtual reader. The IOBLOK will cause DMKVIOIN to get control.

c) If transferring to the system, call DMKCSOSD to start the system printer or punch.

d) Call DMKCKSPL to checkpoint the changed SFBLOKs.

13.5.2 Real spooling

The class D spool commands, which affect real spooling devices and spool files, are processed as follows:

1. BACKSPAC: DMKCSOBS

   a) Find the RDEVBLOK for the device and set RDEVBACK.

   b) Save the backspace count in the RSPLCTL block.

2. DRAIN: DMKCSODR

   a) Find the RDEVBLOK for the device and set RDEVDRAN.

   b) Send a message to the operator.

   c) Call DMKCKSPL to checkpoint the new device status.

3. FLUSH: DMKCSOFL

   a) Find the RDEVBLOK and set RDEVTERM.

   b) If requested, set SFBSHOLD and set SFBCOPY to 1.

4. FREE: DMKCSQFR

   a) For all of the user's SFBLOKs, clear SFBSHOLD.

   b) Call DMKCKSPL to checkpoint the changed SFBLOKs.

   c) Remove the user's SHQBLOK if there is no longer any hold status present.

   d) Call DMKCKSPL to checkpoint the hold queue status.

5. HOLD: DMKCSQHL

   a) Starting at DMKRSPHQ, find the user's SHQBLOK. If not found, create a new one.

   b) Set the print or punch hold bit in the SHQBLOK. Call DMKCKSPL to checkpoint the hold queue status.

   c) For all the user's SFBLOKs of the appropriate type, set SFBSHOLD.

   d) Call DMKCKSPL to checkpoint the changed SFBLOKs.

6. LOADBUF: DMKCSBLD

   a) Find the RDEVBLOK and build an IOBLOK.

b) Set up the CCW string, using data located in these routines:

   i) DMKPIA for 3289-E.

   ii) DMKPIB for 3262.

   iii) DMKUCB for 3211 UCB.

   iv) DMKUCC for 3203 UCB.

   v) DMKUCS for 1403 UCS.

c) Call DMKIOSQR to start the buffer load and go to DMKDSPCH.

d) When the I/O completes, fret all buffer areas and call DMKCKSPL to checkpoint the device changes.

7. REPEAT: DMKCSORP

   a) Find the RDEVBLOK for the device and find the active SFBLOK.

   b) Store the repeat count or the hold bit into the SFBLOK.

8. START: DMKCSOST

   a) Find the RDEVBLOK for the device and clear RDEVDRAN.

   b) Send a message to the operator.

   c) Stack an IOBLOK with a DE to cause DMKRSPEX to get control later.

13.6 MISCELLANEOUS TOPICS

The following topics are peripherally associated with the spooling subsystem.

13.6.1 Console spooling

Console spooling support is very much like spooled printer support, except that the data to be spooled comes from the console I/O routines rather from the virtual SIO simulation routines. The basic logic flow is as follows:

1. When DMKQCNWT or DMKCQORD have console data to be spooled, they call DMKQCOCS, which removes any imbedded 3270 SF characters and converts 3270 attribute characters to blanks.

2. DMKQCOCS then calls DMKVSTVP for each "line" to be spooled (long console output is broken into lines of up to 132 characters each):

   a) If the virtual printer is busy (due to active tracing, for example), then build a buffer containing the console data. Build a CPEXBLOK and add it to the chain of CPEXBLOKs that is anchored at DMKVSPWA. Exit back to the caller as if the data has been spooled.

   b) If the virtual printer is available, then perform the spooling operation as follows:

      i) If no console file is open, then call DMKSPLOV to open one. Call DMKCKSPL to checkpoint the new SFBLOK.

      ii) Change the output CCW to perform a skip to channel 1 for every 60 lines of output.

      iii) Call DMKVSQPD to actually move the data into the spool buffer.

   c) When the data is moved, check DMKVSPWA to see if there is a deferred CPEXBLOK for this user and this virtual device. If so, unchain the CPEXBLOK and call DMKSTKCP to give the CPEXBLOK to the dispatcher for imminent execution.

13.6.2 SPTAPE

The SPTAPE command is a special facility that allows a class D user to move spool files between DASD and tape. The module DMKSPT gets control from DMKCFM to process the SPTAPE command. DMKSPT parses the command's operands and sets up module DMKSPS to actually perform the tape I/O operations. DMKSPT constructs a special control block, the SPTBLOK, chained to the tape's RDEVBLOK. The SPTBLOK contains control information for use by DMKSPS and is released at the end of the requested processing. Note that the SPTAPE command itself terminates immediately, but its I/O operations will run to completion asynchronously.

13.6.2.1 Logic flow in DMKSPT

DMKSPT consists of a short main routine and several subroutines. The main routine itself decodes the first parameter, the operation code, and then divides into smaller routines for each of the 5 operations:

1. For 'DUMP', do the following:

   a) Use the subroutines RADDR and PRTPU to process the remaining parameters.

   b) Search the desired spool file chain and select those spool files that match the given selection parameters. Construct a chain of small control blocks, FILELISTs, containing the spool file id numbers of all the selected files.

   c) Use the subroutine INIT to build the SPTBLOK, an IOBLOK, and a page buffer. Chain the FILELISTs onto the SPTBLOK.

   d) Place into SPTINTR an index value that will later cause control to pass to the dump routine in DMKSPS. Use the subroutine CALLIOS to start a MODESET channel program and then exit to DMKCFM.

2. For 'LOAD' do the following:

   a) Use the subroutines RADDR and PRTPU to process the remaing parameters.

   b) Use the subroutine INIT to build the SPTBLOK, an IOBLOK, and two page buffers.

   c) Set the SPTLOAD flag to show that spool files are to be loaded, and set an index value into SPTINTR such that control will later pass to the SPTSCAN routine in DMKSPS. Use the subroutine CALLIOS to start a read channel program and then exit to DMKCFM.

3. For 'SCAN', follow exactly the same logic as for LOAD, but do not set the SPTLOAD flag.

4. For 'STOP', do the following:

   a) Use the subroutine RADDR to get the tape's RDEVBLOK.

   b) Set the SPTSTOP flag in the SPTBLOK and exit immediately to DMKCFM.

5. For 'CANCEL', perform the same logic as for 'STOP', except set the SPTCAN flag.

The subroutine INIT not only builds an SPTBLOK and an IOBLOK but it also locks into storage the pageable module DMKSPS so that the I/O interrupt processing routines in DMKSPS will be available when the tape channel programs complete.

13.6.2.2 Logic flow in DMKSPS

The IOBLOK for tape I/O contains an IOBIRA field that points to DMKSPSIO. That entry point therefore gets control from the dispatcher whenever an I/O operation completes on the tape drive. DMKSPSIO fields exceptional conditions such as unit check and unit exception (tape EOF) and then uses the index value in SPTINTR to pass control to the appropriate internal routine:

1. For 'DUMP', SPTINTR contains an index to the SPTSF routine, whose logic is as follows:

   a) If SPTCAN is set, then perform BSF, WTM, and RUN tape operations and set the SPTINTR index to pass control to SPTFREE, which will clean up everything and terminate the SPTAPE operation by going to DMKDSPCH.

   b) If SPTSTOP is set, then perform those same functions, except do not issue the RUN tape operation.

   c) If there is another SFBLOK to write to tape, get it and build a channel program to write it to tape. Write a message to the system operator indicating which file is now being dumped. Set the SPTINTR index to pass control to the SPTSPL routine at tape I/O completion. Start the channel program and exit to DMKDSPCH.

   d) If there are no more SFBLOKs to dump, then perform the selected termination operations (UNLOAD, REWIND, or LEAVE) and pass control to SPTFREE for clean up.

2. For the second phase of 'DUMP', the SPTINTR index has been changed to pass control to SPTSPL:

   a) If SPTCAN is set, then perform as described above.

   b) Call DMKRPAGT to page-in the first or next DASD page of the spool file. Build a channel program to write that page to tape. Leave the SPTINTR in-

dex so that control will continue to come to SPTSPL.

c) If this is the last page of the spool file, then change the channel program to do a final WTM and change the index to pass control to SPLSF. Start the channel program and exit to DMKDSPCH.

3. For 'LOAD' and 'SCAN', STRINTR contains an index to SPTSCAN, whose logic is as follows:

   a) If SPTCAN is set, perform a RUN operation and pass control to SPTFREE to clean up.

   b) If SPTSTOP is set, go directly to SPTFREE.

   c) If a unit exception occurred, then process the end of the spool file.

   d) If a normal read completed, and if it was an SFBLOK that should be loaded, then get storage for an SFBLOK and copy the contents. Call DMKCKTSF to get a new spool file id. Set the index to pass control to SPTSPLNK after the next tape read is complete. Start a channel program to read the first data buffer and exit to DMKDSPCH.

   e) If an SFBLOK read completed and if the spool file is not to be loaded, then start a channel program to perform the FSF tape operation and exit to DMKDSPCH.

4. The second phase of 'LOAD' continues with the routine 'SPTSPLNK', whose logic is as follows:

   a) For normal I/O completion, call DMKPGTSG to assign a new DASD spool slot and call DMKRPAPT to write the data buffer into the spool.

   b) If that was not the last data buffer in the spool file, then start a channel program to read the next buffer and exit to DMKDSPCH.

   c) If that was the last data buffer, then chain the SFBLOK onto the appropriate spool file chain and call DMKCKSPL to checkpoint the new file. Start a channel program to forward space over the pending tape mark and read the next SFBLOK. Set the SPTINTR index to pass control to SPTSCAN and exit to DMKDSPCH.

13.6.2.3 Potential problems with SPTAPE

There are several potential problems that you might encounter when using SPTAPE:

d) Tape error recovery consists of rewinding and unloading the tape. If the tape has subsequent good data, then you will have to manually reposition the tape beyond the error location. You might want to consider adding some logic to DMKSPSIO to do an FSF command instead of a RUN in this case.

e) There is no interlock to control simultaneous SPTAPE LOAD operations. If you have multiple tapes to load, be VERY careful about trying to load them at the same time. It is possible that you can safely load a print file tape simultaneously with a punch file tape, but be very careful.

13.7 SUMMARY

The spooling subsystem includes portions of several major functional areas: real I/O device support, virtual I/O device support, paging, and command processing. The virtual device support includes not only the virtualization of the corresponding real devices but also new functions such as TRANSFER that are the virtualization of many real-life operations involving the physical movement of unit-record data.

NOTES

Chapter 14

SPOOL FILE RECOVERY

14.1 INTRODUCTION

14.1.1 Overview

The CP spool file recovery facility is designed to allow CP spool files to continue to exist across planned and unplanned system outages. The spool files themselves are stored on DASD in TEMP space and are therefore available after CP is reloaded; the SFBLOKs and other associated data, however, are normally present only in main storage and must therefore be saved on DASD to allow their continued existence. CP provides two methods of preserving the data: warmstart and checkpoint.

The warmstart area is used as temporary storage for critical data during the time that the CP nucleus is being loaded or reloaded; the data is written once and read once for each system outage. The checkpoint area is used as backup storage in case the warmstart cannot be used; the data is written incrementally and is read only if necessary.

Please note that the term "checkpoint" is often used within CP to mean either the warmstart facility or the checkpoint facility, depending upon the context. That confusion is basically historical in nature but it makes reading the code extremely difficult in places. For that reason we have used the term "spool file recovery" instead of the term "spool file checkpoint" that is more typically used by IBM. In a similar fashion, the terms "warm" and "cold" may have slightly different meanings in various contexts.

14.1.2 References

14.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

2. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

14.1.2.2 CP modules

1. DMKCKP - write warmstart data.
2. DMKCKS - write checkpoint data.
3. DMKCKT - checkpoint recovery.
4. DMKCKV - checkpoint recovery.
5. DMKCPI - CP initialization.
6. DMKDMP - CP ABEND restart.
7. DMKSAV - reload CP nucleus.
8. DMKWRM - read warmstart data.

14.2 DATA AREAS

14.2.1 Warmstart area

The warmstart area is located on the SYSRES device; its block or cylinder address is stored in DMKSYSWM. The area is in standard CP page format and each page contains multiple logical records of various lengths. The warmstart area is unused during normal CP operations; only when the system is being shutdown or restarted does CP write into the warmstart area copies of all the critical data from memory. The data is read at the next initialization of CP, either at the next IPL or during CP restart.

The warmstart area contains the following:

1. A header: 8X'FF' and a code number that must be present at the beginning of each page of the warmstart area. The 8X'FF' indicates that the warmstart area was completely written during the previous shutdown processing and the code is checked in all warmstart pages to help insure that the data is logically correct.
2. Various RDEVBLOK status bits for the enabled terminals so that they can be re-enabled when the system is started again.
3. All the queued accounting cards that have not yet been transferred to the assigned accounting card receiver virtual machine.
4. The system log messages.

5. All defined SFBLOKs.

6. All DASD RECBLOKs (showing DASD slots allocated to spool files).

7. All SHQBLOKs (containing all userids, all of whose spool files are being held).

14.2.2 Checkpoint area

Like the warmstart area, the checkpoint area is located on the SYSRES device and its starting cylinder or block address is contained in DMKSYSCH. The area is in page format and is a maximum of 369 pages long. The checkpoint data is a copy of the important data (but not everything that is saved in the warmstart area) but, unlike the warmstart area, it is written as the data is changed. The area is divided into "slots" that are used in a pseudo random fashion to reduce overhead. The checkpoint data is used only when the warmstart data cannot be used; additional processing must be performed by CP to regenerate all the critical data that warmstart would have preserved.

The checkpoint area contains three major structures:

1. The first page contains the spool fileid bit map, which contains 1 bit for each possible spool fileid (1 through 9900). A bit is 1 if the corresponding spool fileid is in use. This map is used to prevent the duplicate use of a fileid number.

2. The next 1 to 6 pages contains the "slot map". Each map item is a halfword and corresponds to a "slot" in the third portion of the area. The possible values for each map item are:

   a) X'FFFF': unused slot.

   b) X'FEFE': the current logical end of the map.

   c) X'FEEE': the physical end of the map.

   d) X'EEEE': slot has RDEVBLOK status bits.

   e) X'AAAA': slot has an SHQBLOK.

   f) X'nnnn': slot has the SFBLOK for fileid X'nnnn'.

3. The remaining pages of the checkpoint area contain the "slots", each of which is an SFBLOK size in length, so that there are 34 slots per page. If the

slot actually contains an SHQBLOK or some RDEVBLOK status bits, then the extra space in the slot is unused.

14.2.3 Pointers in DMKRSP

The real spooling manager, DMKRSP, serves as a convenient location for several pointers and data words that must be resident in main storage. Some of the more important ones are:

1. DMKRSPSW: is X'00' if checkpointing is alive and well. If this is X'FF', then no checkpointing will be performed.

2. DMKRSPEC: the number of checkpoint slots available.

3. DMKRSPSM and DMKRSPSR: the virtual and real addresses of the spool fileid bit map.

4. DMKRSPM1 through DMKRSPM6: the virtual addresses of the 6 checkpoint slot map pages.

5. DMKRSPID: the next spool fileid to be assigned.

14.3 PROGRAM LOGIC

This section describes the logic of warmstart and checkpoint.

14.3.1 Warmstart logic

Warmstart logic can be divided into three major areas:

1. SHUTDOWN processing for normal system termination.

2. ABEND processing for abnormal system termination.

3. IPL processing for system initialization.

14.3.1.1 SHUTDOWN logic

The logic flow resulting from the class A operator command SHUTDOWN is as follows:

1. The console function master routine DMKCFM calls DMKCPSSH to process the SHUTDOWN command.

2. DMKCPSSH stores 'CPCP' into CPID at X'370' in DMKPSA. That word is used as a flag by many of the warmstart routines.

3. DMKCPSSH calls DMKDMPRS, which simulates an IPL sequence by using the "read IPL" CCW opcode X'02'. The result is that DMKCKP is loaded from the SYSRES device, just as if the operator had pushed the IPL button on the real system console.

4. DMKCKP first loads the rest of itself, since DMKDMPRS has loaded only the first page of DMKCKP. At this point CP ceases to operate; DMKCKP takes over the new PSWs and with them the entire real system.

5. DMKCKP writes the warmstart data into the warmstart area on the SYSRES device. The data is collected from its various sources, such as the RDEVBLOKs, SFBLOKS, etc.

6. DMKCKP writes 8X'FF' into the first warmstart record to signal that it has in fact completely saved all the critical data. It then moves 'SHUT' to CPID (X'370'), issues the 960I and 961I messages and issues LPSW to load a wait state PSW with a code of 8. At this point VM/370 has been successfully shut down.

14.3.1.2 ABEND logic

CP errors result in the execution of an ABEND macro (totally different from the OS macro of the same name).

1. A CP ABEND passes control to DMKDMPDK to write the system dump.

2. DMKDMPDK moves 'WARM' to CPID at X'370' and loads DMKCKP by simulating the IPL sequence as described above. (The 'WRM' indicates that the "system is warm"; that is, critical system data is present in memory and must be saved.)

3. DMKCKP first loads the rest of itself.

4. DMKCKP writes the warmstart data and the completion code 8X'FF' as described above.

5. DMKCKP then loads and goes to DMKSAVRS.

14.3.1.3 IPL logic

At this point, we will break the logic flow and start again, this time following what happens when the system operator pushes the IPL button on the real system console.

1. DMKCKP is loaded from the SYSRES device into X'800' or X'800' above the V=R area.

2. DMKCKP loads the rest of itself since the IPL process loads only the first page of DMKCKP.

3. If CPID (X'370') = 'CPCP', DMKCKP performs the SHUTDOWN processing described above. This allows an orderly shutdown even if the system operator cannot issue the SHUTDOWN command in the normal manner (for example, if the system console became inoperative.)

4. Otherwise, DMKCKP moves 'COLD' to CPID (X'370') and loads and goes to DMKSAVRS. (The 'COLD' indicates that the "system is cold"; that is, there is no critical data to save. This does NOT imply that a "cold start" will be performed.)

At this point, the logic flows for ABEND and IPL join and continue as follows:

1. DMKSAVRS loads the rest of the CP nucleus from the nucleus area on the SYSRES device.

2. DMKSAVRS then goes to DMKCPINT, the actual initialization routine.

3. (DMKCPINT performs many functions that are beyond the scope of this chapter; only the warmstart and checkpoint functions are given here.) DMKCPINT asks the operator what kind of start is desired: WARM, COLD, CKPT, or FORCE (these choices are further explained below).

4. DMKCPINT then calls DMKWRMST to perform the requested type of start. (DMKWRMST logic is given separately below.)

5. DMKCPINT then goes to DMKDSPCH to start running the system.

Let us now look more closely at the processing that is performed by DMKWRM, since that is at the heart of the warmstart facility. Based upon the operator's response, one of the following logic paths is followed:

1. A response of "WARM" indicates that CP should read the warmstart data and use that data to reconstruct the critical data.

   a) If the first record does not contain the 8X'FF' flag, then issue the DMKWRM920I message and enter a wait state 9.

   b) Otherwise, read all the warmstart records and rebuild all the critical data: terminal enables, accounting cards, logmsgs, SFBLOKs, RECBLOKs, and SHQBLOKs.

   c) Call DMKCKSPL to checkpoint everything in case of future failures.

   d) Store the TOD clock value into the first warmstart record and also into DMKRSPCV so that DMKCKP can perform the verification described previously.

   e) Exit back to DMKCPI with the spool files completely restored.

2. A response of "COLD" indicates that CP should begin with no spool files at all, and that all other warmstartable functions be set to their null condition.

   a) Call DMKCKSIN to initialize the checkpoint area, making it completely empty.

   b) Exit back to DMKCPI with no spool files at all.

3. If the response is either CKPT or FORCE, then the operator is indicating that the warmstart data should be ignored and that CP should try to rebuild the critical data from what is available in the checkpoint area. That logic is described in the next section.

14.3.2 Checkpoint logic

As with warmstart, there are two basic operations in checkpoint: saving data into the checkpoint area, and restoring data from the checkpoint area.

14.3.2.1 Saving the checkpoint data

Routines making changes to the critical spool file data will call DMKCKSPL to insure that the changes are recorded in the

checkpoint area. The logic of DMKCKSPL is quite simple. The following notes should help you when looking at it:

1. Upon entry, R2 contains flags that indicate the type of control block to be saved and the type of change which occurred. Typical flags are "add a new SFBLOK", "change an SFBLOK", "delete an SFBLOK", "change an RDEVBLOK", etc.

2. A subroutine, DMKCKTFS, is called to locate the slot for the specified control block. The slot contents are updated according to the request and if necessary the slot map item is also updated. If an SFBLOK is being added, then DMKCKTSF is called to assign the spool fileid and set the corresponding bit in the bit map. If an SFBLOK is being deleted, then DMKCKTSD is called to clear the bit in the bit map.

3. Pages are read using the standard sequence of calling DMKPGUVG to assign a virtual page address and then calling DMKRPAGT to read in the page. Writing is performed by the standard sequence of first calling DMKRPAPT to write the page back into its DASD slot, then calling DMKRPAGT (!) to release the real storage frame, and then calling DMKPUGVR to release the virtual page. For pages that have a permanently assigned virtual address, DMKPTRAN is used to read the page or to locate it if it is already in real storage.

4. The subroutine "NEXTCYL" really gets the next page, not the next cylinder.

14.3.2.2 Restoring the checkpoint data

The logic for restoring the checkpoint data is located in DMKCKVWM, which is called from DMKWRMST when the operator requested CKPT or FORCE. DMKCKVWM does the following:

1. Perform some initialization by calling DMKCKTMP, which builds pointers to the checkpoint maps.

2. Process each slot map item as follows:

   a) If the slot map item is X'FFFF', then skip it since it is unused.

   b) If the slot map item is X'EEEE', then page-in the slot and set the RDEVBLOK status bits according to the slot's contents.

c) If the slot map item is X'AAAA', then page-in the slot and build an SHQBLOK in free storage using the slot's contents.

d) If the slot map item is a spool fileid, then page-in the slot and build a new SFBLOK in free storage using the slot's contents. Read each spool file buffer and construct RECBLOKs to indicate which TEMP space slots are allocated to the spool file.

3. Perform the above steps for all slot map pages. Then unlock and release the appropriate pages and exit to DMKWRMST.

14.4 SUMMARY

Warmstart logic allows the memory-resident portions of the spooling system to be saved across most system outages. Checkpoint logic allows recovery in case the warmstart becomes invalid. Variations of checkpoint processing allow the recovery of valid spool files and the elimination of erroneous files.

NOTES

Chapter 15

CONSOLE FUNCTIONS AND CP COMMANDS

15.1 INTRODUCTION

15.1.1 Overview

The VM user is the system operator of the virtual machine. CP simulates the virtual machine system console via a set of commands and messages. Most of the System/370 system console functions are available, along with some additional functions. Commands allow the user to control the operation of the virtual machine and to modify the virtual machine configuration. This chapter will discuss the functions that are available to all users for use with their own virtual machines. We will not discuss the additional privileged commands that can be used by system operators and system programmers to control the real VM system, but most of those commands are processed in a similar manner.

15.1.2 References

15.1.2.1 Publications

1. *IBM System/370 Principles of Operation* (GA22-7000) is the official public description of the hardware, as seen from the point of view of a program.

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203) describes some simulated additional instructions invented by CP.

3. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202) describes the special system operator commands.

4. *IBM Virtual Machine/System Product: CP Command Reference for General Users* (SC19-6211) describes the general purpose commands.

5. *IBM Virtual Machine/System Product: Commands (General User) Reference Summary* (SX20-4401) is a concise listing of command syntax.

6. *IBM Virtual Machine/System Product: Commands (Other than General User) Reference Summary* (SX20-4402) is a concise listing of command syntax for the privileged commands.

15.1.2.2 CP modules

1. DMKCFC - is the command name lookup routine.
2. DMKCFM - has the main processing loop for commands.
3. DMKCVT - contains many conversion subroutines.
4. DMKSCN - contains the main parameter scan subroutine.
5. Many other modules are called for the detailed processing of the various CP commands.

15.2 SYSTEM CONSOLE FACILITIES

There are several different kinds of system console facilities that are simulated by CP, as listed below.

1. Keys and indicators are the operator interface controls as defined for the System/370 console. The keys are implemented as a group of CP commands and the indicators are implemented as CP messages.
2. The "system activity display" or system activity meter is implemented via a second group of CP commands. These give the user various measurements of real or virtual machine activity.
3. The IBM Customer Engineer is simulated, in effect, by other CP commands that can alter the virtual machine configuration.

Table 21 gives a summary of the system console functions and their corresponding CP commands and messages.

TABLE 21

System console functions

Console function	*CP cmd or msg*
Address compare controls	ADSTOP and PER
Configurator controls	DEFINE and SET
Display and enter controls	DISPLAY and STORE
Interrupt key	EXTERNAL
Load key	IPL
Load unit address controls	IPL
Manual indicator	"CP Read" msg
Power off key	LOGOFF
Power on key	LOGON
Rate control	PER and TRACE
Restart key	RESTART
Start key	BEGIN
Stop key	#CP or "PA1" key
Store status key	STORE STATUS
System reset key	SYSTEM RESET
Wait indicator	"disabled wait" msg
System activity meter	INDICATE and QUERY
IBM Customer Engineer	DEFINE and DETACH

15.3 COMMAND PROCESSING

"Console function" mode is defined as being in effect whenever the user's input is directed at CP and not at the virtual machine. On a 3270 terminal, console function mode is indicated by "CP READ" appearing at the bottom of the terminal screen. Non-3270 terminals indicate console function mode by the prompt string "CP". The virtual machine is placed into console function mode by any one of several events:

1. The user presses the "PA1" key on a 3270 terminal that was in a "RUNNING" or "VM READ" condition.

2. The user types in a line consisting of the characters "#CP", where "#" is the current TERMINAL LINEND character.

3. The user hits the BREAK key twice in rapid succession on a line mode terminal.

4. A virtual machine program issues the DIAGNOSE X'8' instruction with a null command string.

5. A virtual machine encounters an ADSTOP or a PER event.

We are going to describe the case in which the virtual machine stays in console function mode for a series of commands. It is possible to execute just one command, either by prefixing it with "#CP" or by issuing it with DIAGNOSE X'8'. In such a case, the virtual machine leaves console function mode at the end of the command processing. For the purposes of this discussion, we will also assume that "SET RUN ON" has not been issued.

When in console function mode, the virtual machine is in the System/370 "manual" state, as if the virtual STOP key had been pressed. All input is now interpreted as CP commands until the virtual machine is taken out of console function mode (that is, until the virtual START key is pressed). The routine DMKCFM contains the main processing loop for commands.

Note that the command processing is performed as a part of the terminal I/O interrupt or the program check interrupt for DIAGNOSE. If the command processing is complex or requires many I/O operations, then the processing routines may stack a CPEXBLOK to allow the processing to continue asynchronously.

15.3.1 DMKCFM logic

Once console function mode has been entered, DMKCFM processes each command in the following basic loop:

1. Show that the virtual machine is in the stopped state by writing "CP" to a line mode terminal or "CP READ" to a 3270 terminal. This corresponds to the real "manual" indicator light.

2. Allocate a BUFFER control block from free storage and read the next command from the terminal. Place the command address and length into the BUFFER pointer words. Use R9 to address the BUFFER.

3. Call DMKSCNFD to scan off the command name field.

4. Call DMKCFCMD to check the command name against the list of valid names. For a valid command, DMKCFCMD returns the address of the command's processing routine.

5. Call the processing routine. (A few commands will not return to DMKCFM but will instead remove the virtual machine from console function mode and allow it to resume execution.)

6. Return to the second step in the loop; that is, read the next command.

The BUFFER control block is used to pass the command string to all subsequent processing routines. Figure 27 shows the format of the BUFFER. The command string is in BUFIN, and BUFNXT and BUFCNT contain the address and length of the as-yet-unprocessed portion of the string.

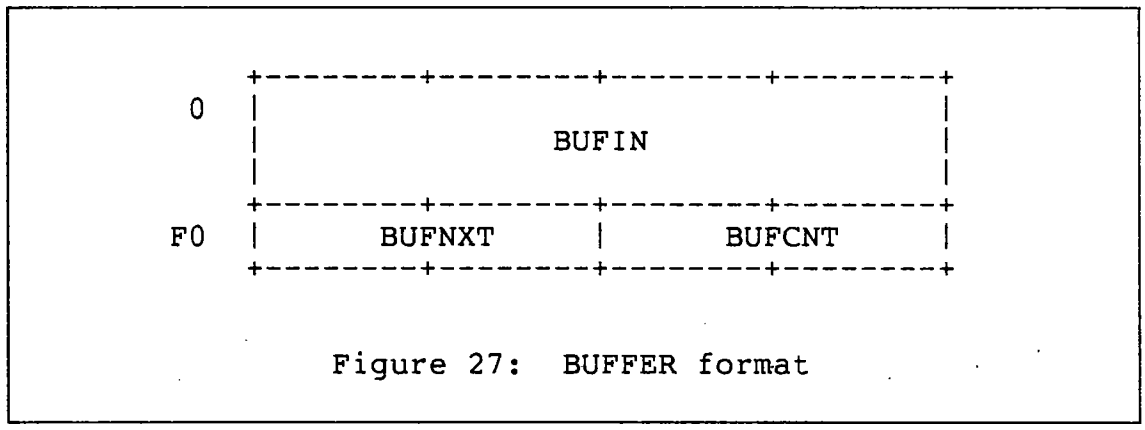


Figure 27: BUFFER format

15.3.2 DMKCFC logic

Module DMKCFC consists of a small search routine and a table of valid CP commands. Using the command name and length passed to it as parameters from DMKCFM, the search routine locates the command table entry and checks that a sufficient abbreviation has been given and that the user is allowed to issue the command. This check is performed by using the VMCLASS field in the VMBLOK; VMCLASS specifies the command classes (A through H) that are enabled for this user. DMKCFC compares the classes enabled for a user and the classes permitted to execute the command. If at least 1 permitted class is found in VMCLASS, then the command is considered legal. Otherwise, DMKCFC returns with an error code. Each entry in the command name table is 16 bytes long and is generated by a local macro called COMND. The COMND macro specifies the name of the command, its minimum abbreviation length, the classes permitted to execute the command, and the address of the command's actual processing routine. Table 22 gives several entries from DMKCFC's com-

mand table. Notice that a command can be enabled for multiple classes.

TABLE 22

DMKCFC command table

```
                  +------------ command name
                  |      +------ permitted privilege class
                  |      | +---- minimum length
                  |      | | +-- routine to call
                  |      | | |
COMNBEG0 COMND LOGON,0,1,DMKLOGON,NCL=1
         COMND LOGIN,0,1,DMKLOGON,NCL=1
         COMND DIAL,0,1,DMKDIAL,NCL=1
COMNBEG1 COMND ATTACH,B,3,DMKVDAAT
         COMND ATTN,G,4,DMKCFJRQ,TYPE=A
         COMND ADSTOP,G,6,DMKCFDAD
         ...
         COMND SET,A+B+E+F+G+H,3,DMKCFJSE,TYPE=A
         COMND SHUTDOWN,A,8,DMKCPSSH
         COMND SLEEP,0,2,DMKCFJSL,NCL=1,TYPE=A
         COMND SPACE,D,3,DMKCSOSP
         ...
         COMND WARNING,A+B,1,DMKMSGWN
         COMND *,0,1,0,NCL=1,TYPE=A
COMNDEND COMND CP,0,2,0,NCL=1,TYPE=A
```

Length	Field Use
8	Command name
1	CP class(es)
1	No class (NCL) flag
2	Minimum characters
4	Address of execution module

The macro parameter TYPE specifies the type of address constant to be generated; the default is V. The "no class" flag is set by specifying NCL=1. Contrary to popular opinion, this flag does not indicate commands with a low level of cultural acumen. Instead, if it is set to one, the command can be issued before logon is complete. Before you enter your userid and password, it is impossible to determine what your VMCLASS value will be. Hence the name of the flag

is NCL. The following commands have the NCL flag set to 1: LOGON, LOGIN, DIAL, DISCONN, LOGOFF, LOGOUT, MSG, MESSAGE, SLEEP, *, and CP. It is also possible to issue the NCL commands, other than the first three, after logon.

The command table has two different points to begin the command search. The first, label COMNBEG0, is used for executing commands before logon. After logon, the search begins at label COMNBEG1. Therefore the three commands LOGON, LOGIN, and DIAL can be found only when they are issued before logon. The other commands with the NCL flag set to 1 are valid both before and after LOGON.

15.3.3 General command processor logic

Each command has a main processing routine that is called from DMKCFM after the command name has been processed. These processing routines are usually located in pageable modules because they are relatively seldom used. In many cases control has to be passed to additional routines because the command logic is often larger than 1 page.

15.3.3.1 Common subroutines

While the exact logic of each command routine is of course unique, some common functions are needed for many commands, and so several utility subroutines exist in the resident nucleus. The following is a list of some of these routines.

1. DMKCVTHB converts an EBCDIC string of hexadecimal characters to their equivalent binary value.

2. DMKERMSG writes an error message in the standard format and optionally returns to its caller's caller (usually DMKCFM).

3. DMKSCNFD scans for the next BUFFER data field and returns its address and length.

4. DMKSCNAU scans the circular chain of VMBLOKs, looking for one belonging to the specified userid.

5. DMKSCNRU returns with the addresses of the RDEVBLOK, RCUBLOK, and RCHBLOK for the specified real I/O device.

6. DMKSCNVU returns with the addresses of the VDEVBLOK, VCUBLOK, and VCHBLOK for the specified virtual I/O device.

7. DMKQCNWT writes a line of output to the user's terminal.

15.3.3.2 Special scanning for QUERY and SET

QUERY and SET are unusual commands in that they have parameters that are dependent upon privilege classes. Appropriate checks cannot be made in DMKCFC because that routine checks only the command name. Both commands therefore have an initial processing routine, DMKCFJ, that scans off the first parameter and performs the class validity checking before invoking the real processing routines. DMKCFM calls entry point DMKCFJQU for QUERY and DMKCFJSE for the SET command. Let us look more closely at SET processing. We will assume that the command is "SET DUMP AUTO". DMKCFJ has a table of SET parameters, containing for each its name, minimum abbreviation, privilege class, module index, and a branch index (used for further branching within the next processing module). Each entry is 12 bytes long. Table 23 shows both the parameter table entry for "SET" and a module address table with the linkage code.

TABLE 23

Table of SET parameters

```
          DC     CL8'DUMP    ',AL1(4,B,8,4)
                      |              | | | |
                      |              | | | |
     Parameter -------+              | | | |
     Minimum parameter length ------+ | | |
     Permitted privilege class(es) --+ | |
     Module index (into R7) ------------+ |
     Branch index (into R6) --------------+

          L    R15,SETMODS(R7)  Get our adcon and call
          CALL (15)               the parameter routine.
*                                (There is no return.)
*
SETMODS   DC   A(DMKCFSET)     0 Module Index
          DC   A(DMKCFOEX)     4
          DC   A(DMKCFUEX)     8
          DC   A(DMKJRLSE)    12
```

As with the processing in DMKCFC, DMKCFJ scans the parameter table looking for a valid match. If a match is not found, an error message is generated by a call to DMKERMSG. When a match is found, the module index is used to load into R15 the address of the correct entry point from the table at label SETMODS, and the branch index within the module is loaded into R6. In the case of "SET DUMP", DMKCFUEX will be called with 4 in R6. DMKCFUEX will use the contents of R6 as an index into its own table of processing routines, one of which will then get control to handle the "DUMP" parameter and the additional "AUTO" parameter. DMKCFUEX has no reason to return to DMKCFJ and so upon entry it issues an SVC 16 to delete its SAVEAREA and make current the previous one. When DMKCFUEX is finished, it will EXIT directly back to DMKCFM; this is an attempt to avoid unnecessary use of the pageable module DMKCFJ.

15.3.3.3 Exit from console function mode

The following commands do not return to DMKCFM but instead take the virtual machine out of console function mode, allowing it to resume execution:

1. ATTN simulates an attention I/O interrupt from the virtual machine console followed by the START key.

2. BEGIN simulates the System/370 console START key.

3. EXTERNAL simulates the System/370 INTERRUPT key followed by the START key.

4. IPL simulates the System/370 LOAD key.

5. RESTART simulates the System/370 RESTART key.

Console function mode will also be ended by a second "PA1" key depression from a 3270 terminal.

15.4 SELECTED COMMAND LOGIC

To further illustrate CP command processing, we will give below some examples of typical commands. For all commands, CP must decode the command string. In some cases, the effect of the command is just the setting or displaying of some memory contents. In other cases, complex processing is necessary to implement the requested function.

15.4.1 ADSTOP command

ADSTOP is the console function command that performs the address comparison operation. The corresponding COMND table entry in DMKCFC points to DMKCFDAD as the processing routine. DMKCFDAD calls DMKSCNFD to get the address parameter, calls DMKCVTHB to convert it to binary, and compares it to VMSIZE in the VMBLOK to check that the address is within the virtual machine's memory.

In order to let the CPU run at full speed, ADSTOP is implemented by storing an SVC X'B3' instruction at the designated memory address. DMKCFDAD uses the TRANS macro to bring the virtual page into real memory and saves the targetted instruction, which it then replaces with the SVC X'B3'. DMKCFDAD then EXITs back to DMKCFM. At some later time, the virtual machine will be taken out of console function mode and will begin normal execution.

When the SVC is executed, DMKSVC restores the original instruction, issues a message "ADSTOP AT ..." via DMKQCNWT, and calls DMKCFM to place the virtual machine in console function mode (that is, it presses the virtual "STOP" key).

15.4.2 INDICATE command

The INDICATE command provides a virtual "system activity meter" to show how the virtual machine or the real machine is performing. Routine DMKTHIEN is called by DMKCFM to process the command ("THI" was the original name of this command: "temperature and humidity index"). DMKTHIEN calls DMKSCNFD to get the parameters, if any, and checks each one for validity using its own table. It obtains a work area via DMKFREE and builds in it a message containing the system or user performance data. The information is gotten from various fields in the PSA and in DMKSCH and is converted to character format by DMKCVTBD. DMKQCNWT is called to display the message and FRET the work area. DMKTHIEN then EXITs back to DMKCFM and the command is complete.

15.4.3 DEFINE command

The DEFINE command is the implementation of the "virtual Customer Engineer" and has DMKDEFIN as its main processing routine; the logic for DEFINE STORAGE and DEFINE CHANNEL has been split off into module DMKDEG. Let us take as an example "DEFINE GRAF 123". DMKDEFIN calls DMKSCNFD for the first parameter and checks "GRAF" against an internal table of valid parameters. It then calls DMKSCNFD again for the

second parameter and then calls DMKCVTHB to convert the address to binary. After all the parameters have been scanned and checked, DMKDEFIN calls DMKVDSDF to perform the actual device definition.

DMKVDSDF calls DMKFREE to obtain storage, if necessary, for new sets of VxxxBLOKs (remember that the blocks must be contiguous for each of the 3 types). It then stores into the new VDEVBLOK all the appropriate flag and status bits to show that this is a virtual 3270 device for which there is not yet a corresponding real device. DMKVDSDF then EXITs to DMKDEFIN, which builds a verification message in its SAVEWRKn area, calls DMKQCNWT to write the message, and then EXITs to DMKCFM. The command is now complete.

15.5 PROGRAMMING CONSIDERATIONS

There are several programming considerations that you should keep in mind if you want to write a command processing routine or modify an existing one.

1. The VxxxBLOKs can move across calls that define new virtual devices or that can lose control in such a way that a device can be detached in the meantime. The standard convention is to subtract the origins of the VxxxBLOKs from R6, R7, and R8 before such a call and to then add the origins again afterwards. Many IBM APARs contain this code because this is a very subtle trap that has caught IBM several times.

2. While you are processing the command BUFFER, you must be careful not to change R9, its base address. Even if your routine does not explicitly use the BUFFER, some subroutines might (such as DMKSCNFD).

3. Many of the command processing routines are nearly full. IBM often has to split the modules as a result of adding new function or applying APARs. If you have code in such a module, then you must examine each PUT tape to see if your code has to be moved to a new module. This is of course a potential problem in all of CP, but the problem is perhaps more severe in the area of command processing routines.

NOTES

Chapter 16

DIAGNOSE INTERFACE

16.1 INTRODUCTION

16.1.1 Overview

The DIAGNOSE instruction, operation code X'83', is a privileged instruction. It is reserved for System/370 model-dependent diagnostic operations. For example, to validate error checking circuits, a DIAGNOSE operation can force a bad Error Correction Code (ECC) in main storage.

The instruction is four bytes long and has a layout similar to the LOAD MULTIPLE (LM) instruction. In System/370, the standard way for a user process to invoke a supervisor service is to issue a Supervisor Call (SVC). Because the operating system in the virtual machine may have all SVC numbers assigned, another method had to be found for virtual machines to communicate with CP. The DIAGNOSE instruction was chosen. Internally it is also know as the Hypervisor Call. When DIAGNOSE is issued from a virtual machine, a privileged operation exception is generated. The DIAGNOSE interface is used to execute CP commands and to manipulate the architectural extensions available to the virtual machine.

16.1.2 References

16.1.2.1 Publications

1. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

16.1.2.2 CP modules

1. DMKHVC - is the initial processing routine for DIAGNOSE instructions.

2. DMKHVD - contains additional pageable DIAGNOSE processing routines.

3. DMKHVE - contains additional pageable DIAGNOSE processing routines.

16.2 INSTRUCTION FORMAT

The DIAGNOSE instruction is specified in a user program as one byte of operation code (X'83'), one byte for the two register numbers, and finally two bytes containing the hexadecimal code of the function to be performed. Codes from X'00' through X'FC' are reserved for IBM use. Codes from X'100' through X'1FC' are available to installations for local uses.

The two specified registers are designated X and Y for descriptive purposes. If more that two registers are needed, up to two more registers can be designated as registers 'X+1' and 'Y+1'. In this case, neither 'X' nor 'Y' is usually allowed to be R15.

16.3 RETURN CONDITIONS

In addition to the four instruction condition codes, a program can receive a program interrupt with an operation, specification, or other exception code to indicate an error condition. The System Programmer's Guide contains a description of the uses and error conditions for each DIAGNOSE code.

16.4 COMMON DIAGNOSE CODES

Table 24 lists some of the more frequently-used DIAGNOSE codes and their privilege classes. Also notice that to successfully execute the DIAGNOSE X'4C', the virtual machine must have the ACCOUNT option specified in its directory entry.

TABLE 24

Common DIAGNOSE codes

Code	Class(es)	Description
04	C+E	examine real storage.
08	G	execute a console function.
0C	G	virtual clock.
14	G	spool file manipulation.
18	G	DASD I/O.
20	G	general I/O.
3C	A+B+C	dynamic directory swap.
4C	(ACCOUNT)	punch accounting card.
58	G	3270 full screen interface.

16.5 EXAMPLE DIAGNOSE

As an exercise, let us look at one DIAGNOSE code closely. It will serve as the model for other DIAGNOSE processing. DIAGNOSE X'08' is used by CMS and user programs to execute CP commands. It is specified in the form given in Figure 28.

83 is the operation code.
x is the first parameter register.
y is the second parameter register.
0008 is the DIAGNOSE function code.

Figure 28: The DIAGNOSE instruction

The X register contains the address of the command line to be executed. Normally the response lines for the command

are printed on the virtual console. The first byte in register Y is a flag byte. The other three bytes of register Y contain the length of the command line. There are two flag values currently defined. The first flag value, X'80', indicates that the printing of any needed password be suppressed. The X'40' flag value specifies that the user wants the command output to be placed in a buffer in the user's virtual memory. This facility allows programs to examine the output generated by commands they issue.

If the X'40' flag is used, then two additional registers must be given. They are designated the 'X+1' and 'Y+1' registers. Register 'X+1' contains the address of the response buffer supplied by the user and register 'Y+1' contains the length of this buffer.

Normal completion of this DIAGNOSE, as with all others, ends in a condition code of 0 being set. A condition code of 1 indicates that the user-supplied buffer was insufficiently large to hold all of the output from the command. Register 'Y+1' is returned with the number of bytes truncated from the response. A recovery procedure in this circumstance is to calculate the size of a larger buffer and reissue the DIAGNOSE. If the length of the command line is specified as zero, the user's virtual console is placed in CP mode. If the length of the response buffer is specified as a number outside of the range from 1 to 8192, a specification exception is generated. Other error conditions are handled in analogous ways. Figure 29 shows a typical user invocation of DIAGNOSE code X'08'.

```
         LA    R2,CMD          Load X register.
         L     R4,PARM         Load Y register.
         LA    R3,RESBUFF      Load X+1 register.
         LA    R5,LRESBUFF     Load Y+1 register.
*
         DC    X'83240008'     Issue DIAG R2,R4,8.
         BZ    OKCONT          Ok, continue normally.
*   Handle error conditions here.
         B     SOMEWHER
*
CMD      DC    C'Q RDR ALL ' CP command
LCMD     EQU   *-CMD             and its length.
*
RESBUFF  DC    CL80' '         Output buffer
LRESBUFF EQU   *-RESBUFF         and its length.
*
         DS    0F
PARM     DC    X'40'           Flag byte and
         DC    AL3(LCMD)         command length.
```

Figure 29: Using DIAGNOSE 8

16.6 DIAGNOSE PROCESSING

Because operation code X'83' is a privileged operation, DMKPRGIN goes to DMKPRVLG. DMKPRV determines that the operation code is a X'83' and goes to DMKHVCAL (HyperVisor Call). Finally DMKHVC uses the DIAGNOSE code in the instruction to index into a branch table to pass control to the module or internal subroutine that will process the DIAGNOSE request.

In the case of DIAGNOSE X'08', a subroutine labelled HVCONFN gets control. It checks the parameters for validity and then invokes DMKCFMEN, the normal console command processor. Upon return, DMKHVC goes to DMKDSPCH. Other DIAGNOSE codes function in a similar manner.

16.7 SUMMARY

The DIAGNOSE interface is the primary way in which architectural extensions to the System 370 virtual machine are invoked. Codes X'00' to X'FC' are reserved for IBM's use. Codes X'100' to X'1FC' are reserved for installation use.

NOTES

Chapter 17

MULTIPROCESSOR SUPPORT

17.1 INTRODUCTION

In Release 4 of VM/370, IBM distributed support for Attached Processor (AP) configurations. With VM/SP, IBM added support for MultiProcessor (MP) systems and also improved the AP support in response to marketing opportunities and to enhance the performance of VM on the engines made in Poughkeepsie.

An AP configuration has 2 processors sharing main memory but only one processor is allowed to perform I/O operations. An MP configuration is similar except that both processors are able to perform I/O operations. The limit of two processors is a restriction of the software implementation within CP and is not a restriction imposed by the hardware architecture.

17.1.1 Overview

This chapter covers the hardware features available in System/370 processors to support multiprocessing and discusses the manner in which CP uses those hardware features to increase system throughput. Implications for user modifications are discussed, including examples of how to adequately protect data structures from simultaneous update in a MP system.

17.1.2 References

17.1.2.1 Publications

1. Brown, M. P. et al, *VM/SP MP and Enhanced AP Support*, IBM Washington Systems Center Technical Bulletin, GG22-9212, December 1980.

2. Enichen, M. C., and D. R. Patterson, *VM/SP: Introduction to Multiprocessing Concepts*, IBM Washington Systems Center Technical Bulletin, GG22-9247, September 1981.

3. *IBM System/370 Principles of Operation*, GA22-7000.

17.1.2.2 CP modules

1. DMKAPI - handles initialization functions specific to systems with more than one processor.

2. DMKCLK - handles synchronization of the TOD clock for MP configurations with more than one TOD clock.

3. DMKCPU - handles VARY commands when the device specified is a processor rather than an I/O device.

4. DMKDSP - processes TRQBLOKs, IOBLOKs, and CPEXBLOKs in relationship to the locks that must be held. Contains an entry point such that if the system lock is not held, virtual machines can still be dispatched.

5. DMKEXT - handles processing for the SIGNAL macro instruction. Issues the SIGP instruction to the other processor and handles the external interrupt resulting from another processor performing a SIGP.

6. DMKIOS - allowseither processor to queue an I/O request for any path. Processors are switched only when the I/O can be started on a path that the other processor owns.

7. DMKLOK - handles requests for MP locks that cannot be satisfied immediately. Centralizes handling of both spin and defer locks.

8. DMKSVC - handles SVC 24 to SWITCH processors. Also, handles the chain of SVC save areas maintained for each processor.

9. DMKMC* - this family of modules handles various pathological conditions that arise in a multiprocessor situation.

17.2 370 MP/AP ARCHITECTURE REVIEW

This section is the obligatory review of the hardware characteristics on System/370 processors that are specific to multiprocessing systems. For more details on the hardware support, refer to the *System/370 Principles of Operation* manual.

17.2.1 Prefix register

The prefix register in multiprocessor and attached processor systems acts as a "magic mirror" on certain addresses. When bits 0-11 of a real address formed by a processor are all zero, bits 8-19 of the prefix register are substituted. The result of replacing the 0-11 zero bits of a real address with the contents of the prefix register bits 8-19 is called an absolute address. Conversely, when bits 0-11 of a real address match bits 8-19 in the prefix register, the absolute address is formed by replacing bits 0-11 of the real address with zeros. Note that addresses formed by the channels when fetching the Channel Address Word and storing the interrupt address and Channel Status Word are real addresses and are therefore relocated by the prefix register. However, the addresses within channel programs are absolute and not affected by the contents of the prefix register.

17.2.2 Instructions

17.2.2.1 Instructions for shared memory

The instructions used for serializing accesses to memory are Compare and Swap (CS), and Compare Double and Swap (CDS). By using these two instructions it is possible to perform some kinds of operations on counters, linked lists, etc. without the necessity of defining a lock for a resource. Also, it is possible to define and implement a resource lock mechanism using these instructions, as is discussed later.

17.2.2.2 Signal Processor

The Signal Processor instruction (SIGP) has a number of defined order codes used to implement communications between processors. Table 25 lists the subset of the defined SIGP order codes that are used by VM/SP. Note that only the first 3 order codes are meant to be used frequently; use of the other order codes is limited to situations where a performance problem induced by their use is not a high priority issue.

The emergency signal and external call functions cause the addressed processor to have an external interrupt pending. Additionally, any processor that fails and enters the "check-stop" state causes all other processors to have a malfunction alert external interrupt pending. These external interrupts have specific interrupt codes and also have assigned mask bits in CR0.

TABLE 25

SIGP order codes used by VM/SP

Code	*Meaning*
1	Sense
2	External call
3	Emergency signal
4	Start
5	Stop
6	Restart
7	Initial Program Reset
9	Stop & Store Status

Other than the separate mask bits and interrupt codes, the difference between an emergency signal and an external call is that a processor can have only one external call condition pending but a processor can keep pending an emergency signal condition from each CPU of a multiprocessor system (including itself).

17.3 CP ARCHITECTURE FOR MP/AP

17.3.1 Storage and data structures

17.3.1.1 Layout of main memory

Figure 30 shows the layout of main memory following a normal CP initialization. The PSAs of the processors may be in the dynamic paging area if the VARY PROC command is used to change the operation from UP-mode to MP-mode without an intervening IPL.

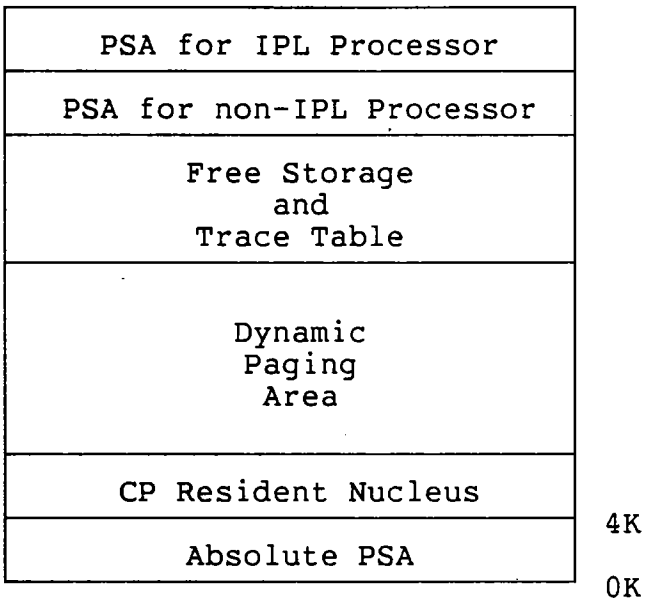


Figure 30: Layout of MP/AP memory

17.3.1.2 Lock words

In general, any 4-byte location in memory can be defined to be a lock for some resource. Certain CP-defined locks have a small data structure associated with them to aid in debugging and performance investigations. For these CP-defined locks, there is not only the lock itself, but also R12 at the time the lock was obtained (this should be the base address of the CP module that obtained the lock), the amount of time spent spinning while trying to obtain the lock (in microseconds), and the number of times the lock was requested and not available.

17.3.1.3 Save areas

Each processor maintains its own singly-linked list of SVC save areas anchored in its PSA. Compare and Swap is used to add entries to the list, and Compare Double and Swap is used to remove list entries along with incrementing a counter for the number of entries deleted. (Why CDS is used in this case and not CS is an interesting exercise for the reader). References to a processor's private save area chain have to

be protected because if one processor runs out of SVC save areas, it steals save areas from the other processor before becoming desperate enough to call DMKFREE. In fact, while the processor is stealing one save area it steals two (they're cheap). This leads to a somewhat humorous picture of the two processors stealing save areas back and forth from each other until there are none left and someone has to call DMKFREE to carve a save area from CP free storage.

17.3.1.4 Shared segments

Because there is no read-only bit defined in the 370 architecture for the page tables, CP must maintain a duplicate set of shared segments and pages for each processor. Whenever a virtual machine is being dispatched by a processor or when a processor may access the virtual machine's memory, a check must be performed that the virtual machine's segment table points to the right set of shared pages for that processor. Normally this check is performed by the SWTCHVM macro (discussed in more detail later). Note that the HPO version of VM/SP does in fact support the segment protection facility that is available on some System/370 processors. In this case, only one copy of each shared segment is maintained for the entire system.

17.3.1.5 PSA

Some values maintained in the PSA of a uniprocessor system are maintained in the absolute PSA of a multiprocessor system. Examples of these fields are: CPUID, trace table pointers, paging rate, pointers to monitor buffers, count of page and swap tables in the system, etc. When the PSA MACRO expands in an assembly listing, fields in the absolute PSA have a comment to that effect.

All other PSA fields are either duplicated between the processors (have the same value) or are maintained separately by each processor (like the chain anchor for the SVC save area parameter list). Table 26 lists some of the PSA fields that are interesting from the multiprocessing point of view.

TABLE 26

MP-related fields in the real PSAs

APSTAT1-5	AP/MP status flags
EMSPEND	Emergency signals pending
EMSREC	Emergency signals received
XCPEND	External calls pending
LPUADDR	Logical address of processor
LPUADDRX	Logical address of other proc
PREFIXA	Prefix reg contents
PREFIXB	Prefix reg contents-other proc
TIMEDISP	Disp. to overhead in VMBLOK
TRACPROC	Value to OR with trace code

17.3.2 Signaling

17.3.2.1 Emergency signal

Table 27 lists the function codes and their meanings defined within CP. Whenever the SIGNAL macro is used to issue an emergency signal request to the other processor, the program does not resume until the other processor has received the external interrupt and started processing the request.

TABLE 27

SIGP - emergency signal function codes

Fcn Code	*Meaning*
X'8000'	Quiesce
X'4000'	Extend
X'2000'	TOD clock sync
X'1000'	Shutdown
X'0800'	TOD clock check
X'0400'	Extend exit

17.3.2.2 External call

Table 28 lists the function codes and assigned meanings when the SIGNAL macro is used to issue an external call to the other processor.

TABLE 28

SIGP - function codes for external call

Fcn Code	*Meaning*
X'8000'	Alternate Proc. Recovery
X'4000'	Resume
X'2000'	Wakeup
X'1000'	Dispatch

17.3.3 Locking structure

In order to protect data structures from simultaneous update, certain functions and data areas should not be updated unless the processor has first obtained a specific "lock". Owning the lock insures that only one processor at a time is updating or examining fields. Note that there is nothing magical about this protection and nothing within CP forces that a particular lock be held at a particular time (except that requesting a lock the processor already holds results in a LOK001 ABEND of the system).

17.3.3.1 Spin locks

One kind of system lock protects resources that are only used for short periods of time, such as a queue that is being updated. Since the lock is held for short periods of time, it doesn't make sense for the other processor to perform some complicated procedure for deferring the request; instead, the other processor loops until the lock is available (termed "spinning on the lock"). These "spin locks" are used to serialize processing within the first level interrupt handlers so that interrupts can be processed without requiring a global lock.

17.3.3.2 Defer locks

Certain resources may require protection for longer periods of time; for these types of fields there is defined a type of lock that defers processing until the lock is available. These "defer locks" cause the current status of the processor to be saved in a CPEXBLOK when the lock is not available. Control returns via the standard CPEXBLOK unstack mechanism when the lock is available.

17.3.3.3 Private locks

Any CP system programmer may define additional locks. If the lock word is defined within DMKLOK, the whole lock data structure of four fullwords (as discussed above) must be provided. If the lock word is not in DMKLOK, then only one fullword (suitable for use by a CS instruction) is required. These "private locks", as they are called, are all spin locks so they should only be held for very short periods of time. It is the programmer's responsibility to explicitly release any lock that is obtained.

17.3.3.4 CP-defined locks

TABLE 29

Defined locks within CP

Name	*Type*	*Comments*
SYS	DEFER	Global system
VMBLOK	DEFER	
RL	SPIN	Run list
TR	SPIN	CKC request queue
DS	SPIN	Dispatcher queues
IO	SPIN	I/O subsystem
RM	SPIN	Real mem/paging
RDEVBLOK	SPIN (Private)	IOBLOK queue
FREE	SPIN	Free storage

Table 29 lists the locks defined by IBM to protect various resources within CP. Although all the documentation would lead one to believe there is something special about some of the locks, there is really no difference in the implementation of how the locks are treated (except the VMBLOK lock). In particular, there is no difference between the global system lock (SYS) and the other spin locks except that only the dispatcher actually spins waiting for that lock to be available. For a more details, see the section on the LOCK macro, below.

The CP-defined locks are used to serialize access to the following resources and functions:

1. SYS - All execution under IOBLOKs, TRQBLOKs and CPEXBLOKs is done with the system lock held. Also, execution of pageable CP modules, and selection of pages for stealing require ownership of the global system lock. If any lock other than a VMBLOK lock must be held along with the system lock, the system lock must be obtained first to avoid deadlock.

2. VMBLOK - The VMBLOK lock is at the field labelled VMLOCK within the VMBLOK, and the field VMLOCKER contains the contents of R12 at the time the lock was obtained. The VMBLOK lock of a virtual machine is held while the virtual machine is dispatched on a processor and is used to serialize updates to the VMBLOK itself. If the VMBLOK lock cannot be obtained, a deferred execution CPEXBLOK is stacked.

3. RL - The run list lock serializes access to the run list by the dispatcher and the scheduler. Use of this lock allows virtual machines to be dispatched without having to hold the global system lock.

4. TR - The TRQBLOK list lock serializes access to the clock comparator request queue. This lock allows serialization between the routines in the scheduler and the external FLIH that can remove the top entry from the queue without holding the global system lock.

5. DS - The dispatcher queue lock serializes access to the CPEXBLOK, IOBLOK, and TRQBLOK queues. This lock allows serialization between the dispatcher and routines that do not hold the global system lock but call DMKSTK to add the deferred execution blocks to the appropriate queue.

6. IO - The I/O lock serializes I/O initiation and interrupt processing (DMKIOS and DMKIOT). Also, access to certain fields in the real I/O control blocks is serialized; the complete list of fields is contained in Reference 1.

7. RM - The real storage management lock serializes access to many of the lists and queues within the paging subsystem. This lock allows much of the paging subsystem to execute without requiring the ownership of the global system lock.

8. RDEVBLOK - This lock is contained within each RDEVBLOK and serializes access to the IOBLOK queue (at RDEVFPNT). This lock allows serialization of access to queued I/O requests by a processor that holds the IO lock or the RM lock and allows the requests to be manipulated by either the paging subsystem or the I/O subsystem.

9. FREE - The free storage lock is obtained and released within DMKFRE to serialize access to the free storage chains. This lock allows serialization of free storage management for any function within CP that needs to acquire or release free storage without requiring the ownership of any particular lock.

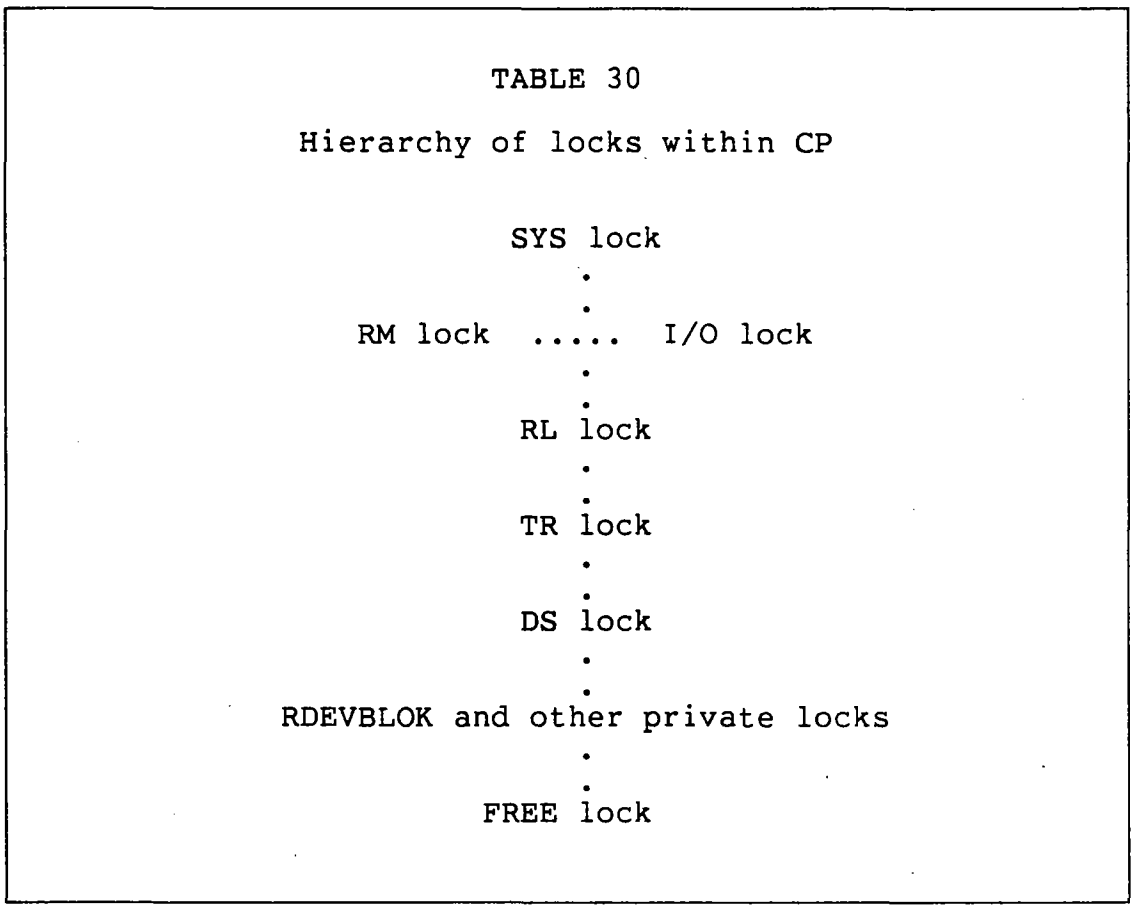
17.3.3.5 Lock hierarchy

Any system of locks is susceptible to deadlock situations unless certain precautions are taken. One well-known way of preventing deadlock is to insure that locks are always requested in a certain order.

Table 30 gives the hierarchy of the currently defined locks within CP. Note that there have been no situations in the CP code where the RM lock and the I/O lock need to be held at the same time that weren't eliminated by the use of the RDEVBLOK lock. Therefore, no ordering is defined between these two locks. For the other locks, if a processor owns a lock, it may only request locks lower in the hierarchy and not higher. Note that this hierarchy is NOT enforced in any consistent fashion. It is primarily the responsibility of the programmer to insure that the hierarchy is obeyed.

TABLE 30

Hierarchy of locks within CP



17.3.4 CP macros for MP/AP support

This section describes some of the macros used within CP in support of the multiprocessor code. While other macros are affected by MP support, these are the macros encountered most frequently.

17.3.4.1 COUNT

The COUNT macro is an excellent tool for demonstrating the use of Compare and Swap. Figure 31 contains a small code segment that at one time demonstrates the use of CS to update a value without a lock and also is an example of the code generated by the COUNT macro when assembled for a MP configuration.

```
          L     R14,COUNTER    Get counter
TRYAGAIN  DS    0H
          LR    R1,R14         Copy original value
          A     R1,F1          Bump the counter
          CS    R14,R1,COUNTER    Try to store it
          BNE   TRYAGAIN       -> No good, again
```

Figure 31: Example of COUNT macro and CS

17.3.4.2 TRACE

The TRACE macro handles obtaining a trace table slot on both UP and MP systems. The code generated for UP systems handles the check for whether the trace table should wrap and places the trace code in the first byte of the entry. On an MP system, the trace table slot must be allocated with CS since the processors share the table pointers that are maintained in the absolute PSA. Additionally, the TRACPROC field from the real PSA is "or'ed" with the trace code to identify the processor generating that particular entry.

17.3.4.3 SWITCH

The SWITCH macro provides a way to have execution proceed on a particular processor (or the other processor). The macro generates an SVC 24 that causes the SVC interrupt handler to save register and execution address in a special CPEXBLOK that is stacked LIFO on the dispatcher's queue.

The macro can take an operand that supplies the logical processor address either in a register or a half-word storage location. Note that use of the macro without an operand implies that a switch should be made to the I/O processor, which only has meaning in AP mode.

A similar way of performing the same function is to prepare a CPEXBLOK and call either DMKSTKOP (to stack it for the Other Processor) or DMKSTKMP (to stack it for My Processor). DMKSTK puts the logical processor address in the CPEXBLOK to identify which processor may unstack it. If one processor, when going through the dispatcher, finds a CPEXBLOK for the other processor that is currently idle, then the active processor issues a SIGNAL DISPATCH to the idle processor.

17.3.4.4 CHARGE

```
                START
CHARGE          STOP
                SWITCH,operand
                SYNC
```

Figure 32: Possible operands for the CHARGE macro

As mentioned in the chapter on timer support, the CHARGE macro is used to help maintain correspondence between the contents of R11 and the CPU timer value. VMBLOKs contain two fields, VMCPTIME and VMAPTIME, that are used to record the overhead time attributed to each processor. In the PSA of each processor, the field TIMEDISP gives the offset into the VMBLOK of the field to be used. VMCPTIME and VMAPTIME are both initialized to X'3FFFFFFFFFFF000'.

1. START - the CPU timer is loaded from the offset in the VMBLOK indicated by TIMEDISP.

2. STOP - the CPU timer is stored into the offset in the VMBLOK indicated by TIMEDISP.

3. SWITCH - the charging is STOPped for the current VMBLOK, R11 is loaded from the location indicated by the operand, and charging for the new VMBLOK is STARTed.

4. SYNC - the values of VMCPTIME and VMAPTIME are added together and placed in VMTTIME (really!).

17.3.4.5 LOCK

The LOCK macro is used to request ownership of one of the CP-defined locks discussed above.

1. The first operand is positional, required, and indicates the type of operation to be performed.

```
LOCK RELEASE,TYPE=type,SPIN=YES,SAVE
     OBTAIN                NO
```

Figure 33: LOCK macro operands

2. The second operand is keyword, required, and indicates the name of the lock to be obtained or released. The valid names for "type" are the same as have been discussed before except that the free storage lock cannot be specified, and the RDEVBLOK lock should be specified as TYPE=PRIVATE.

3. If TYPE=PRIVATE, then register 1 is assumed to point to the lock word when the macro is invoked.

4. If TYPE=VMBLOK, then register 1 must contain the address of the VMBLOK to be locked. After obtaining the lock, the lock manager calls DMKVMASW to insure that the segment table for the virtual machine points to the proper set of shared segments for this processor.

5. The SPIN keyword is optional, defaults to YES, and indicates whether or not the processor is to loop until the lock is available.

6. If both OBTAIN and SPIN=NO are specified, then the results of the lock request are variable. If the lock was obtained, condition code 0 is set. If the lock could not be obtained because it was owned by the other processor, a condition code of 1 is set. If the lock could not be obtained because it was already owned by the current processor, you will soon be looking at the dump caused by a LOK001 ABEND.

7. The keyword SAVE is optional and specifies that the programmer really cares about the contents of registers 0, 1, 14, and 15, so they should be saved and then restored before exit from the macro expansion. The registers are saved in a 4-word save area named LOCKSAVE defined within the PSA of each processor.

If the lock cannot be obtained, or if the request is for a VMBLOK lock and the shared segments must be straightened out, control is transfered to various routines within the

lock manager, DMKLOK. Note that the lock manager performs no particular special processing for the deferral of requests for the global system lock.

17.3.4.6 SWTCHVM

The SWTCHVM macro provides a convenient interface to the routine DMKLOKSW to handle the details of switching to a different VMBLOK in a multiprocessor environment. Normal processing when the SWTCHVM is executed proceeds as follows:

1. Obtain the lock for the VMBLOK addressed in R1. If the lock cannot be obtained, stack a deferred execution CPEXBLOK and exit to the Dispatcher.

2. Update the shared segments, if necessary, to be the segments used by the current processor.

3. Execute a CHARGE SWITCH to update R11 and the CPU timer fields.

4. Release the lock on the VMBLOK pointed to by R11 on entry to the routine.

5. Return to the caller.

Various options of the SWTCHVM macro allow the programmer to specify that the shared segment pointers need not be updated, that the return should be on the same processor invoking the macro, or that it is not necessary to obtain the lock of the new VMBLOK.

17.4 IMPLICATIONS FOR USER MODIFICATIONS

One serious concern in the acquisition of a MP or AP system is the conversion of user modifications so that they work properly on the new hardware. This section comments on what kinds of mods require changes for MP/AP support and provides some examples based upon techniques already used within CP.

17.4.1 Considerations

17.4.1.1 Does it need protection?

Careful examination and thought is needed to assess the potential impact of race conditions caused by two processors updating a field at the same time. Unfortunately, since

each processor has access to all of memory, the altering of any storage location causes, in theory, a possible problem. In practice, however, the processors tend to "mind their own business" so that a local modification that, for instance, implements a new query-type function quite possibly would not require changes.

17.4.1.2 Is it already protected?

Many parts of CP are already serialized by requiring the global system lock to be held when they are executed. Examples of such areas are:

1. CP command processing.
2. Execution of pageable CP modules.
3. Processing invoked by the unstacking of a TRQBLOK or IOBLOK.
4. Processing invoked by the unstacking of a CPEXBLOK.

In fact, looking carefully at the list of locks, one can observe that the resources that have separate locks are on the high performance interrupt generated paths where one processor locking out the other could seriously degrade system performance.

Data areas within CP that are only accessed while the global system lock is held require no further protection. IMPORTANT: The VMBLOK lock is held while a virtual machine is dispatched and is generally used to serialize updates to that control block and its appendages. In "normal" CP code, the processor owns the lock for the VMBLOK pointed to by R11 but this should be verified and SWTCHVM used before updating fields of another VMBLOK.

17.4.2 And if you have to ...

Of course, in some cases an additional lock must be obtained or access to a particular resource serialized in some fashion. Below are some examples of the techniques used within CP to protect fields from simultaneous update by multiple processors.

17.4.2.1 Switch VMBLOKs

```
         L      R1,OTHERVM       get other VMBLOK
         SWTCHVM ,               over to other vm
                 .
                 .
                 .
         C      R11,SAVER11      do we need to switch
         BE     NOSWT            -> nope, all set
         L      R1,SAVER11       yes, get original VMBLOK
         SWTCHVM ,
NOSWT    EXIT   ,
```

Figure 34: Example of SWTCHVM usage

Figure 34 shows a simple example of switching to another VMBLOK in a module invoked by the SVC 8 form of CALL. Note that the test for avoiding the SWTCHVM if R11 has not been changed is not strictly necessary since the code in DMKLOKSW checks for that condition and simply returns if R1 = R11, but such a test is often performed in any case.

17.4.2.2 Obtain the global system lock

Figure 35 shows an example of obtaining the global system lock and the code necessary to defer the request if the lock is not available. This code is often within a local subroutine of a module that must obtain the lock since writing the same code does get very tedious and there is no macro defined to handle it. There are several observations that can be made about the code:

1. The LOCK macro generates a call to DMKLOKDF if the lock cannot be obtained. Rather than doing much that is very useful, DMKLOKDF counts the number of times the lock was unavailable and returns with a non-zero condition code.

2. The CPEXBLOK pointed to by VMDFTPNT is a permanent appendage of every VMBLOK and is not released by the dispatcher when it is unstacked.

```
        L      R15,=A(DMKLOKSY+2)  get addr of lock
        CLC    LPUADDR,0(R15)  do we own the lock?
        BE     GTLK            -> yes, piece of cake
        LOCK   OBTAIN,TYPE=SYS,SPIN=NO
        BZ     GTLK            -> hard work, but success
        L      R1,VMDFTPNT     get address of defer block
        USING  CPEXBLOK,R1
        STM    R0,R15,CPEXREGS save registers
        LA     R0,GTLK         address of defer routine
        ST     R0,CPEXADD      set return point
        CALL   DMKSTKDE        go stack the CPEXBLOK
        DROP   R1
        GOTO   DMKDSPRU        run a user while waiting
          .
          .
          .
GTLK    DS     0H              Here when lock obtained
```

Figure 35: Example of obtaining the system lock

3. The external names DMKSTKDE and DMKDSPRU must have explicit EXTRN statements within the module. However, DMKLOKDF is addressed with a V-type address constant.

4. DMKDSPRU is the standard exit point if the system lock cannot be obtained. The dispatcher has sufficient protection to be able to dispatch virtual machines even though the global system lock may be held by another processor.

17.4.2.3 Maintain a queue

Figure 36 illustrates the technique for adding an entry to a singly linked list using the Compare and Swap instruction. The technique shown is in fact used by CP to maintain the local SVC save area chain for each processor.

Figure 37 illustrates the technique for removing an entry from a singly linked list. Note that Compare Double and Swap must be used to protect from a pathological but possible case resulting from the other processor removing two entries and replacing the top entry between the time that

```
        USING QENTRY,R1       Map the new queue entry
        L     R0,QHEAD        Get pointer to first entry
QLOOP   DS    0H
        ST    R0,QNEXT        Point to rest of list
        CS    R0,R1,QHEAD     New entry is the first
        BNZ   QLOOP           -> Failed, try again
          ...
QHEAD   DC    A(*-*)          Pointer to first qentry
          ...
QENTRY  DSECT
QNEXT   DS    F
```

Figure 36: Example of adding to a queue with CS

```
        LM    R14,R15,QCNT    Get queue header info
RLOOP   DS    0H
        LTR   R15,R15         Is entry available
        BZ    NOENTRY         -> Nope, recover somehow
        LA    R0,1            Increment value
        L     R1,QNEXT-QENTRY(,R15) Point to 2nd entry
        ALR   R0,R14          Bump delete counter
        CDS   R14,R0,QCNT     Get the first entry
        BNZ   RLOOP           -> Failed, try again
          ...
QCNT    DC    F'0'            Delete counter
QFIRST  DC    A(*-*)          Pointer to first entry
              .
```

Figure 37: Removing a queue entry with CDS

QFIRST is loaded and the CS would be done. While that circumstance seems unlikely, it could happen if one processor was manually stopped or had to field a recoverable machine check in the middle of the code sequence.

17.4.2.4 Set a flag

Figure 38 shows the technique for setting a flag in a word that both processors might try to "maintain" at the same time. Such examples of setting and resetting flags may be found in DMKEXT, which handles SIGP instructions and the external interrupts resulting from a SIGP. Flags for pending external calls and emergency signals must be treated in the fashion shown.

```
         L      R1,FLAG          Get the current value
FLOOP    DS     0H
         LA     R0,MASKEQU       What we want to set
         OR     R0,R1            New value for flag
         CS     R1,R0,FLAG       Stuff it away
         BNZ    FLOOP            -> Oops, not fast enough
           ...
FLAG     DC     F'0'             Flag word (bits 24-31)
MASKEQU  EQU    X'40'            User is a blankity
```

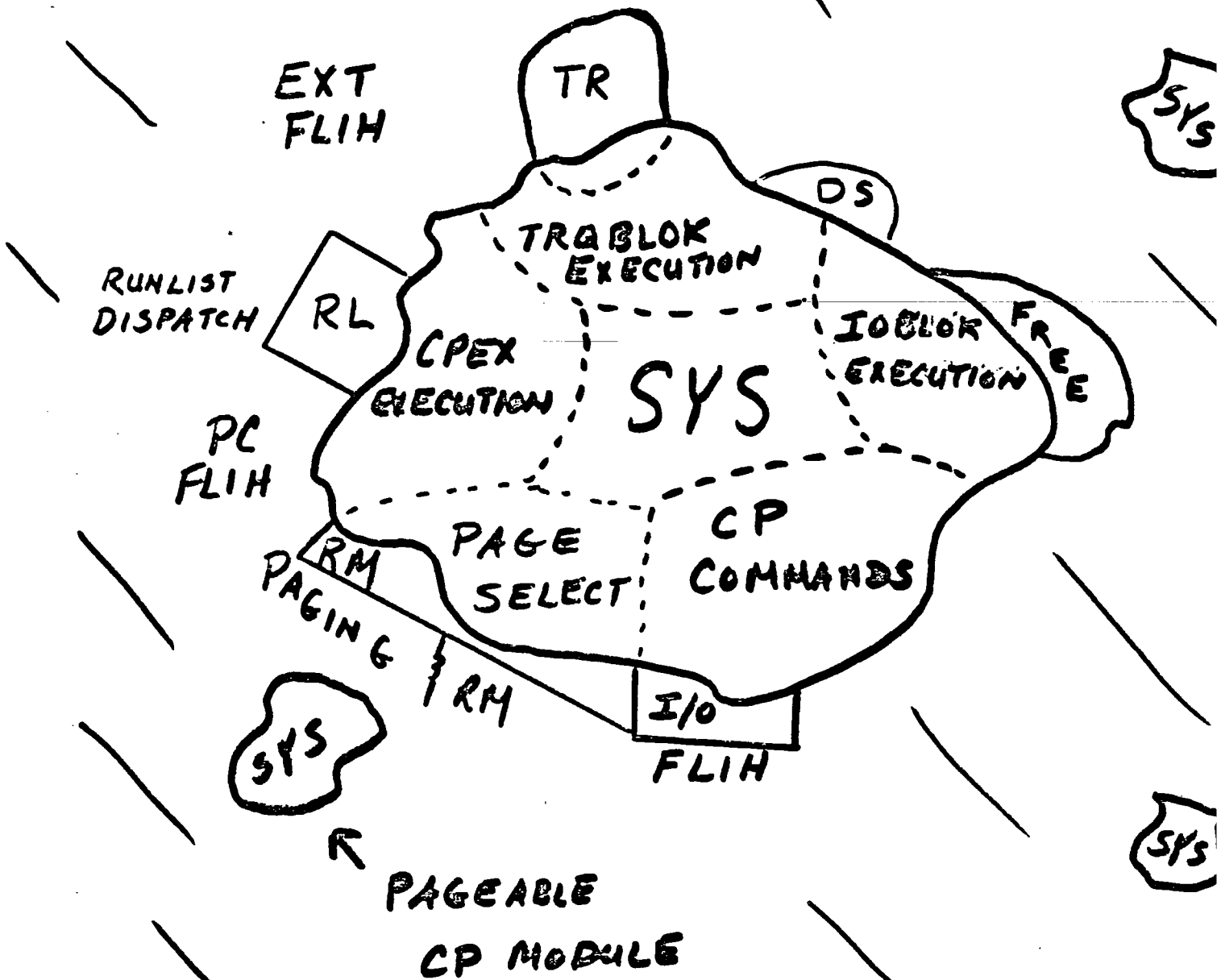
Figure 38: Example of setting a flag with CS

17.4.2.5 Define and use a private lock

Figure 39 shows how a private lock is defined and used. The LOCK macro assumes R1 contains the address of the lock word when TYPE=PRIVATE is coded. The standard basic restrictions apply in that if a error is encountered, CP terminates. Standard errors that a programmer would want to protect against are: obtaining a lock already held (LOK001 ABEND) and releasing a lock that is not yet held (LOK003 ABEND). The RDEVBLOK lock used by the I/O and paging subsystems to serialize access to the queue of I/O requests pending for a device (at RDEVFPNT) is an example of the use of a private lock within standard CP code.

```
         LA    R1,MYLOCK       A(my own special lock)
         LOCK  OBTAIN,TYPE=PRIVATE,SPIN=YES
               .
               .
               .
           (Lots of good high-quality code)
               .
               .
               .
         LA    R1,MYLOCK       Finger the lock again
         LOCK  RELEASE,TYPE=PRIVATE    Give it back
               .
               .
               .
MYLOCK   DC    F'0'            Lock for smc resource
```

Figure 39: Example of private lock

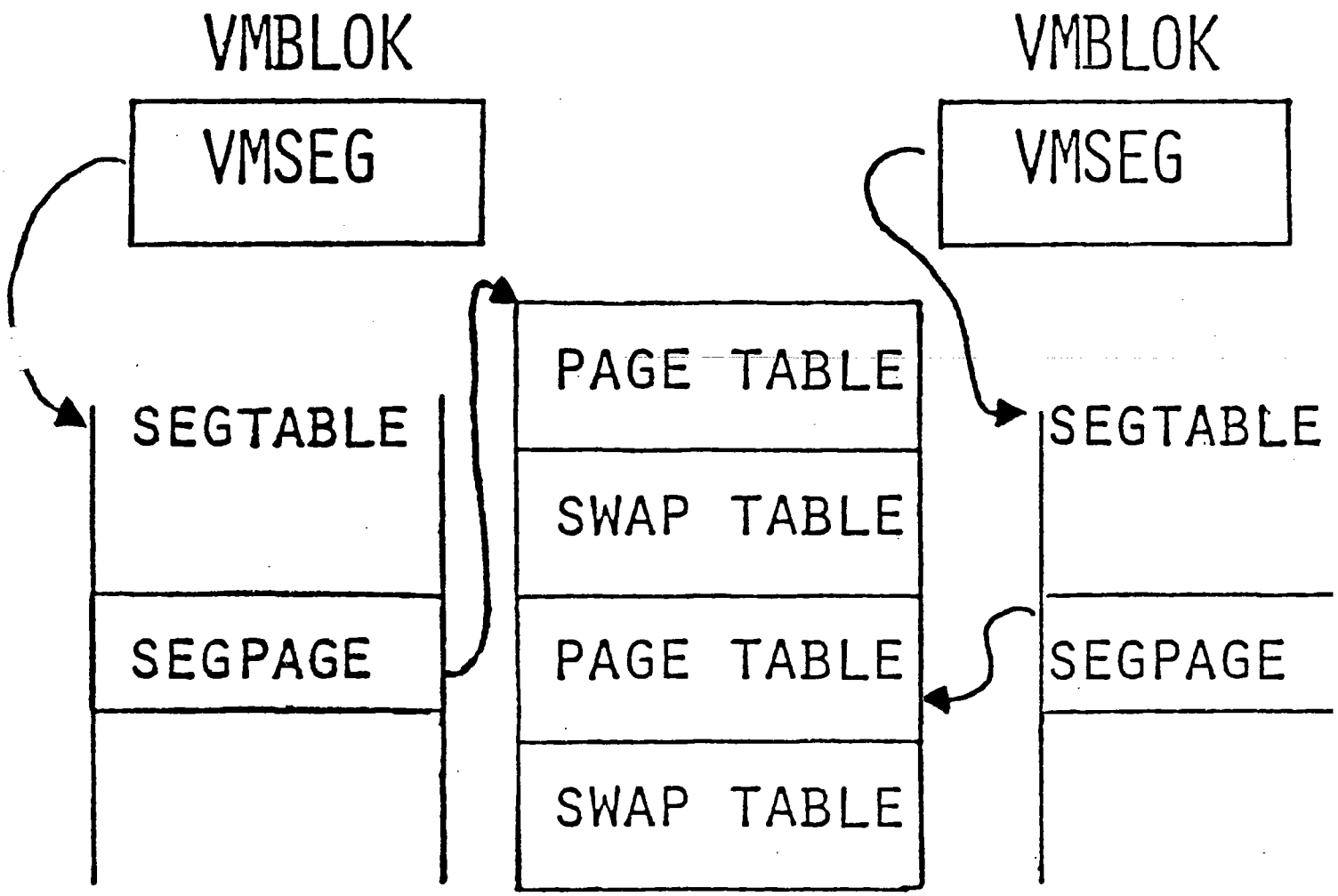

EXT
FLIH
TR
SYS
DS
TRQBLOK
EXECUTION
RUNLIST
DISPATCH
RL
CPEX
EXECUTION
SYS
IOBLOK
EXECUTION
FREE
PC
FLIH
PAGE
SELECT
CP
COMMANDS
RM
PAGING
RM
I/O
FLIH
SYS
SYS
PAGEABLE
CP MODULE

VMBLOK

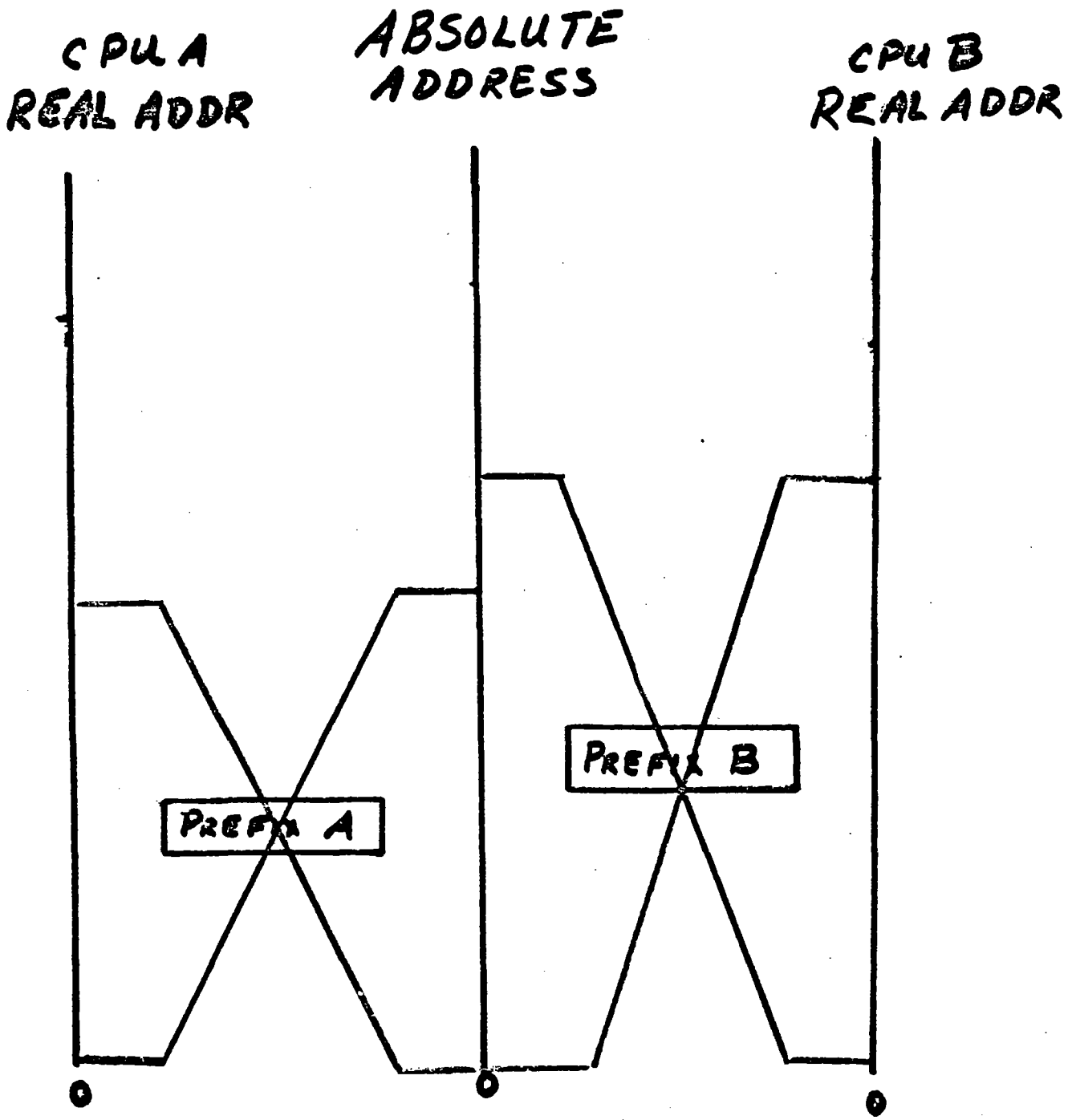
MP/AP

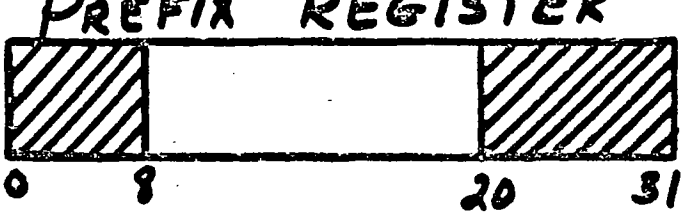
STORAGE & DATA STRUCTURES

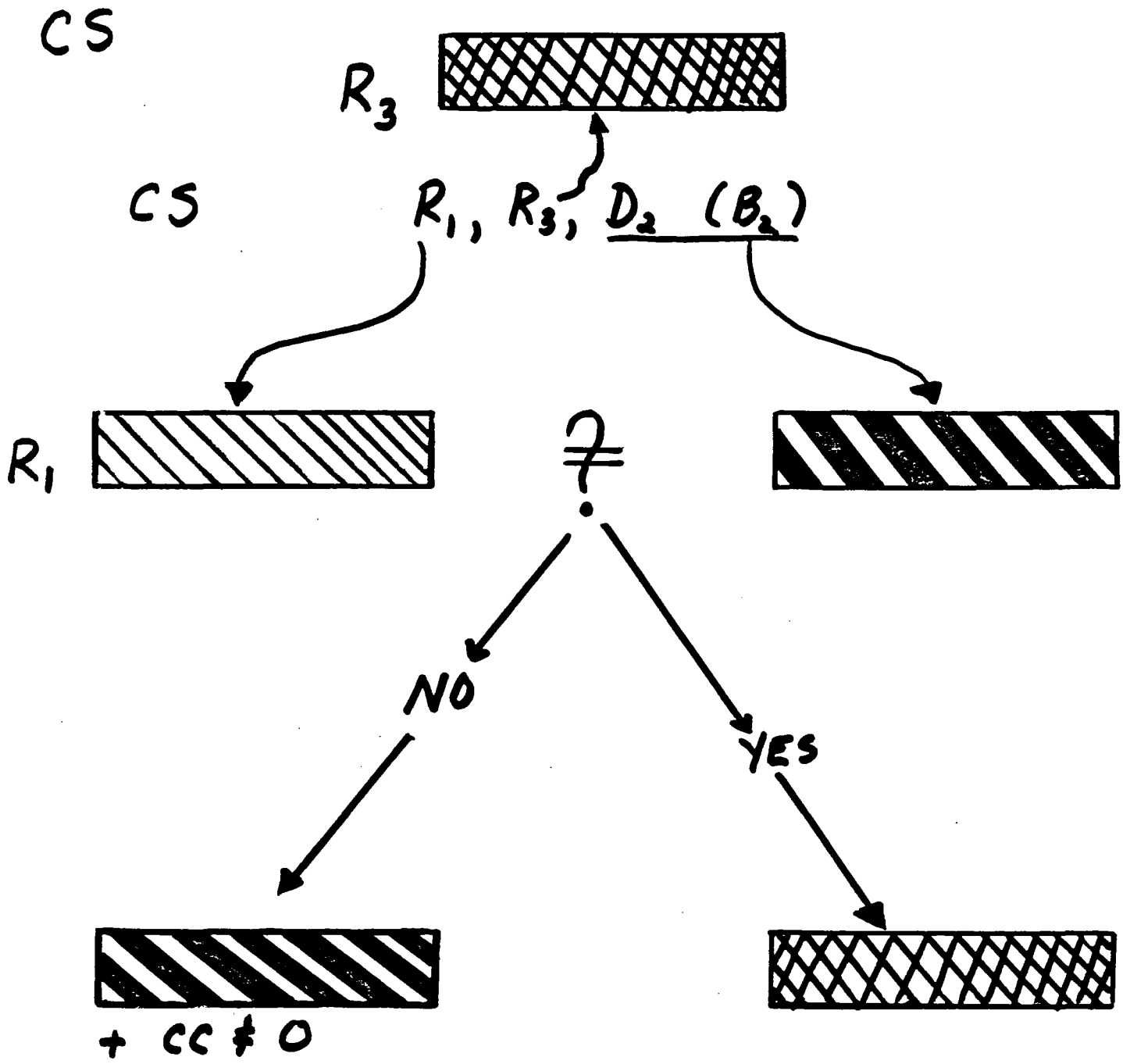
SHARED SEGMENTS



PREFIX REGISTER





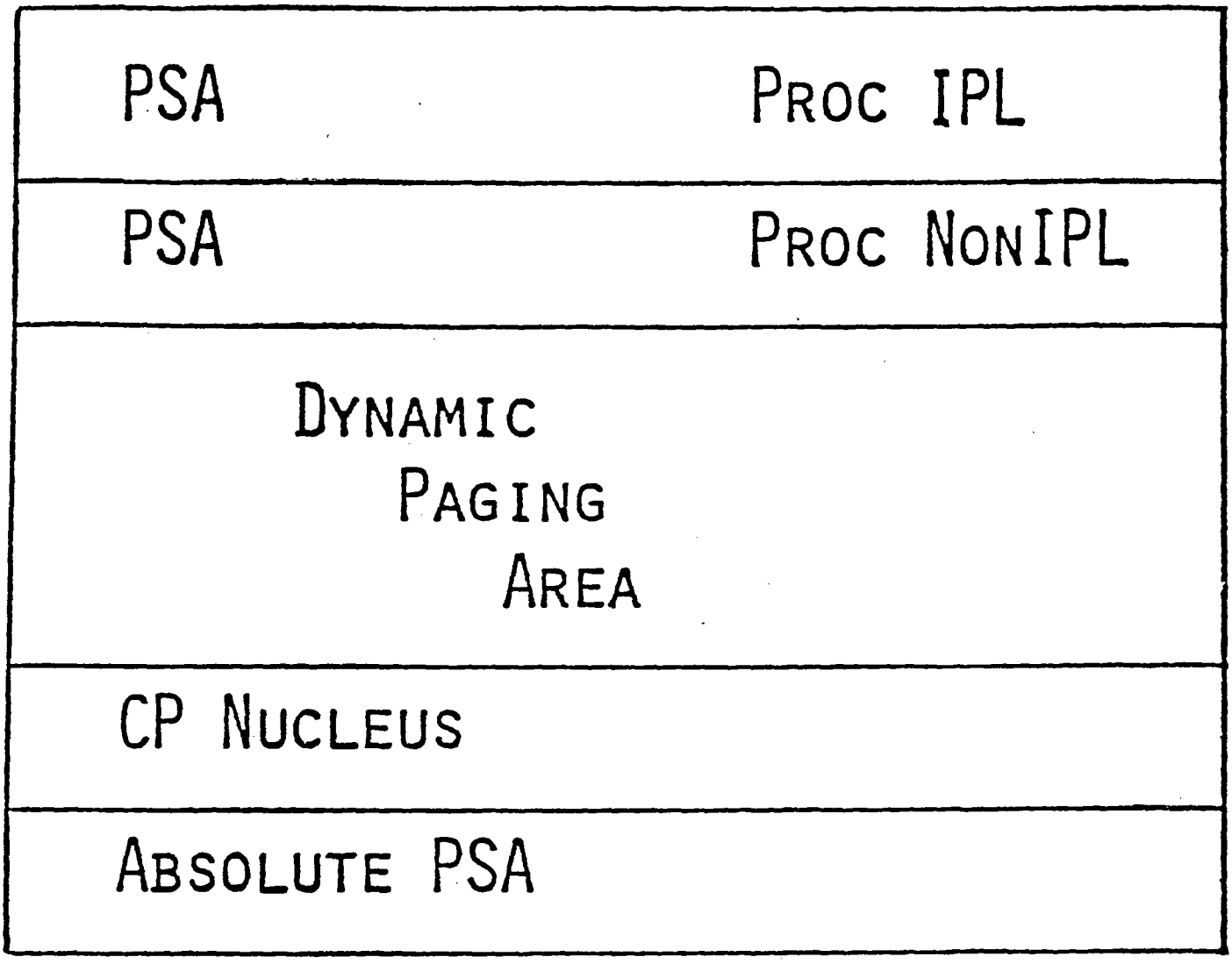



```
LOOP    L     Rx, WORD
        LR    Ry, Rx
        A     Ry, F1
        CS    Rx, Ry, WORD
        BNZ   LOOP
```

MP/AP

Storage & Data Structures

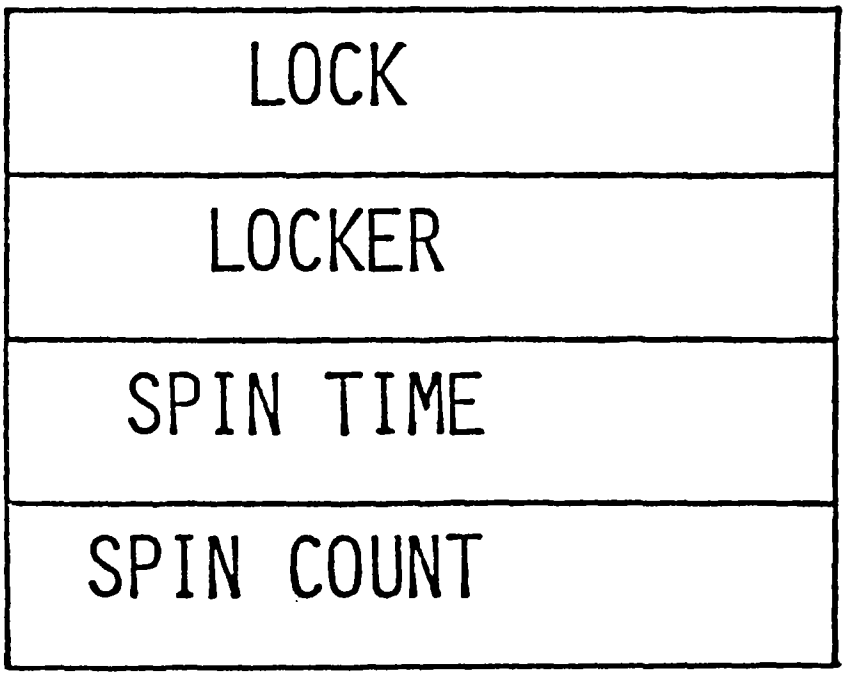
Real memory



Prefix PSA's not necessary at high memory

MP/AP

Lock Words



NOTES

Chapter 18

TRACE TABLE AND DUMPS

18.1 INTRODUCTION

18.1.1 Overview

This chapter is concerned with two of the most often used facilities in CP for finding errors. The first is the CP trace table. The second is a CP dump.

18.1.2 References

18.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

2. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

3. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

4. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

5. *IBM Virtual Machine/System Product: System Messages and Codes* (SC19-6204).

18.1.2.2 CP modules

1. DMKDMP - writes system dumps to the system spool, to tape, or to a printer.

2. Many CP modules contain the TRACE macro, which expands into the code that places data into the trace table.

18.2 TRACE TABLE OPERATION

The trace table is simply an area of main storage that is allocated at IPL time. Its size generally increases as the amount of main storage increases. If an installation wants to explicitly set the trace table size, there is a parameter in DMKSYS for specifying the size.

There are three pointers, TRACSTRT (X'0C'), TRACEND (X'10'), and TRACCURR (X'14'), located in the absolute PSA. The three addresses are the beginning of the trace table, the end of the trace table, and the address of the next trace table entry to be used. Since these pointers are in the absolute PSA, they are shared by all processors in a multiprocessor environment; there is just one trace table even in an AP or MP system. Any module within CP that creates an entry in the trace table updates the current entry pointer. When the last entry of the trace table is created, the current pointer wraps to the beginning entry. Each entry is 16 bytes long and has a fixed format. The flags used by CP modules to decide which items to trace are located at TRACEFLG (X'400') in the PSA. By setting the trace table flags, you can selectively trace events in the system. For example, if you are tracking an I/O problem, you can set the trace table flags to trace only SIOs, I/O interrupts, and posting I/O interrupts to virtual machines.

A detailed discussion of the entries follows.

18.3 TRACE TABLE ENTRIES

Table 31 describes the various entries in the trace table. While reading the trace table, you should remember:

1. The hexadecimal ID is ORed with a X'40' if the entry was generated on the non-IPL side of an multiprocessor system and is ORed with X'80'if the entry was generated in ECPS microcode.

2. If the SVC is X'0C' (EXIT), then the return address is given instead of R15.

TABLE 31

Trace table entries

Entry Type	Module	ID	Information in entry
EXT Intrp	DMKPSA	01	Interrupt Code & old PSW.
SVC Intrp	DMKSVC	02	Interrupt Code, R15, & old PSW.
PGM Intrp	DMKPRG	03	Interrupt Code, old PSW & first 3 bytes of VPSW.
Machine Check Intrp	DMKMCH	04	VMBLOK, first 4 bytes of MCH interrupt code, & old PSW.
I/O Intrp	DMKIOT	05	RAddr, CSW, & old PSW addr.
Get Storage (FRE)	DMKFRE	06	VMBLOK, amount requested, and address assigned.
Return Storage (FRET)	DMKFRE	07	VMBLOK, amount and address to be given back.
Enter SCH	DMKSCH	08	VMBLOK, VMBLOK flags, & R14.
Queue Drop	DMKSCH	09	VMBLOK & VMBLOK flags.
Run User	DMKDSP	0A	PSA RUNUSER value & RUNPSW.
SIO or SIOF	DMKIOS DMKCPI DMKCNS	0B 18	CC, RAddr, IOBLOK addr, CAW & CSW+4.
Unstack I/O	DMKDSP	0C	VAddr, VMBLOK, & VCSW.
VCSW Store	DMKVSI	0D	VAddr, VMBLOK, VCSW, & instruction opcode.
Test I/O	DMKCPI DMKIOS	0E	RAddr, CC, addr of IOB, CAW, & CSW+4.
Halt Device	DMKCNS DMKIOS DMKVSI DMKCPI	0F	CC, RAddr, addr of IOB, CAW, & CSW+4.
Unstack IOBLOK or TRQBLOK	DMKDSP	10	VMBLOK, VMBLOK flags, IOBLOK or TRQBLOK addr, & interrupt return address (IRA).
NCP BCU	DMKRNH	11	CON flags
Lock Spin	DMKLOK	12	VMBLOK, lockword addr and contents, & return addr.
SIGP	DMKEXT	13	Return addr, cc, function & order codes.
Clear Ch	DMKVSI	14	CC, RAddr, VMBLOK & VCSW
IUCV	DMKIUA	15	function code, path id, & addr of IUCVBLOK.
SNA CCS	DMKVCV	16	Transaction type.
DIAG 80	DMKMHC	17	HCBLOK & MSFBLOK addr, MSSF cmd

18.4 DUMP FACILITY OVERVIEW

If CP encounters a logical or physical error, it issues an SVC 0 to ABEND CP. The ABEND request is processed by DMKDMP, a resident nucleus module. The contents of main memory and the registers are dumped and CP tries to restart itself. The standard ABENDs in CP have names composed of two parts. The first three characters of the name specify the module issuing the ABEND and the last three characters are a three digit number. For example, a PRG001 ABEND in CP indicates that DMKPRG, the program interrupt handler, has fielded an interrupt code of 001. Program interrupt code 001 is an operation exception, which should not occur within CP. Therefore CP is terminated. The possible causes of each ABEND are documented in the *System Messages and Codes* manual.

The normal destination of the dump created by an ABEND of CP is a spool file. As was mentioned earlier, an installation may explicitly reserve dump space. If dump space is reserved and not occupied by a previous dump, then the dump spool file is allocated in the dump space. If no dump space is reserved or the reserved space is full of previous dumps, the dump program attempts to allocate contiguous space in the spool. If the spool space is fragmented enough to prevent allocation of contiguous space, the dump fails. With the SYSOPR macro in DMKSYS the installation can specify the userid to which the dump should be directed.

If the dump program has been failing because of reported I/O errors on the spool or dump space, the SET DUMP command can be used to redirect future dumps to a tape drive or to the real printer. Since DMKDMP is not very clever, if the installation is trying to write a dump on tape, the dump must fit on one reel and will be written at 1600 bpi. The dump tape is written 132 bytes per record unblocked. Any system with more than about 4 megabytes of storage can not be written on one reel of tape. Also the installation must have a real tape drive available and ready for use during the time the tape is the target for the dump. The ability to re-direct the dump to a tape is helpful when an installation is creating a dump for the IBM support center.

18.5 PROCESSING THE OUTPUT OF A CP DUMP

If the dump has been to a spool file, normally a system programmer logs on to the userid specified in DMKSYS as the recipient of system dumps and issues the commands needed to process the dump. A full discussion of this process is outside of this presentation. If the installation has the Interactive Problem Control System (IPCS) product, the User's Guide has documentation. If the installation does not have IPCS, then the VMFDUMP command is available for reading the dump from the virtual reader, storing it on a CMS minidisk and optionally, printing it.

If the dump has been written to a tape, it can be printed by attaching a tape drive to the system programmer's virtual machine, mounting the tape, and using the MOVEFILE command with the appropriate FILEDEFs for the tape drive and the virtual printer. The details of this process are documented in the *Operator's Guide* manual.

18.6 SUMMARY

This chapter has dealt with two tools available to debug CP. The trace table and CP dumps are invaluable for tracking down problems and documenting them. Several other facilities are also available. SMART, the real time monitoring system for VM, allows a system programmer to look at formatted trace table entries while the system is running. SMART also allows the setting of trace table flags to record only entries that are helpful in tracking a particular problem.

NOTES

Chapter 19

SYSTEM DIRECTORY

19.1 INTRODUCTION

19.1.1 Overview

The CP directory is a page-formatted DASD space containing data items that define the configuration of each virtual machine. Each definition contains the userid, password, memory limits, I/O configuration, and options for the virtual machine.

There are two different facilities for maintaining the directory. The directory can be built from scratch by issuing the DIRECT command in CMS or by running a standalone program on real hardware. Both of these facilities use the module DMKDIR. The second way of updating the directory allows selected fields for an existing virtual machine configuration to be modified without regeneration of the full directory. This second method is used by the IBM product known as 'DIRMAINT', although it can also be used by a user-written program.

Before we look at these two methods of maintaining the directory, we will review the structure of the directory space. Remember that utility program DMKFMT writes an allocation record 1024 bytes long. One of the kinds of space that can be declared for special use is directory space. A directory cylinder is marked in the allocation map with either a X'04' or X'0C'. The X'0C' flag marks a cylinder that contains the current system directory. This current directory is also pointed to by a field in the DASD volume's VOL1 record. An X'04' flag marks a cylinder available for receiving a new directory. After the new directory is built, the allocation map entry will be changed to X'0C' and the old directory's cylinder will be changed to X'04'. The use of the build and swap technique is used in directory maintenance for maximum reliability. All directory cylinders are page-formatted and most accesses to the directory use the paging subsystem for I/O.

19.1.2 References

19.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

19.1.2.2 CP modules

1. DMKDIR - is the DIRECT command, both as a CMS command and as a stand-alone program. (It is *not* a part of the CP nucleus.)

2. DMKUDR - contains the major subroutines that deal with directory I/O.

3. DMKUDU - supports the update-in-place DIAGNOSE X'84'.

19.2 DIRECTORY STRUCTURE

The directory exists as page-sized records in the directory space. There are five different control blocks used to map different kinds of directory data. Table 32 lists the five different control blocks.

TABLE 32

Directory control blocks

Name	Length	Information
UDIRBLOK	X'18'	Userid and password
UMACBLOK	X'50'	Options and IPL name
UDEVBLOK	X'38'	Virtual device
UIPLBLOK	X'40'	IPL parameters
UIUCBLOK	X'10'	IUCV parameters

Each page in the directory is devoted to one kind of control block. For example, all of the UDIRBLOKs exist in pages containing only UDIRBLOKs.

The directory space is page-formatted. Each page in the directory is addressed with a CCPD similar to DASD slot addresses in CP paging space.

19.2.1 UDIRBLOK

A UDIRBLOk is built for each USER card in the directory source file. In addition to the userid and password, there are pointers to the machine description blocks (UMACBLOKs) associated with this user. These pointers consist of a CCPD and an offset into the page.

The first UDIRBLOK on a page is not a description of a user. It is a control block containing pointers to the last UDIRBLOK on the page and the CCPD for the next page of UDIRBLOKs. Each page can contain 169 UDIRBLOKs plus the pointer control block.

19.2.2 UMACBLOK

The user machine description block (UMACBLOK) is pointed to by the UDIRBLOK. This control block maps the ACCOUNT card, the IPL card, the OPTIONS card, and all parameters given on the USER card except userid and password. The UMACBLOK can also contain information specified by a SCREEN card. Like the UDIRBLOK, there is only one UMACBLOK per userid. The UMACBLOKs for several virtual machines are located on one page.

The UMACBLOK also contains the pointers to the first UDEVBLOK for the virtual machine. If the IPL card does not specify parameters, the named system to be IPL'ed is contained in the UMACBLOK. If, however, the IPL card specifies parameters, then the UMACBLOK contains the CCPD and offset of the IPL extension block (UIPLBLOK). There can also be pointers to an UIUCBLOK if IUCV has been authorized for this virtual machine.

19.2.3 UDEVBLOK

The UDEVBLOK contains information about one virtual device described by an MDISK, SPECIAL, or SPOOL card. For virtual DASD devices, there are passwords for each of the access modes, the volume name of the real device that contains the virtual disk, and the size and relocation factor for the minidisk.

19.2.4 Masking

Many fields in directory control blocks are masked for security by exclusive ORing onto the field a mask value of X'AAAAAAAAAAAAAAAA'. If someone is casually perusing a dump or examining CP storage, the userids, passwords, and other data will not exist in "clear text". Unfortunately, the mask is distinctive enough that someone having access to dumps or CP storage can spot the data areas and do the exclusive OR operation manually. CP programs that deal with the directory will perform the exclusive operation on data coming into or going out of the directory.

19.3 BUILDING THE DIRECTORY

During CP initialization, the allocation records of CP-owned disks are read into main storage and searched for the X'0C' flag that marks the active directory. All of the UDIRBLOK pages are mapped into the CP's own virtual memory. These pages are paged-out through normal paging mechanisms to paging space. The list of the virtual memory addresses of these pages is anchored at DMKSYSPL (page list). The CCPD fo the beginning of the active directory is stored at DMKSYSUD (user directory). Directory pages containing other directory control blocks are not touched. When one of them is needed, that page will be paged-in, but will never be written to the paging space. Placing the UDIRBLOK pages in the page space is a performance feature of CP.

19.3.1 DIRECT command

DMKDIR is the module that runs standalone on real hardware or is invoked from CMS as the DIRECT command. It reads a source file and builds the appropriate control blocks in page-sized records. It also reads the VOL1 record and allocation record on cylinder 0 and updates them if the source directory is processed without error. If DMKDIR is executing on native hardware, then it is finished. If, however, it is running under CMS, DMKDIR executes a DIAGNOSE X'3C', which performs a dynamic directory swap to make the new directory active in the system without an IPL. The only parameter for this DIAGNOSE is the volume name of the volume containing the new directory.

19.3.2 DMKUDRDS - directory dynamic swap

As part of the dynamic swap processing, DMKUDRDS also reads the VOL1 and allocation records using an extended IOBLOK. The IOBLOK contains all of the buffer areas for the contents of the VOL1 and allocation records. After validity checking the directory pointers, DMKUDRDS calls DMKUDRBV (Build Virtual buffer).

19.3.3 DMKUDRBV - directory activation

DMKUDRBV is responsible for getting virtual addresses for the new directory's UDIRBLOK pages. It then calls DMKLOCKQ to serialize access to the fields in DMKSYS to be updated. DMKUDRBV then stores the new directory's beginning CCPD at DMKSYSUD and the anchor to the list of virtual addresses for the new UDIRBLOK pages at DMKSYSPL. It rescinds the serialization request by calling DMKLOCKD and returning to DMKUDRDS, which returns to its caller.

19.4 OTHER ENTRY POINTS IN DMKUDR

In order for LOGON, LINK, and other CP processes to access data in the directory in a standard way, there are several entry points in DMKUDR. Table 33 lists selected entry points and their functions.

TABLE 33

Selected DMKUDR entry points

Entry point	Description
DMKUDRFU	Find User (UDIRBLOK).
DMKUDRFD	Find Device (UDEVBLOK).
DMKUDRMD	Find Machine (UMACBLOK).
DMKUDRXI	Find IPL Extension (UIPLBLOK).
DMKUDRIA	Find IUCV (UIUCBLOK).

DMKUDRFU moves the contents of a UDIRBLOK into a buffer supplied by the caller. It also exclusive ORs the userid and password with the X'AAAA' mask to make the data readable to the caller. The other entry points perform similar functions.

19.5 UPDATING THE DIRECTORY IN PLACE

It should be noted that the DIRECT command rebuilds the entire directory. Until recently, the DIRECT command, or its standalone equivalent, was the only way of making any change to the directory. Given a large directory with several thousand users, there was much unneeded processing required to make even a minor change in the directory. In addition, installations had asked IBM for a facility to allow general users to modify their own LOGON and minidisk passwords.

To provide this function, IBM invented a new DIAGNOSE code. DIAGNOSE X'84' will update selected existing fields in the active directory by simply paging in the appropriate record, making the change, and paging the record back out to directory space. The module responsible for this processing is DMKUDU. The module is well written and is recommended to anyone wishing to understand directory processing in general. There is a limit to what the 'update in-place' function can perform. It can not add control blocks. If a user wants an new minidisk, the standard directory process must be used.

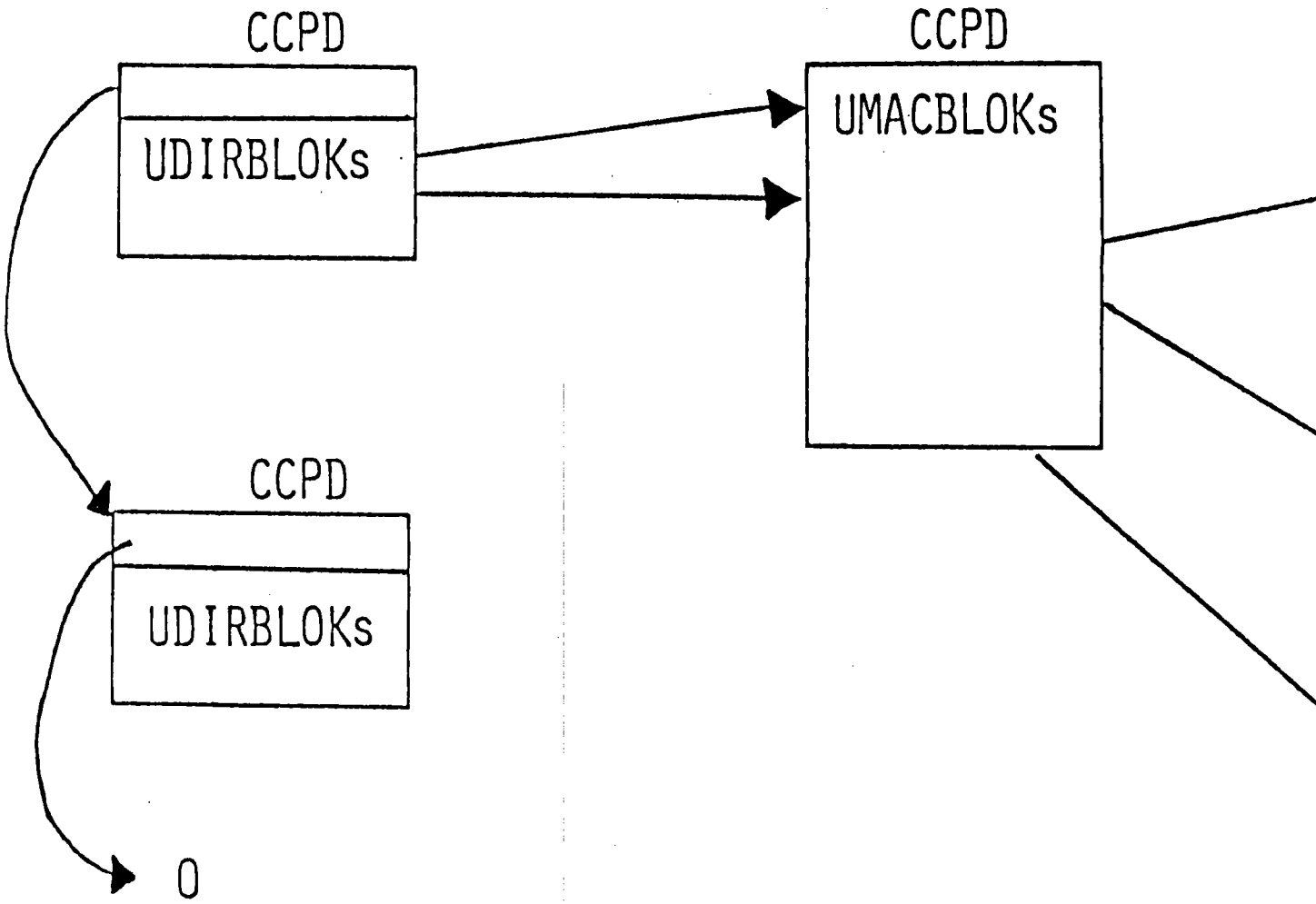
Remember that all of the UDIRBLOKs are mapped into the system virtual memory and therefore can exist both in page space and in directory space. For any request that changes the contents of a UDIRBLOK, DMKUDU must page-out the changed page to both the page space and the directory space (or the object space, as it is referred to in DMKUDU).

19.6 SUMMARY

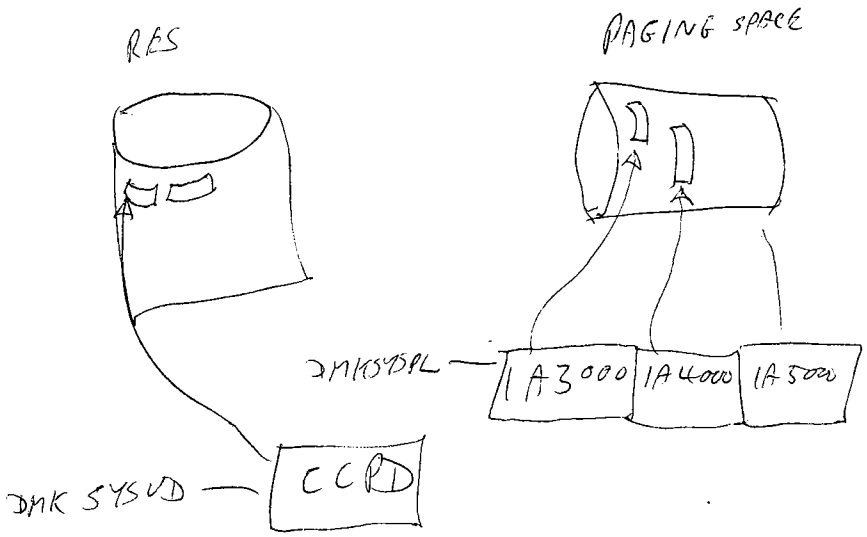
The directory facility is unique to CP. The paging subsystem is used by all accesses to the directory. DMKDIR builds a directory from a source file. DMKUDR allows CP programs to access directory contents in a standard way. DMKUDU updates fields in the active directory based on user request. DMKDIR uses DIAGNOSE I/O when it is executing under CMS and SIOs when running on the real machine. The major outstanding problem with the directory is its size. When CP/67 was designed, directories with thousands of users were not anticipated. The IBM product, DIRMAINT, does not alleviate the performance problem caused by directory searchs. How-

ever, by using the 'update in-place' mechanism, it does solve most of the updating performance problem. It also allows users to change passwords and other virtual machine characteristics in real time.

DASD DIRECTORY SPACE LAYOUT



NOTES


RES
PAGING SPACE
DMKSYSPL
IA3000
IA4000
IA5000
DMK SYSUD
CCRD

PART III

GLOBAL TOPICS

The following chapters discuss "global" topics, each of which involves several of the specific topics which we have already discussed. Each chapter should show you how various parts of CP interact to perform large-scale tasks.

Chapter 20

MICROCODE ASSISTS

20.1 INTRODUCTION

Since its introduction, VM/370 has been the target of criticism about the "excessive" overhead of privileged instruction simulation. Some of that criticism has been shown to be unjustified, but there are certain cases in which instruction simulation does in fact consume a great deal of system resource; that is especially true when a virtual storage operating system is run in a virtual machine. The more complex virtual operating systems (such as MVS, VM, and TSS) suffer more from instruction simulation overhead than do the simpler systems (such as VS/1 and DOS/VS). In an attempt to improve the simulation, IBM has implemented various parts of the simulation in hardware or in microcode.

20.1.1 Overview

This chapter will discuss the two forms of assists, VMA and ECPS for VM. There exist assists for MVS and for VS1, but we will not discuss them. Each type of assist provides support for certain instructions and features, and each can be enabled and disabled in whole or in part. Some examples of instruction timings will also be given.

20.1.2 References

20.1.2.1 Publications

There is little generally available documentation on VMA and ECPS, but the following should be of some help:

1. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203). This has a brief description of VMA.

2. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891)

3. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892). This has an excellent description of ECPS.

4. *Virtual-Machine Assist and Shadow-Table-Bypass Assist* (GA22-7074). This is a detailed description of VMA.

5. The best description of VMA is given in the maintenance instructions for RPQ S20573, VMA for the 370/168. These instructions are exceedingly detailed, but they do not refer at all to ECPS.

20.1.2.2 CP modules

1. DMKBLD - builds a VMBLOK.

2. DMKCFS - processes the SET ASSIST command.

3. DMKCFY - processes the SET SASSIST command.

4. DMKCPI - performs system initialization.

5. DMKDSP - is the system dispatcher.

6. DMKFPS - performs special OS/VS simulation.

7. DMKIUE - processes an IUCV interrupt.

8. DMKPMA - supports the preferred machine assist.

In addition to those modules, the following modules have logic that is associated with ECPS:

1. DMKCCW - performs CCW translation.

2. DMKDGD - performs DIAGNOSE X'18' I/O.

3. DMKDGF - handles DIAGNOSE X'18' I/O interrupts.

4. DMKFRE - manages CP control block storage.

5. DMKHVC - handles virtual DIAGNOSE simulation.

6. DMKMCH - handles machine check processing.

7. DMKPGS - handles paging.

8. DMKPRV - handles privileged instruction simulation.

9. DMKPTR - handles paging.

10. DMKRPA - handles paging.

11. DMKSCN - various scan subroutines.

12. DMKSTR - handles paging.

13. DMKSVC - handles CP CALL and EXIT macros.

14. DMKTMR - handles timer support.

15. DMKTRA - handles virtual paging.

16. DMKUNT - performs CCW and CSW translation.

17. DMKVAT - manages shadow page tables.

18. DMKVAU - manages shadow page tables.

19. DMKVMA - supports shared segment protection.

20. DMKVSI - handles virtual SIO simulation.

21. DMKVSJ - handles virtual HIO simulation.

20.2 STANDARD VMA

The original Virtual Machine Assist feature was provided on the models 135, 145, 158, 165-II, 168, 3031, 3032, 3033 and 3081; in some cases VMA was an RPQ rather than a standard feature. We will discuss VMA processing, hardware control of VMA, and the associated CP commands.

20.2.1 VMA processing

VMA performs three types of work that would otherwise have to be performed by CP itself:

1. Simulation of certain privileged instructions.

2. Simulation of most virtual machine SVC interrupts.

3. Automatic maintenance of shadow page tables for virtual storage virtual machines.

20.2.1.1 VMA instruction simulation

The major source of virtual machine overhead in many cases was found to be the simulation of a few privileged instructions that were issued very often by VS/1 and VS/2. Such instructions of course cause a privileged operation exception interrupt when they are issued from a virtual machine. CP must field the interrupt and eventually simulate the effect of the instruction. For many common cases the majority of CP processing was merely the overhead of dealing with decoding the instruction and dispatching the virtual machine after the simulation was completed; the actual simulation itself was often very simple. For such cases, VMA was introduced; hardware or microcode in VMA would perform the simulation without ever giving control to CP, thereby avoiding most of the overhead of the total simulation process. The following is a list of the instructions that are simulated by VMA:

1. IPK ("insert PSW key")
2. ISK ("insert storage key")
3. LPSW ("load PSW")
4. LRA ("load real address")
5. RRB ("reset reference bit")
6. SPKA ("set PSW key from address")
7. SSK ("set storage key")
8. SSM ("set system mask")
9. STCTL ("store control registers")
10. STNSM ("store then and system mask")
11. STOSM ("store then or system mask")

In some cases, VMA cannot fully simulate the instruction. For example, if a SSM instruction enables for interrupts that have been pending for the virtual machine, then additional processing must be performed in the dispatcher to simulate the PSW swapping that is part of the interruption processing. In such a case, VMA will quit and let the privop exception take place so that CP will perform the simulation. In such cases, some additional processing time is required, but that is usually overshadowed by the improvements that VMA is able to achieve on the average.

Table 34 shows some comparative timings with and without VMA for some older System/370 models.

TABLE 34

VMA privop timings (microseconds)

Opcode	145		158-3		168-3	
	std	VMA	std	VMA	std	VMA
IPK	638	638	314	3	78	2
ISK	708	24	333	7	85	8
LPSW	743	15	352	6	92	2
LRA	1131	40	518	12	133	12
RRB	799	28	367	9	104	8
SPKA	673	673	324	4	81	2
SSK	739	28	344	9	96	14
SSM	720	11	350	6	90	3
STCTL	2015	39	871	13	271	9
STNSM	733	8	355	5	91	3
STOSM	733	9	355	6	91	3
SVC	721	32	331	9	85	7
(TIO	897	961	409	438	107	115)

As you can see, most of the instructions are very much faster when VMA is available. Some however, like TIO, that are not handled by VMA take a little longer since VMA first looks at the potential privop exception to see if it is one that VMA can handle. In an extreme situation, VMA might end up causing more overhead than it is worth, but such a case is only theoretical; in all known real test situations VMA has shown a significant overall improvement in system throughput and reduced overhead. Test MVS systems with VMA have shown relative batch throughput improvement factors of 2 to 3 when compared to non-VMA systems.

20.2.1.2 SVC interrupt simulation

VMA handles SVC interrupts from problem state by simulating the PSW swapping in the virtual machine's page 0. However, that simulation is not performed if the SVC number is 76; in that case, a normal SVC interrupt is caused, giving control

to CP. That strange behavior is designed to allow CP to intercept "error recording" calls, which in OS are performed by SVC 76. If CP finds that R0 and R1 contain valid error recording information, it then performs the recording itself and treats the SVC as a NOP. If the registers are not valid for error recording, CP then reflects the SVC interrupt to the virtual machine for its own SVC interrupt handler to process. Table 34 above shows the improvement in CPU utilization when VMA is able to handle the SVC interrupt.

20.2.1.3 Shadow table handling

For the case of a virtual storage operating system in a virtual machine, CP must maintain "shadow page tables", which perform a two-level translation from virtual-virtual addresses to real addresses. The shadow table is used by the real hardware since its segment table address is in CR1.

When such a table is active and a page exception interrupt occurs, and if the associated real page is in fact still available in real storage, then VMA examines the other page tables and attempts to update the shadow table entry for the correct virtual-virtual to real translation. This process can avoid a real page exception interrupt and the associated CP handling.

20.2.2 Hardware control of VMA

VMA is controlled by control register 6. The following bits in CR6 pertain to VMA:

1. Bit 0 enables VMA. If this bit is 0, then VMA is inactive.

2. Bit 1 indicates that the virtual machine is in virtual problem state.

3. Bit 2 inhibits VMA's processing of ISK and SSK instructions.

4. Bit 3 disallows the virtual supervisor mode execution of the 370 DAT instructions; the virtual machine must behave as if it were executing on a virtual 360.

5. Bit 4 inhibits VMA's processing of virtual machine SVC instructions.

6. Bit 5 inhibits VMA's processing of virtual page table exceptions for shadow table validation.

7. Bits 8 through 28 contain the real memory address of the doubleword aligned MICBLOK, which contains other values used by VMA.

The format of the MICBLOK is given in Figure 40. Note that some of the MICBLOK fields are used by ECPS and not by VMA itself; those fields will be described later. The MICBLOK must not cross a 2K memory boundary; when the virtual machine's MICBLOK is obtained from free storage, DMKLOG must insure that this restriction is obeyed and so it will continue to call DMKFRE until a block is obtained that does not cross a 2K boundary. The MICBLOK is addressed by VMMICRO in the VMBLOK and of course by CR6 when the virtual machine is running.

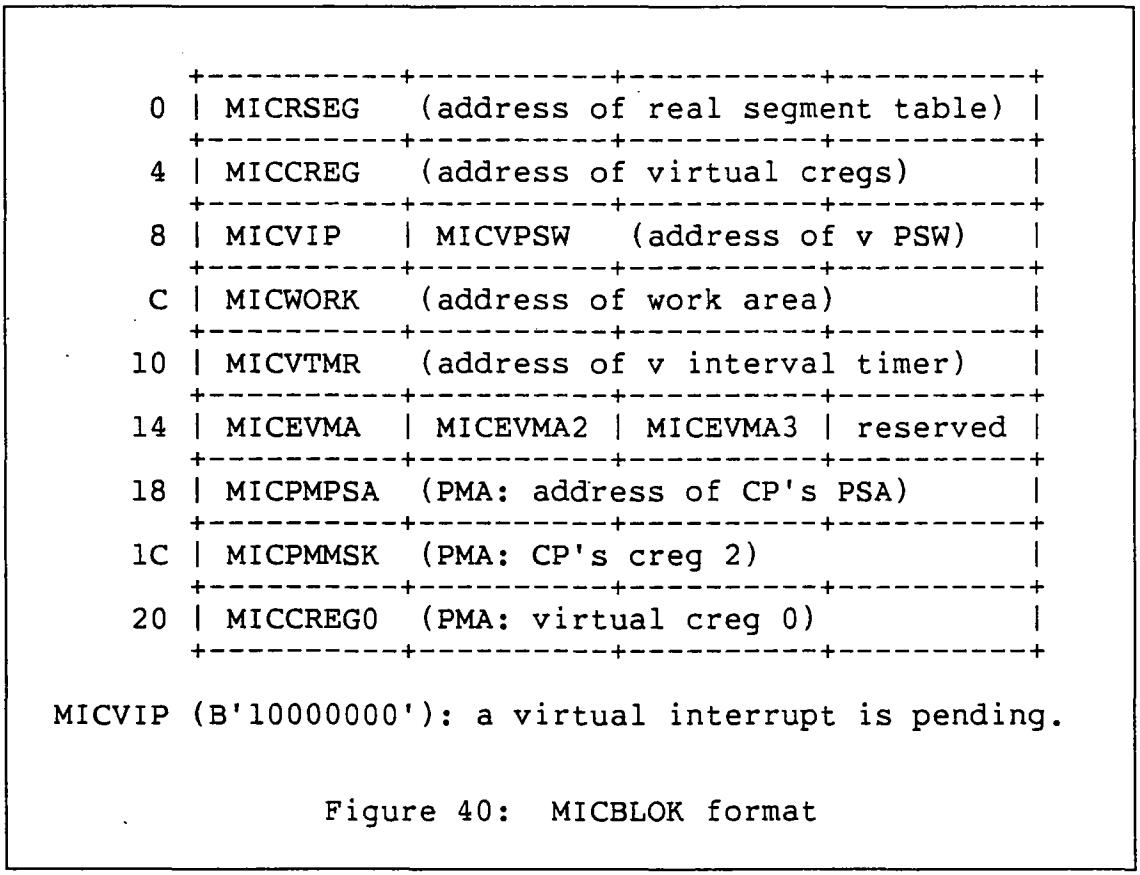


Figure 40: MICBLOK format

The most important thing for a system programmer to remember about the MICBLOK is this:

THE MICBLOK IS USED BY HARDWARE; YOU CANNOT CHANGE IT.

That is, you cannot re-format it in any way. You must be very careful when turning on bits in the MICBLOK for the same reason. Similarly, VMA assumes that various fields in the VMBLOK are at certain standard offsets. In effect, you cannot re-format the portion of the VMBLOK defined by IBM; if you need to add any fields to the VMBLOK, then add them at the end.

20.2.3 VMA commands

CP provides several commands that can be used to enable or disable VMA. These commands also affect ECPS, as described in a later section.

1. SET SASSIST OFF/ON clears or sets bit 0 of CR6 for the entire system. This command can be issued only by the system operator since it is limited to class A users.

2. SET ASSIST OFF/ON clears or sets bit 0 of CR6 (if setting has been enabled for the entire system) when the current virtual machine is being dispatched. This command can be issued by any class G user.

20.3 ECPS

Extended Control Program Support places additional function into microcode. ECPS is available in several forms:

1. Full ECPS is a standard feature of the models 135-3, 138, 145-3, 181, 4341, and 4381.

2. A subset of ECPS is available on the models 3031 and 3031AP.

3. A different subset is available on the 4331.

ECPS replaces (and includes the functions of) VMA. There are three major components of ECPS:

1. Control Program Assist is the implementation in microcode of several portions of CP. These are simple routines that are executed very often.

2. Extended VMA includes the VMA functions and enhances them to assist more instructions in more cases.

3. Virtual Interval Timer Assist causes a virtual interval timer (at a virtual address X'50') to be updated along with the real interval timer.

20.3.1 CP assist

CP Assist is implemented via a new instruction of the following format:

E6xx abbb cddd

The second byte "xx" contains an operation sub-code that defines the assist routine to be invoked. The base-displacement values "abbb" and "cddd" point to two parameter lists whose contents are different for each of the assist routines. Typically the first list contains pointers to data values and the second list contains exit addresses. Table 35 gives a list of CP Assist instructions.

TABLE 35

CP assist instructions

E600	DMKFRE	Get free storage (old)
E601	DMKFRET	Return free storage (old)
E602	DMKPTRLK	Lock a page
E603	DMKPTRUL	Unlock a page
E604	DMKCCW0	CCW decode
E605	DMKUNTFR	Free real CCW chain
E606	DMKSCNVU	Find VxxxBLOKs
E607	DMKDSP1	Full dispatch
E608	DMKCCW	TRANS in a page
E609	DMKCCW	TRANS and lock a page
E60A	DMKVAT	Invalidate segment table
E60B	DMKVAU	Invalidate page table
E60C	DMKCCW1	Decode first CCW
E60D	DMKDSP0	Main dispatcher
E60E	DMKSCNRU	Find RxxxBLOKs
E60F	DMKCCW	CCW decode
E610	DMKUNTRN	Untranslate CCWs
E611	DMKDSP2	Fast dispatch
E612		Store ECPS version ID
E613	DMKVMA	Locate changed shared page
E614	DMKFRE	Get free storage (new)
E615	DMKFRET	Return free storage (new)
0A08		CALL macro
0A0C		~~RETURN~~ EXIT macro

CP Assist can be disabled by replacing the E6xx instructions with NOP instructions (NOP plus NOPR for 6 bytes total). CP itself performs that function in DMKCPI when it finds that ECPS is not available on the processor. Note that some of the CP Assist instructions call each other internally, and so if you disable one of the routines then you may have to disable others as well. The CP Logic manual has a good description of the interrelationships.

Each CP Assist instruction is followed by regular code that performs the same function, and that code is executed if the E6xx instruction is converted to NOPs. Each line of code is identified by a percent sign (%) in column 64, directly before the update id field.

IF YOU NEED TO CHANGE ANY LINE OF CODE CONTAINING A % IN COLUMN 64, THEN YOU MUST DISABLE THE ASSOCIATED E6xx INSTRUCTION.

That caution strictly need not apply if your CPU does not support ECPS, but to violate the caution is to invite trouble!

20.3.2 Extended VMA

The second major component of ECPS is an extended VMA. Table 36 gives a list of the additional instructions that are handled by EVMA.

TABLE 36

Expanded VMA instructions

80	SSM	(partial)
82	LPSW	(partial)
83	DIAGNOSE	(partial)
9C	SIO, SIOF	(partial)
9F	TCH	(complete)
AC	STNSM	(partial)
AD	STOSM	(partial)
B206	SCKC	(partial)
B208	SPT	(partial)
B209	STPT	(complete)
B20D	PTLB	(complete)

Control of EVMA is in CR6, whose bit 6 enables or disables the entire EVMA support. Individual bits in the MICBLOK (fields MICEVMA and MICEVMA2) are used to enable and disable each of the individual instruction assists. CP normally runs with all of the EVMA functions enabled, except when the microcode and the software are incompatible:

1. The support for PTLB cannot be used at all by VM/SP due to the software's support of multiple shadow page tables.

2. The support for TCH cannot be used if the micro code is at version number 18 or 19.

3. The support for SIO cannot be used if the microcode is at version number 21 and HPO is installed.

4. The ECPS instruction E60E (DMKSCNRU) cannot be used if the RDEVBLOK shift value is 4, as indicated by bit RDIDX in CPSTAT5. This can occur if a very large number of RDEVBLOKs are generated in DMKRIO.

20.3.3 Virtual interval timer

The third major component of ECPS is the virtual interval timer support. This support, when enabled by bit 7 of CR6, causes a virtual address X'50' to be updated whenever the real X'50' is updated. The virtual timer is located via the field MICVTMR, which CP causes to point to the running virtual machine's virtual X'50' when it is resident in real storage or to the field VMTIMER in the VMBLOK when the virtual page 0 is not in real storage. Virtual interval timer support allows the virtual machine interval timer to be as accurate as the real interval timer, which is important for the correct execution of certain older operating systems such as OS/MVT.

20.4 PREFERRED MACHINE ASSIST

The preferred machine assist (PMA) is available on the 3033, 308x, and 4381 processors and provides additional performance improvements for a V=R MVS virtual machine. This assist reduces CP overhead by running the virtual machine in real supervisor state for almost all instructions, including some I/O operations. CP support for PMA is available only in HPO versions of VM/SP.

PMA is controlled by bit 0 in control register 6; when the bit is 1, then PMA is active for the current virtual ma-

chine. The MICBLOK is also extended to provide some additional data. Since the PMA virtual machine is given the absolute PSA as its page 0, CP assigns another page as a "pseudo-absolute" PSA and places its address into the prefix register. When the PMA microcode switches control back to CP to handle some instruction, then it also loads the prefix register with CP's value from the MICBLOK and clears bit 0 in control register 6. This operation is referred to as a "context switch". Most of the special code for PMA is contained in the new module DMKPMA.

PMA supports dedicated I/O channels, and for those channels no CP intervention is required for I/O instructions or interrupts. The PMA microcode examines the CP control register 0 contents, as stored in the MICBLOK, to determine which channels belong to CP; all other channels are assumed to belong to the PMA virtual machine. If an I/O instruction refers to a non-CP channel, then that instruction executes normally; if it refers to a CP channel, then a privileged operation program check interrupt is generated along with a context switch to CP. CP initialization builds the MICBLOK channel mask by examining the RCHBLOKs in DMKRIO; those channels that are not generated in DMKRIO are therefore dedicated to the PMA virtual machine. The CP channel mask is stored in DMKDMPC2 and the PMA channel mask is in DMKDMPG2.

The following program check interrupts may occur to switch control from the PMA virtual machine to CP:

1. An operation exception (X'01') occurs when a TB (Test Block) instruction is executed.

2. A privileged operation exception (X'02') occurs for a SIGP, a LCTL that loads CR2 or CR6, or a SIO to a virtual (non-dedicated) channel.

3. A specification exception (X'06') occurs when an LPSW instruction loads a wait state PSW.

4. A context switch exception (X'27') occurs when an interval timer or CP I/O channel interrupt is pending.

These interrupts are handled by the routine DMKPMAIN.

Other interrupts that occur while a PMA virtual is running belong to the virtual machine and will be handled directly by it with no CP intervention. This includes interrupts for the clock comparator, the CPU timer, dedicated I/O channels, SVC instructions, and program checks. Note that since the PMA virtual machine runs in real supervisor state, it will actually execute the DIAGNOSE instruction. You cannot use DIAGNOSE to pass control to CP for any purpose. The interval timer is not used by MVS and so it is used by CP to

interrupt the PMA virtual machine from time to time, even when no other interrupt conditions occur.

Dispatching the PMA virtual machine is particularly difficult because the prefix register must be set to 0 for MVS, leaving CP no place from which to load the PSW and the registers. PMA therefore includes a new instruction, "Load Guest PSW", E616xxxxxxxx. That instruction uses the MICBLOK address in control register 6 as an indirect pointer to the VMBLOK and from there to the virtual PSW. The approximate sequences of instructions in the dispatch routine DMKPMASW is shown in Figure 41 below.

```
SPX   ZERO          Give MVS the absolute PSA.
MVC   TIMER,...T    Time-slice to interval timer.
LCTL  C0,C15,EXCR0 Get MVS's control regs,
LCTL  C6,C6,VMMICRO  but keep the MICBLOK.
SPT   EXTCPTMR      Get MVS's CPU timer value,
LM    R0,R15,VMGPRS  and all its registers.
DC    X'E61600000000'  Load guest PSW and go.
```

Figure 41: PMA dispatcher

20.5 SUMMARY

Rarely do system programmers have to be concerned with VMA and ECPS, except to be sure they are in use if available on the CPU. If for some reason you need to modify those portions of CP that are present in VMA or ECPS, then you must be sure to disable the microcode versions so that your software modifications will be used. That of course implies a penalty in CPU consumption for that function. It is sometimes possible to redesign a modification in such a way that the microcode can be allowed to run normally. Even IBM is not immune from this problem, since VM/SP has to disable parts of ECPS with which it is no longer compatible.

NOTES

Chapter 21

GUEST OPERATING SYSTEM SUPPORT

21.1 INTRODUCTION

21.1.1 Overview and historical perspective

The past decade has seen many changes in the status of VM in terms of how IBM chose to market various features. Initially, VM was viewed as a vehicle for converting from one operating system to another (MVT to MVS) or for testing new releases of an operating system. In neither of these environments was it particularly important how well virtual operating systems performed under VM, so only a token effort was devoted to improving their performance, except by user installations running VM. By the time that VM became a strategic product, VS/1 and DOS/VS installations were clamoring for a timesharing system on processors that could not possibly support TSO. Also, a manufacturer of plug-compatible processors was marketing an enhancement to VM allowing MVS and SVS to run alongside CP (in real supervisor mode) with nearly the same performance as native operation. The late '70s saw a number of IBM enhancements to improve the efficiency of guest operating systems.

This chapter discusses several of the facilities and options available in VM to improve the operation of IBM's operating systems. The emphasis is on those facilities designed for MVS, if for no other reason than that appears to be IBM's major direction and likely to be of most interest to you.

21.1.2 References

21.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Operating Systems in a Virtual Machine* (GC19-6216).

2. *IBM Virtual Machine/System Product: Planning and System Generation Guide* (SC19-6201).

3. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

4. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

5. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

6. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

21.1.2.2 CP modules

1. DMKFPS - provides the fast path simulation of privileged instructions heavily used in guest operating systems.

2. DMKPRV - handles the Set Prefix Register (SPX) and Store Prefix Register (STPX) instructions as well as the Signal Processor (SIGP) instruction when running with SPMODE on.

3. DMKSPM - processes the SPMODE ON/OFF command. Allocates 10 "back pocket" IOBLOKs used by DMKIOS and DMKIOT to reflect control unit busy status back to a virtual machine running in SPMODE.

4. DMKVAT and DMKVAU - handle virtual address translation and maintenance of the STO blocks used to speed up shadow page table maintenance.

5. DMKVSC - checks channel programs to verify that they are valid for bypassing CCW translation.

21.2 GENERAL FACILITIES FOR GUEST OPERATING SYSTEMS

21.2.1 Error recording

In order to centralize the collection and recording of EREP data, CP intercepts all virtual machine execution of SVC 76 and moves the EREP records to its own data area. The hardware microcode assists recognize the special handling required for SVC 76 and do not reflect the interrupt to the virtual machine.

In some circumstances, EREP data is recorded in multiple areas. First, SVC 76 issued by MVS on the native-mode pro-

cessor is not intercepted and the data is therefore recorded in SYS1.LOGREC. Second, if the QVM function is used to make a transition from running an SCP under VM to running the SCP native, while the SCP is running in native mode all EREP data is recorded in the SCP's recording area rather than CP's recording area. Installations using these facilities must recognize that EREP data from several sources must be merged to have a complete record of the hardware information.

21.2.2 Quiesce VM

The QVM command allows the VM/SP system operator to make a transition from running an SCP under VM to running the SCP native. Many restrictions must be observed before this command may be used; it would be advisable to carefully study the documentation in *IBM Virtual Machine/System Product: Operating Systems in a Virtual Machine* (GC19-6216) and the code in DMKQVM.

21.2.3 Performance options

There are several performance options available for improving the scheduling for a virtual machine running a guest operating system or for reducing the overhead of simulating a real machine for "well behaved" guests.

21.2.3.1 SET FAVOR

The SET FAVOR command is a performance option affecting the scheduling of virtual machines. For more information, refer to the chapter on the scheduler.

21.2.3.2 SET RESERVE

The SET RESERVE command allows an installation to reduce the paging of a particular virtual machine without the necessity of allocating enough real memory to contain the whole virtual machine.

DMKPTRRL in module DMKPTR contains the limit (set by the class A SET RESERVED userid nn command) of the number of reserved pages allowed for a virtual machine. DMKPTRRC in module DMKPTR contains the number of pages currently reserved. The CORRSV flag in the CORFLAG byte of the cortable

indicates that a particular page has been reserved. When the SELECT routine in DMKPTR scans the cortable looking for an available page frame, it does not select (or reset the change and reference bits) for reserved pages until:

1. the second pass unless the page steal is happening for the reserved page virtual machine, and

2. DMKPTRRC is already equal to or greater than DMKPTRRL.

When a page is read into memory for the reserved page virtual machine, CORRSV is set and DMKPTRRC is incremented so long as DMKPTRRC is less than DMKPTRRL. The number of pages available to the system for paging (DMKDSPNP) is adjusted each time DMKPTRRC is changed because the reserved pages are included in the working set size of the reserved page virtual machine (VMWSPROJ) but are NOT included in the available page count.

The RESERVEd page option must be used with care since it performs what has been referred to as a "random lock" of pages in the virtual machine's memory. Pages the virtual machine references often enough to keep in real memory never get reserved since CORRSV is only set when the particular page is read.

21.2.3.3 V=R

Defining a V=R region is the first requirement for being able to use a number of performance options discussed below. These options are especially important to reduce the overhead associated with OS guest virtual machines.

Since the virtual and real addresses are the same for all memory except page 0, and since all pages are resident, using V=R eliminates paging for a virtual machine and it also eliminates the possibility of the CP scheduler placing the virtual machine in an eligible list. Specifying SET NOTRANS ON for a virtual machine running in the V=R area allows CP to bypass CCW translation for dedicated devices or full-volume minidisks in most cases. Refer to DMKVSC for the code that determines if CCW translation can be bypassed. The most likely traps for the unaware are that devices having an alternate path defined or that devices using virtual reserve/release or defined to be read/only are not eligible for CCW translation bypass. Note also that the guest operating system is expected to use SIOF for its normal I/O operations and SIO for unusual cases (like performing a SENSE operation).

21.2.3.4 STBYPASS and STFIRST

The CP Directory options of STFIRST or VIRT=REAL must be specified before the SET STBYPASS command may be issued by a virtual machine. For a V=R virtual machine, SET STBYPASS VR causes CP to bypass most shadow table maintenance and instead use the page and segment tables of the guest system. For a V=V virtual machine or a V=R virtual machine running with Single Processor Mode (see discussion later in this chapter), SET STBYPASS nnnnK defines size of the guest operating system nucleus that is common to all address spaces within the guest.

21.2.3.5 STMULTI

The SET STMULTI command allows the specification of the size of the stack of shadow page tables maintained for a virtual machine. Without STMULTI, CP purges and rebuilds the shadow page tables each time the virtual machine reloads its Control Register 1 (the Segment Table Origin or STO register). With the STMULTI option turned on, CP maintains several sets of shadow page tables, each associated with a different value of the STO register. Therefore, when the guest operating system loads a new STO value, CP may often be able to avoid rebuilding the shadow page tables by using the tables it had previously constructed for that particular value of the STO register. One additional operand of the SET STMULTI command allows the operator of a virtual machine running MVS to specify the number of segments at the high end of the virtual address space that are common to all address spaces, allowing CP to bypass much of its shadow table maintenance for those segments. The number of saved shadow page tables can be varied with the SET STMULTI command but it has a maximum value of six.

21.3 PSEUDO PAGE FAULTS FOR VS/1

Most of what is commonly called 'VM/VS handshaking' is really code in VS/1. VS/1 initialization issues a STIDP instruction and examines the version code field; the value X'FF' indicates that VS/1 is running in a virtual machine. VS/1 then performs several operations such as using DIAGNOSE X'08' to issue the CP CLOSE command for virtual printers.

The only significant portion of VM/VS handshaking that is in CP is "pseudo page fault" processing, by which VS/1 can continue to execute even though it has suffered a real page fault. CP tells VS/1 that the page fault occurred and VS/1 in turn suspends its currently active TCB. Meanwhile CP

causes the page to be brought into memory. CP then tells VS/1 that the page is available, and VS/1 makes the suspended TCB runnable once again. The CP logic flow is as follows:

1. The real page fault causes DMKPRG to receive control. If the virtual machine is in 370 mode (SET ECMODE ON), if the virtual PSW translate bit is off and the I/O interrupt bit is on, and if SET PAGEX ON was previously given, then DMKPRG goes to DMKVATPF to handle the pseudo page fault.

2. DMKVATPF calls DMKPTRAN to bring in the page, if necessary. If that call required a page-in operation, then control returns immediately to DMKVAT, which in turn simulates a program check interruption with a code of X'14'. DMKVAT exits to the dispatcher, which finishes the interrupt simulation.

3. When the page-in operation completes, a CPEXBLOK previously constructed by DMKPTR gains control. The CPEXBLOK logic constructs a PGBLOK and attaches it to the VMBLOK; the PGBLOK contains an interrupt code of X'14' and the virtual address of the new page with bit 0 set to 1 as a "page-in" flag. The CPEXBLOK logic sets the flag VMPGPND in VMPEND and then exits to the dispatcher, which finishes simulating the second pseudo page fault interrupt for the virtual machine.

Note that this logic can be used with any virtual machine operating system that can process the X'14' program check interrupt. Currently, IBM includes that logic only in VS/1, but some users have added the logic to OS/MVT with satisfactory results.

21.4 SUPPORT FOR MVS

IBM has placed a great deal of emphasis in recent years on running MVS under VM in a production environment. While nothing on the order of the support for VS/1 handshaking has been forthcoming from MVS development, there have been significant additions to CP in order to decrease the overhead of running a guest MVS operating system. Hardware support for MVS under VM has been provided for some new processors in the form of the preferred machine assist (PMA).

21.4.1 DIAGNOSE X'6C' and low address protection

The newer 370 processors include a facility known as Low Address Protection (LAP). The purpose of LAP is to detect and prevent programming errors that store into the first 512 bytes of memory. When activated by turning on bit 3 of Control Register 0, LAP forces a protection exception for any instruction that attempts to store in addresses 0-512; the check on the address range is performed BEFORE translation (if any) and prefixing are done.

MVS SP Release 3 makes use of LAP to protect its low memory from inadvertent modification. To allow legitimate modification of the first 512 bytes of memory, MVS maps another page of its address space to real page zero; since LAP checking takes place before translation, store operations through this alternate page address proceed normally but are much less likely to happen by mistake. CP needs to know all virtual pages that might map to page 0 since CP relocates page 0 of a virtual machine running in the V=R area and directly changes the page tables of the guest system to reflect this fact. MVS issues DIAGNOSE X'6C' to inform CP of the location of the page being used to bypass LAP and gain store access to the first 512 bytes of memory. CP insures that changes made to the shadow page tables or the page tables of the guest for page 0 of the guest's virtual memory are also reflected in the page table entry for this additional page.

21.4.2 Single processor mode

VM/SP has no support for virtual machines to run in MP or AP mode. Therefore, when MVS runs under VM there is no way to have the MVS virtual machine dispatched on more than one processor at a time. Faced with the problem of needing the ability to use more of the resources of an MP system for a virtual MVS (and needing some response to a software product developed and marketed by Amdahl that allowed a guest MVS system to run in real supervisor state), IBM developed the Single Processor Mode (SPMODE) of operation. SPMODE may be used on an AP or MP processor complex. It operates by allowing MVS to run in real supervisor state with full control of one processor while sharing the other processor with VM and other virtual machines.

21.4.2.1 Restrictions

As one might imagine, use of SPMODE involves observing many restrictions; the IBM publication *Operating Systems in a*

Virtual Machine should be read with great care. Some of the restrictions are as follows:

1. If the VM system is generated for AP or MP operation, it must be running in UP mode before SPMODE may be used. Use the VARY PROCESSOR command to remove one of the processors from the configuration and return VM to a UP mode of operation. On 3081 processors, take care to use the YLOG operand of the VARY PROCESSOR command so that the second processor is not physically varied offline and is still useable by the MVS system.

2. The MVS system must be set up to run in a V=R area and generated to match exactly the I/O configuration accessible from the processor on which it is running in native mode.

3. The STBYPASS V=R option may not be used. Instead, use STBYPASS nnnnK and the STMULTI options to improve the performance of shadow table maintenance for the part of MVS running under VM.

4. The MVS virtual machine must be IPLed or re-IPLed after the system operator (or class A user) issues the SPMODE ON command.

5. Before leaving SPMODE, the MVS system should finish all processing and the MVS virtual machine should be reset (by the IPL, SYSTEM RESET, or LOGOFF commands for example). The system operator can then enter the SPMODE OFF command.

6. When in SPMODE of operation, the MVS system operator should not vary the main processor offline.

7. The LOGREC data is recorded both by VM, for errors occurring while the MVS under VM side is running, and by MVS, for errors occurring while the native MVS is running. The data from SYS1.LOGREC and CP's error recording cylinders must be merged to get a complete record of the errors that have been logged. IBM recommends that you look at the timestamp of the records to track the proper order of events (and you thought that was what they made computers for!).

8. Virtual and real channel reconfiguration are not supported. The MVS CTRLPROG sysgen macro should not specify OPTIONS=(CRH) if SPMODE is going to be used. Note that this option IS specified in the MVS/SP IPO system distributed by IBM.

9. TCAM should not be generated with the VM/370 option. Instead, use SET NOTRANS ON to allow TCAM's dynamically modified channel programs to operate properly.

10. RESTART does not produce a dump of VM when SPMODE is being used. Refer to the procedure described in *Operating Systems in a Virtual Machine* for the gory details on how to get VM to produce a dump.

21.4.2.2 Operation

When SPMODE is turned on, a flag in the PSA is set to indicate the mode of operation, a page frame is allocated to contain the prefix area for use by CP, CP's page 0 is copied to the page frame, and the prefix register is loaded with the address of the page. Note that CP is using the prefix register even though it is running in uniprocessor mode because the MVS system running in real supervisor mode on the other processor will think that is has full control and access to absolute page 0; therefore, CP must get its page 0 out of harm's way.

Once SPMODE is on, CP simulates SIGP instructions and reflects MP-type external interrupts for the virtual machine running in the V=R area. When MVS is IPLed in the V=R area, its tests now indicate that multiple processors are available and so it initializes for MP operation. Of course, part of the initialization involves firing up the other processor (under MVS's control) and then the system is off and running with two processors able to dispatch work in MVS. Since the other processor is operating in native mode using the page tables of the operating system, CP cannot directly change the page tables of the MVS guest; rather, it must maintain a set of shadow page tables to remap some of the pages for the virtual MVS system (this is the reason for STBYPASS VR not being allowed in SPMODE).

One additional bit of nasty work to be taken care of is the tracking of the virtual prefix register for the virtual side of the MVS system and making sure that the shadow page tables remain consistent with virtual prefixing. CP allocates sufficient storage for up to three additional private page tables that it must maintain. The first page table is a copy of the first page table from the MVS system except that page 0 is mapped to the page specified by the most recent Set Prefix (SPX) instruction. The second page table is a copy of the MVS page table containing the virtual prefix page. The page table entry for the prefix page must be altered to point to the page containing CP's prefix so that address translation followed by prefixing results in access to absolute page 0. The third page table is used if

DIAGNOSE X'6C' has been issued and the page used for accessing page 0 while bypassing LAP is not in the same segment as the prefix page. The page table entry for the page referenced by DIAGNOSE X'6C' is altered so that references to it map to the page specified by the most recent SPX instruction.

21.4.2.3 Summary

Use of SPMODE is a way to bring more of the hardware resources to bear on the MVS system. Depending on the configuration, performance of other virtual machines could be seriously degraded since there is less resource to give them and VM has less control over the allocation of that resource.

21.4.3 Fast privileged instruction simulation

Reordering the special case tests in the privileged instruction simulation leads to improved performance for the cases frequently executed by V=R virtual machines running SVS or MVS. The module DMKFPS performs fast path simulation of frequently executed instructions and is particularly efficient for virtual machines with STBYPASS VR. The module quickly exits back to DMKPRV or DMKPRW if there is anything unusual about the instruction or its operands (not resident, crossing a page boundary, etc.).

21.5 PREFERRED MACHINE ASSIST

(For a discussiong of PMA, please refer to the chapter on microcode assists.)

21.6 SUMMARY

Many options and features are available within VM to improve the performance and useability of guest operating systems under VM. This chapter has presented a discussion of some of the facilities available but, depending on your hardware and software requirements, there may be other options useful to your installation. The IBM publication, *IBM Virtual Machine/System Product: Operating Systems in a Virtual Machine* (GC19-6216), is an invaluable reference for providing suggestions as to other options that may be appropriate.

The performance of guest operating systems has improved enormously during recent years. Most of the improvements for running MVS under VM have been made on the VM side; consider the performance improvements that could be made if the MVS developers undertook to improve performance by taking cognizance of the CP services and doing things like avoiding privileged operations whenever possible.

NOTES

Chapter 22

VIRTUAL MEMORY INITIALIZATION

22.1 INTRODUCTION

22.1.1 Overview

Virtual machine memory can be initialized in two different but related ways. The IPL command can load programs and data into memory as well as start virtual machine execution; this is a virtualization of the real LOAD function. In addition, pre-loaded segments of memory can be attached to the virtual machine, and for this there is no real counterpart.

22.1.2 References

22.1.2.1 Publications

1. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

2. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

22.1.2.2 CP modules

1. DMKCFF - contains subroutines for DMKCFG.

2. DMKCFG - supports the IPL command and DIAGNOSE X'64' processing.

3. DMKCFH - supports the SAVESYS command.

4. DMKCFP - performs a virtual machine reset.

5. DMKHVC - processes all DIAGNOSE instructions.

6. DMKHVD - supports the special DIAGNOSE X'40'.

7. DMKPGS - performs virtual memory resets.

8. DMKVMA - supports shared memory segments.

9. DMKVMI - is the IPL simulator routine.

22.2 IPL LOGIC FLOW

The chapter on command processing describes the basic logic flow for the first portion all CP commands, where DMKCFM examines the table of valid command names in DMKCFC and then calls the appropriate processing routine. For IPL the selected routine is DMKCFGIP. An alternative entry point, DMKCFGII, is called during LOGON processing if an IPL statement is present in the user's directory entry.

The overall logic flow for IPL simulation is comprised of the following steps:

1. Scan the command line and examine the parameters.

2. For IPL from a virtual device, load the IPL simulator routine and set up the registers and virtual PSW such that the simulator will get control when the virtual machine is dispatched. Exit to our caller.

3. For IPL from a named system, find the entry in the system name table, set up the virtual machine contents, and exit to our caller.

22.2.1 Scan the IPL command line

The process of scanning the command line is the standard one of calling DMKSCNFD to locate the next token and then setting flags to correspond to the value of the token. The token 'PARM' is handled differently in that the remainder of the command line is simply stored into the virtual machine's registers.

If the first token is a hexadecimal number in the range X'000' to X'FFF', then IPL from a virtual device is attempted. Otherwise, IPL from a saved system is assumed.

22.2.2 Load from a virtual device

Do the following:

1. DMKSCNFD is called until there are no tokens left on the command line. For each token, set a flag corresponding to the token. For the CYLNO and BLKNO tokens convert the following token to a decimal number and save it. For the PARM token store the remainder of the command line into VMREGS.

2. Call DMKCPFRR to reset the virtual machine and call DMKPGSPO to possibly clear virtual storage.

3. Select the address at which the IPL simulator routine will be run. Use the address X'20000' or the middle of memory, whichever is smaller.

4. Call DMKRPAPT to force a page-out of the virtual memory page just selected.

5. Call DMKRPAGT to force a page-in of the IPL simulator, DMKVMI, into the selected virtual address.

6. Copy the command parameters into virtual page 0 so that DMKVMI can find them.

7. Call DMKSCNVU to find the VxxxBLOKs for the IPL device. If the device cannot be found, then give the DMK040 error message and exit to DMKCFM, leaving the virtual machine in a reset condition.

8. Take the virtual machine out of virtual wait and out of VMIOWAIT so that it will start execution at the entry point of DMKVMI within the virtual memory. Exit to DMKCFM.

22.2.3 DMKVMI logic

When the dispatcher gets around to running the virtual machine, then DMKVMI will start execution in virtual supervisor state within the virtual memory. Except for a few minor parts of DMKCPI, this is the only CP routine that runs in real problem state.

1. Save into DMKVMI the registers, which contain the PARM data, and the flags, which have been placed into the first 2 doublewords of virtual memory.

2. Use a DIAGNOSE code X'24' to determine the IPL device type and use that to select an internal routine.

There is one routine for DASD, another for TAPE, and another for READER and CTC.

3. Each of the routines first reads the 24 byte IPL record and then begins a process of performing I/O using the IPL channel program. Each CCW is checked to be sure that DMKVMI itself does not get overwritten.

4. At the end of the IPL channel program, use DIAGNOSE X'8' to set an ADSTOP if that was requested. Use DIAGNOSE X'8' to cause an ATTN if that was requested. Store the IPL device address into the I/O interrupt code area in page 0. Point to the saved registers and issue the special DIAGNOSE X'40' to restore the original virtual machine page, load the registers, and load the virtual PSW from location 0. This completes the virtual IPL and the virtual machine is now running the loaded program.

22.2.4 Load from a saved system

Loading from a saved system is quite different in detail although the final effect is much the same as what we have just described. The parameter scan is abbreviated since only the PARM parameter is allowed for IPL by name. The logic flow continues as follows:

1. Perform a TRANS operation to get the real address of DMKSNTBL, the system name table built at SYSGEN time.

2. Find the given IPL name in DMKSNTBL and call DMKPGSPO to reset the virtual memory.

3. Get the CCPD or PPPD address of the saved system on DASD and call DMKCFFSB to set the saved virtual PSW and registers into the VMBLOK and build the SWPTABLE entries for the saved virtual memory pages.

4. If the new virtual PSW will cause the virtual machine to run in ECMODE, then call DMKVATMD to prepare a new set of shadow page tables.

5. Take the virtual machine out of wait and out of VMIOWAIT so that it will run as soon as possible. The virtual machine will then continue execution from the point at which the SAVESYS command had been given.

22.3 SAVING A SAVED SYSTEM

A system image can be saved with the SAVESYS command. The user must first load the appropriate system into memory and bring it to the point at which a saved image is required. For CMS, that point is the first console read. The SAVESYS command is processed by DMKCFH:

1. TRANS in DMKSNTBL and search for the name of the system.

2. Get a free storage buffer for saving the protect keys of the pages in the saved system area.

3. TRANS in each page of the saved system and call DMKRPAPT to write each page out to its assigned DASD location, as shown in the DMKSNTBL entry. Save each page's protect keys into the gotten buffer.

4. Temporarily using the first saved virtual page, build a control area containing the virtual PSW, registers, protect keys, and current date and time. Call DMKRPAPT to write that page out to the first DASD slot assigned to the saved system.

22.4 DISCONTIGUOUS SAVED SEGMENTS

A discontiguous saved segment, or DCSS, is a group of segments at a specific memory address; the segments' contents are written to DASD by the SAVESYS command as described above. The DIAGNOSE code X'64' can be used to attach such a segment to a virtual machine at the defined memory address. As initially implemented, a DCSS was intended to be discontiguous; that is, it usually occupied an address range above that of the virtual machine proper. In fact CP supports the overlapping of a DCSS with the original virtual memory, but application programs must be very carefully designed if they are to support the use of an overlapped DCSS. IBM-provided functions such as CMS free storage management would most likely have problems if an overlapped DCSS were being used.

22.4.1 DCSS support logic

DIAGNOSE X'64' supports three functions, LOADSYS to attach a DCSS to a virtual machine, PURGESYS to detach a DCSS, and FINDSYS to return the memory address of a DCSS. One of the parameters to DIAGNOSE X'64' is the name of the DCSS and the other is a code for the desired function. DMKHVC and DMKHVD

decode the DIAGNOSE and call DMKCFGCL for the X'64' processing:

1. Check the function code for legality. Give an error if the user is running V=R.

2. If this is the LOADSYS function, then enter at point 1 above in "load from a saved system". Of course, the PSW and registers will not be set as in the case of IPL, but the rest of the logic is almost identical to IPLing a saved system.

3. TRANS in DMKSNTBL and search for the named DCSS. If not found, give an error return.

4. For the FINDSYS function, set up to return the beginning and ending addresses of the DCSS. Set the condition code to show whether or not the segment is already loaded. Exit to the caller.

5. For the PURGSYS function, call DMKPGSPS to purge the segment if already loaded. Exit to the caller.

22.4.2 Implications for memory protection

Since a DCSS may be shared among several virtual machines and since segment protection is not available on many System/370 processors, CP must provide for the possibility that the currently running virtual machine might change the contents of the DCSS. This is of course always a possiblity since the virtual machine can run in protect key 0 and the DCSS is naturally in the virtual machine's address space.

The solution to this problem is simple: DMKDSP must check the DCSS for modification whenever RUNUSER in the PSA is changed. That check is performed in DMKVMASH, which examines all the change bits in the DCSS's protect keys. If any change bit is set, then the virtual machine is given an error message and is placed into console function mode (CP READ). The changed page is marked invalid so that it will be refreshed when next accessed.

For a multiprocessor, this problem becomes worse since a virtual machine might be running on one processor while DMKVMASH is running on the other. The solution is to have two sets of shared pages, one for each processor. When a virtual machine is dispatched its segment tables must be updated to use that real processor's version of any loaded DCSSs.

Both of these functions contribute to CP overhead. They can be eliminated on those processors that include a read-only protection bit in the segment table entry. Such hardware is included with the 308X family of processors and is supported in HPO.

22.5 SUMMARY

Virtual memory initialization includes the critical function of virtual IPL simulation. Named systems and DCSSs can improve performance but they are not virtualizations of real machine functions. DCSSs provide a simple form of shared virtual memory.

NOTES

Chapter 23

CP INITIALIZATION

23.1 INTRODUCTION

23.1.1 Overview

The purpose of this chapter is two-fold. The first purpose is to give an overview of the order in which various CP components are initialized after a hardware IPL. The second purpose is to provide an opportunity to tie together the chapters that we have already discussed. This chapter is a "walk-through" of CP initialization and each step should help you recall the basic logic of the component parts.

23.1.2 References

23.1.2.1 Publications

1. *IBM Virtual Machine/System Product: Operator's Guide* (SC19-6202).

2. *IBM Virtual Machine/System Product: System Programmer's Guide* (SC19-6203).

3. *IBM Virtual Machine/System Product: Data Areas and Control BLock Logic* (LY20-0891).

4. *IBM Virtual Machine/System Product: System Logic and Problem Determination Guide Volume 1 - CP* (LY20-0892).

23.1.2.2 CP modules

1. DMKAPI - performs multiprocessor initialization.

2. DMKCKP - is the bootstrap loader.

3. DMKCPI - is the main CP initialization routine.

4. DMKCPJ - contains additional initialization logic.

5. DMKCPK - initialization for CP-owned DASD volumes.

6. DMKCPX - initializes the T-disk space.

7. DMKSAV - loads the CP nucleus into main storage.

8. DMKSTA - initializes main storage allocation.

9. DMKWRM - initializes spool file processing.

23.2 HARDWARE INITIAL PROGRAM LOADING

Every System/370 includes the old "big blue" LOAD button or its logical equivalent. When the system operator pushes LOAD or selects the load option on the system console, then the hardware initial program load function begins. This causes the loading of DMKCKP, as described already in the spool file recovery chapter. DMKCKP eventually invokes DMKSAV, which reads into main storage the entire CP nucleus, including the pageable part. DMKCPI, the module with primary responsibility for CP initialization, is a part of the pageable nucleus. It is invoked with a branch from DMKSAV and uses R12 and R13 for base registers. As function has been added to CP, DMKCPI has been split often and modified even more often. After it has executed, DMKCPI calls DMKCPJ, a module recently created from code in DMKCPI, which continues initialization. When all initialization tasks are completed, DMKCPJ goes to the dispatcher.

23.3 DMKCPI INITIAL HOUSEKEEPING

DMKCPI must first establish the environment in which it is running. It proceeds with this discovery as dynamically as possible. For the most part, features that CP will use are hard-coded only when there is no dynamic way to determine their existence.

1. Set up new PSWs in PSA.

2. Load Control Registers C0-C14.

3. Set ClocK Comparator with X'FF's.

4. STore ID Processor (If running second level, allow UP operation only.)

5. *ST*ore Address *P*rocessor (program exception if UP hardware.)

6. If MP hardware installed, issue:
   *CON*nect *C*hannel *S*et (X'B200')
   for channel sets from X'00' to X'3F' with a subsequent SENSE I/O operation for the IPL device address.

7. Calculate the number of BCT instructions that will execute in 50 milliseconds and store the result into the field BCTWAIT in the PSA. Valid results range from 1000 to 5 million. This gives CP the approximate processor speed.

23.4 MAIN STORAGE INITIALIZATION

Call DMKSTA to perform main storage initialization. DMKSTA sets up the trace table, the dynamic storage area, and the free storage area (see the trace table chapter and the storage management chapter). DMKSTA primes DMKFRE by giving it control of the free storage chain. From this point, control blocks are gotten by calling DMKFRE.

8. Call DMKSTANT to initialize main storage.

23.5 ECPS INITIALIZATION

Call the subroutine INITECPS to determine which level of ECPS is active on this hardware and modify CP to execute correctly at that level (see the microcode assists chapter).

Load control register 6 to enable ECPS and EVMA; prime the program check new PSW to catch a program check interrupt. Mark the flag byte PSAEVMA to indicate that the following instructions have EVMA active.

- o MICLPSW LPSW
- o MICSCSP STCK, SPT
- o MICSIO SIO
- o MICSTSM STNSM, STOSM, SSM
- o MICSTPT STPT
- o MICTCH TCH
- o MICDIAG DIAG

Issue an X'E612' instruction to determine the ECPS level number. If a program interrupt occurs, then ECPS is not active on the hardware. Therefore, convert all the ECPS instructions in CP to NOPs, using the address list at label CPATABLE. Restore the program check new PSW and return control to the mainline of DMKCPI.

If the X'E612' instruction executes, then check the ECPS level and store it at DMKCPEML. If the ECPS level is below 18 or above 21, then convert the ECPS instructions to NOPs as described above. If the level is 18 or 19, then disable the TCH assist (MICTCH in PSAEVMA) and modify the DMKFREE/DMKFRET assist instructions; this modification is necessary to reflect the increase in the maximum subpool size in DMKFREE at the level 20 microcode (see the storage management chapter). If the level is exactly 20, then leave the instructions as assembled. If the level is 21 and if HPO is installed, then disable the SIO assist (MICSIO in PSAEVMA).

9. Determine if ECPS is active on this hardware. If it is, then determine the level number and modify CP accordingly.

23.6 SAVEAREA INITIALIZATION

For a uniprocessor, set aside some save areas in a stack maintained by DMKSVC; the number of save areas is 18 plus an additional 3 for each megabyte of main storage (see the CP architecture chapter.) For a multiprocessor, compute that same number and set aside three-quarters of that value for each processor. Reserve the save areas by making individual calls to DMKFREE. Get an extra save area and anchor it at DMKFRESV for use in extend processing (see the storage management chapter).

10. Initialize the save area stack.

23.7 PREPARATION FOR EXTEND PROCESSING

Get the various CPEXBLOKs needed for extend processing; call DMKFREE to get one area large enough to contain 16 CPEXBLOKs and anchor them at DMKPTRFA. For a multiprocessor get five more CPEXBLOKs and anchor them at DMKFREAP.

Call DMKFREE for a 12 doubleword work area to be used by DMKIOS and anchor it at PSAIOSW.

11. Reserve the "back-pocket" CPEXBLOKs needed for extend processing.

12. Get a work area for DMKIOS.

23.8 DISPATCHER INITIALIZATION

13. Place the address of the system VMBLOK (ASYSVM) into the PSA fields RUNUSER and LASTUSER.

23.9 ESTABLISH THE OTHER PSA

On a multiprocessor, obtain the SYS lock and initialize the other processor's PSA. Begin using PSA prefixing on both processors. (CP's locking structure must be used from now on.)

14. Call DMKAPI to initialize the other processor.

23.10 I/O SUBSYSTEM INITIALIZATION

At the label MOUNTIPL, begin processing all the I/O devices defined in DMKRIO. Start with the IPL device and then do all the others.

15. Issue an HIO to the device and record the event in the trace table.

16. Issue a TIO to the device.

17. If RDEVCLAS is either CLASTAPE or CLASDASD (not CLASFBA), issue a SIO with a Release CCW. This operation determines whether or not the Reserve/Release feature is installed on the device.

18. If extended CKD device issue SENSE ID to determine if fixed head feature is installed.

19. If CLASDASD or CLASFBA, read VOL1 label record.

    a) Issue a TIO instruction.

    b) VOL1 record OK?

    c) CALL DMKSCNVS for duplicate volume name.

    d) If CP-formatted, look for directory pointer 52 bytes into the VOL1 record.

    e) If CLASFBA, read real device characteristics and notice if the reserve/release feature is installed. Build RDCBLOK if needed.

    f) Read allocation record.

    g) If allocation record valid, build ALOCBLOK and chain it.

20. On the non-IPLed processor perform this same I/O initialization.

23.11 MINI-IOBLOK STACK INITIALIZATION

21. Initialize the mini-IOBLOK stack and anchor it at DMKIOSMQ (see the I/O processing chapter).

23.12 SYSTEM ADDRESS BUFFER INITIALIZATION

Calculate the number of system virtual memory pages that can be allocated at any given time, based on the real memory size minus the V=R size. If the main storage size specified in DMKSYS is less than the real storage size, then use the DMKSYS storage size in the calculations. Use a default of 120 buffer pages for processors with less than 655k bytes of available storage. Increase the default up to a maximum of 1280 if the processor has more than 3 megabytes of available storage. Store the calculated maximum into DMKPGUBN.

22. Dynamically calculate the maximum number of pageable system virtual address pages.

23.13 SYSTEM VMBLOK INITIALIZATION

23. Call DMKBLDRT to build CP's own page and swap tables.

23.14 CALCULATE THE NUMBER OF DASD SLOTS

24. Call the internal subroutine CALCUL to add up the number of fixed head and movable head DASD slots in the system.

23.15 VIRTUAL MACHINE ASSIST INITIALIZATION

Set bit 0 in CR6 to enable VMA and then issue a Set System Mask instruction in problem state. If it fails, then VMA is not active on the processor, and therefore turn off the flags CPMICAVL AND CPMICON in CPSTAT2 in DMKPSA.

25. Invoke internal subroutine MICTEST2 to check for VMA.

23.16 EXTENDED 370 PROCESSOR?

Execute a Test PRoTection instruction X'E502' to determine if the 370 architectural extensions are available. If the instruction executes without an interruption, then turn on the flags CP370EAV and CP370EON in CPISTAT2 in DMKPSA.

26. Test for 370 architectural extensions.

23.17 SYSTEM CONSOLE INITIALIZATION

Begin initialization of the operator interface. First, locate and verify the availability of a system console. The internal subroutine beginning with label STRTERM contains most of this logic. This subroutine calls DMKSCNEP for the primary console address and each of the alternates listed in DMKRIO until one is located with what appears to be an available path. Check the path with a TIO and a subsequent SENSE. Stop the search at the first available console device and use it as the system console. For consoles attached through teleprocessing lines, make an additional call to DMKCNSEN to issue an ENABLE to the line before trying the TIO and SENSE.

After the console is verified, write the initial console messages. Prompt the operator to change the TOD clock value. If the operator changes the time, convert the value to internal use by an algorithm documented by Richard Stone in an article appearing in the "Communications of the ACM", October 1970, page 621.

Build a TRQBLOK to cause a timer interrupt at midnight. Build a second TRQBLOK to expire in 60 minutes to cause DMKTMRFR to be entered to garbage collect the free storage area (see the storage management chapter).

27. Locate and verify the system console.

28. Issue DMKCPI955 if memory is too small for CP.

29. Write console message with CP release and level information.

30. Ask operator verification of TOD. Change TOD as requested.

31. Build TRQBLOKs for midnight message and free storage garbage collection.

23.18 DIRECTORY INITIALIZATION

Store the active user directory location into DMKSYSUD and build the virtual page list for user machine descriptions. Normal directory processing is now available. Lock DMKCPI and DMKCPJ into main storage to avoid problems when paging activity starts.

32. Initialize the CP directory.

33. Lock DMKCPI and DMKCPJ into main storage.

23.19 LOCATION 80 TIMER TEST

Check that the interval timer at X'50' is running by doing the following test. First, in case CP is running second level, force a trip through the real CP dispatcher by executing a Store ID Processor instruction. Then, execute a 3 instruction loop 25,000 times. If the timer does not change, then send a message to the system console. Repeat this test until the interval timer is found to be working.

Send a warning message to the system console if ECPS is available on this hardware but had to be disabled because of a mismatch between the software and microcode levels.

34. Check that the timer is working properly.

35. Write a message to system console if ECPS and software are at different levels.

23.20 SPOOL FILE RECOVERY

36. Ask the operator for kind of start: WARM, COLD, CKPT, or FORCE.

37. Call DMKWRMST with the parameter provided by the operator.

23.21 ALLOCATE DUMP SPACE

38. Preallocate explicit dump space, if available, or temp space for system dump. Write a message to the system console if dump allocation fails.

39. Call DMKIOSQR to write DMKSYM into the beginning of the dump space.

23.22 FINAL PAGING INITIALIZATION

Begining at the label ALOCLP, loop through the CP-owned list at DMKSYSOW generating a system-wide count of temp pages (DMKPGTTM & DMKPGT90). Call an internal subroutine to fold the allocation map for each volume around the center of the volume (see the paging chapter). Call the internal subroutine INDXBLD, which uses the installation-specified information in DMKSYS to update the indirect lists of pointers to the ALOCBLOK chains maintained in DMKPGT.

40. Loop through CP-owned volumes.

    a) Count the number of temp pages in the CP-owned space.

    b) Call internal subroutine ALOCFOLD to logically fold the ALOCMAP about the center of the volume.

41. Calculate and set system-wide 90 percent temp page count.

42. Call internal subroutine INDXBLD to implement in DMKPTR the device search orders specified in DMKSYS.

43. Call internal subroutine RECHAIN to relink ALOCBLOKs according to the search order.

23.23 3705 INITIALIZATION

Use the 3705 addresses in DMKRIO to locate RDEVBLOKs with RDEVTYPC not equal to CLASTERM (that is, the 3705 base addresses), and mark them 'not ready' since the 3705 may not even be running.

44. Process 3705s (count in DMKRIORN), marking base addresses 'not ready'.

23.24 T-DISK INITIALIZATION

Invoke the module DMKCPX for each volume with T-disk space defined. DMKCPX clears the T-disk by stacking CPEXBLOKs that will in turn perform I/O to clear the beginning of each T-disk cylinder.

45. Call DMKCPXCK for every T-disk volume.

23.25 OPERATOR LOGON

Initialize the operator virtual machine by calling DMKLOGOP. If this call fails, then write a message to the system console asking for an explicit LOGON of the operator. Read the reply line and give the line to DMKCFMEN to be processed as a normal CP command. Upon successful creation of the operator virtual machine, write to the system console the names of the volumes in the CP-owned list that are not now online. Write the names of duplicate volumes. Write a map of main storage. Call DMKCQRFI to display the number of spool files present. Call DMKCSOSD to start the real spool devices.

46. AUTOLOG the operator virtual machine by calling DMKLOGOP.

47. If AUTOLOG fails, force explicit LOGON by prompting system console and passing command line to DMKCFMEN.

48. Report to operator names of CP-owned volumes *not* online currently.

49. Report duplicate volume names.

50. Report main storage size.

51. Report FREE, TRACE, NUCLEUS, DYNAMIC, and V=R sizes.

52. Call DMKCQRFI to report number of spool files in system.

53. Call DMKCSOSD if operator requested real spool device start.

23.26 PROGRAM PRODUCT MAP INITIALIZATION

54. CALL DMKHVDPP to initialize program product bit map.

23.27 PAGEABLE NUCLEUS PAGED OUT

Make explicit calls to DMKPGTPG and DKMRPAPT to write out to page space the symbol table module DMKSYM and the IPL simulator program DMKVMI. Invoke the TRANS macro for each page in the pageable nucleus to bring those pages into main storage. They will later be written out to the paging space due to normal system activity.

55. Page out the symbol table and IPL simulator.

56. TRANS in all pageable nucleus pages and cause DASD slots to be assigned.

23.28 CONTINUE INITIALIZATION IN DMKCPJ

57. GOTO DMKCPJNT

23.29 DMKCPJ CONTINUING INTIALIZATION

58. Call DMKIOEFL to initialize error recording.

59. Setup machine check new PSW and enable machine checks in control register C14.

60. Call DMKNLDR to load 3705 program for those 3705s that have automatic loading specified in the RDEVBLOK.

61. Mark APSTAT1 with initialization complete flag.

62. If this is a system restart, enable available 3705 lines by calling DMKCPVAE for CLASGRAF and CLASTERM. Enable NET ATTACHed lines by calling DMKNETAE.

63. Perform monitor initialization.

64. Perform IUCV initialization.

65. Call DMKDIDEP to initialize the missing interrupt handler and build a TRQBLOK for a one minute timer.

66. Unlock DMKCPI and DMKCPJ.

67. Write the final initialization message.

68. If restarting after a system ABEND and the operator was not logged onto the current system console, then disconnect the operator virtual machine with active console spooling.

69. GOTO DMKDSPCH. Whew!

23.30 SUMMARY

CP is completely initialized. Users can logon. The course is finished.

NOTES

Appendix A

CP MODULES (ALPHABETICALLY)

The following is a list of all the modules that make up the CP nucleus, with a very brief description of each module's function. This list is current as of VM/SP release 3 service level 307; modules that are new at some service level or are a part of HPO are marked as such. Modules that process CP commands are marked with the character string "cmd:" for easier identification.

DMKACO - accounting
DMKACR - channel machine check recovery
DMKALG - cmd: AUTOLOG
DMKAPI - AP/MP initialization
DMKATS - changes to shared segments
DMKBIO - DASD block I/O via IUCV
DMKBLD - build VMBLOK, ECBLOK, or paging tables
DMKBOX - 3270 VM logon picture
DMKBSC - remote 3270 line error recovery
DMKCCH - channel check error recovery
DMKCCW - virtual CCW translation
DMKCDB - cmd: DISPLAY and DCP
DMKCDM - cmd: DUMP and DUMPCP
DMKCDS - cmd: STORE and STCP
DMKCFC - command name lookup
DMKCFD - cmd: ADSTOP and LOCATE
DMKCFF - cmd: subroutines for DMKCFG
DMKCFG - DIAGNOSE X'64' and cmd: IPL
DMKCFH - cmd: SAVESYS
DMKCFJ - cmd: SLEEP, BEGIN, QUERY, REQUEST, and SET
DMKCFM - read and execute console commands
DMKCFO - cmd: privileged SET operands
DMKCFP - virtual machine reset and cmd: SYSTEM
DMKCFQ - virtual device reset
DMKCFS - cmd: non-privileged SET operands
DMKCFT - cmd: TERMINAL
DMKCFU - cmd: more privileged SET operands
DMKCFV - cmd: SET STBYPASS and STMULTI
DMKCFW - cmd: SCREEN
DMKCFY - cmd: SET ASSIST, AFFINITY, PF, and TIMER
DMKCKP - system IPL and forced shutdown
DMKCKS - spool file checkpoint
DMKCKT - subroutines for DMKCKS
DMKCKV - checkpointed file recovery
DMKCLK - AP/MP TOD clock synchronization
DMKCNS - slow-speed terminal handler

DMKCPB - cmd: EXTERNAL, READY, NOTREADY, RESET, REWIND
DMKCPE - end of CP nucleus
DMKCPI - CP initialization
DMKCPJ - continuation of DMKCPI
DMKCPK - (HPO) continuation of DMKCPI
DMKCPO - cmd: processor VARY OFFLINE
DMKCPP - online conversion from AP/MP to UP
DMKCPS - cmd: HALT and SHUTDOWN
DMKCPT - cmd: device VARY
DMKCPU - cmd: processor VARY ONLINE
DMKCPV - cmd: ENABLE, DISABLE, LOCK, UNLOCK, ACNT
DMKCPW - cmd: device VARY OFFLINE
DMKCPX - routines to clear T-disks
DMKCPZ - (HPO) cmd: VARY ONLINE for 3880-11
DMKCQG - cmd: QUERY VIRTUAL
DMKCQH - cmd: QUERY RDR, PTR, and PUN
DMKCQP - cmd: various configuration QUERY
DMKCQQ - continuation of DMKCQP
DMKCQR - continuation of DMKCQP
DMKCQS - cmd: QUERY SCREEN and MITIME
DMKCQY - cmd: more miscellaneous QUERY
DMKCSB - cmd: LOADBUF and SPACE
DMKCSC - cmd: LOADBUF and LOADVBUF
DMKCSO - cmd: more real device spooling
DMKCSP - cmd: virtual spooling
DMKCSQ - cmd: CLOSE, HOLD, and FREE
DMKCST - cmd: TAG
DMKCSU - cmd: CHANGE
DMKCSV - cmd: ORDER, PURGE, and TRANSFER
DMKCVT - various conversion subroutines
DMKCVU - floating hex to EBCDIC conversion
DMKDAD - 3375 and 3380 error recovery
DMKDAS - other CKD device error recovery
DMKDAU - FBA device error recovery
DMKDEF - cmd: DEFINE device
DMKDEG - cmd: DEFINE STORAGE and CHANNELS
DMKDEI - cmd: DEFINE for MSS
DMKDGD - DASD I/O via DIAGNOSE X'18'
DMKDGF - interrupt handler for DIAGNOSE X'18'
DMKDIA - cmd: DIAL
DMKDIB - cmd: subroutines for DMKDIA
DMKDID - missing interrupt detector and handler
DMKDMP - CP ABEND dump routine
DMKDRD - spool I/O via DIAGNOSE X'14' and X'34'
DMKDSB - DASD statistical buffer unload
DMKDSP - CP and virtual machine dispatching
DMKEIG - 2880 channel logout error recovery
DMKEMA - error messages 0 - 139
DMKEMB - error messages 140 - 423
DMKEMC - error messages 424 - 999
DMKENT - linkage between monitor routines
DMKEPS - password checking
DMKERM - error message edit and output
DMKEXT - external interrupt handler

DMKFCB - printer forms control buffer definitions
DMKFPS - fast path simulation for privops
DMKFRE - control block storage management
DMKGIO - virtual machine I/O via DIAGNOSE X'20'
DMKGRA - subroutines for 3270 support
DMKGRC - subroutines for extended 3270 support
DMKGRF - support for local 3270 and 3066 devices
DMKGRH - special routines for 3066 support
DMKGRT - more subroutines for 3270 support
DMKGRU - tables for 3278-3
DMKGRV - tables for 3278-4
DMKGRW - tables for 3278-2A
DMKGRX - tables for 3278-5
DMKHPS - logical device support
DMKHPT - subroutines for DMKHPS
DMKHVC - main DIAGNOSE handler
DMKHVD - pageable continuation of DMKHVC
DMKHVE - pageable continuation of DMKHVC
DMKIOC - subroutines for DMKIOF
DMKIOE - I/O error recording
DMKIOF - pageable continuation of DMKIOE
DMKIOG - I/O error recording initialization
DMKIOH - I/O error recording initialization
DMKIOJ - I/O error recording subroutines
DMKIOQ - (305) subroutines for DMKIOS
DMKIOS - main I/O scheduler
DMKIOT - I/O interrupt handler
DMKISM - ISAM channel program processing
DMKIUA - IUCV main routine
DMKIUC - IUCV subroutines
DMKIUE - IUCV subroutines
DMKIUG - IUCV subroutines
DMKIUJ - IUCV subroutines
DMKIUL - IUCV subroutines
DMKJRL - security journaling support
DMKLNK - cmd: LINK
DMKLOC - system resource lock routines
DMKLOG - cmd: LOGON
DMKLOH - cmd: subroutines for DMKLOG
DMKLOK - AP/MP inter-processor lock support
DMKMCC - cmd: MONITOR
DMKMCD - continuation of DMKMCC
DMKMCH - machine check interrupt handler
DMKMCI - cmd: SET MODE
DMKMCT - AP/MP processor recovery
DMKMHC - MSSF (support processor) calls
DMKMHV - virtual MSSF support
DMKMIA - monitor start and stop
DMKMID - change date at midnight
DMKMNI - various monitor subroutines
DMKMNJ - more monitor subroutines
DMKMNL - (HPO) monitor collection for 3880-11
DMKMON - MONITOR CALL interrupt handler
DMKMOO - subroutines for DMKMON

DMKMPO - (HPO) AP/MP instruction processing
DMKMSG - cmd: MSG, SMSG, MSGNOH, WNG, and ECHO
DMKMSW - I/O error message writer
DMKNEA - cmd: NETWORK ATTACH and NETWORK DETACH
DMKNEM - op-code mnemonics for trace and per
DMKNES - cmd: various NETWORK operands
DMKNET - cmd: NETWORK main routine
DMKNLD - 3705 program loader
DMKNLE - 3705 memory dumper
DMKOPR - emergency operator console writer
DMKPAG - build paging channel programs
DMKPAH - paging interrupt handler
DMKPEI - cmd: PER
DMKPEL - cmd: continuation of DMKPEI
DMKPEN - cmd: PER END
DMKPEQ - cmd: QUERY PER
DMKPER - PER interrupt handler
DMKPET - PER output writer
DMKPGM - page migration
DMKPGS - virtual memory release
DMKPGT - paging DASD slot management
DMKPGU - continuation of DMKPGT
DMKPIA - LOADBUF support for 3289-E
DMKPIB - LOADBUF support for 3262
DMKPMA - (HPO) preferred machine assist support
DMKPRG - program check interrupt handler
DMKPRV - privop simulation
DMKPRW - continuation of DMKPRV
DMKPSA - CP page 0 work area and subroutines
DMKPTR - main page fault processing
DMKPTS - (HPO) continuation of DMKPTR
DMKQCN - queue a virtual console write request
DMKQCO - queue a virtual console read request
DMKQCP - force disconnect
DMKQVM - cmd: QVM
DMKRET - support PF key retrieval
DMKRGA - remote 3270 interrupt handling
DMKRGB - remote 3270 main routines
DMKRGC - remote 3270 input decoding
DMKRGD - remote 3270 extended function routines
DMKRIO - I/O configuration tables
DMKRNH - support 370x NCP terminals
DMKRPA - explicit page read and write
DMKRSE - real spooling device error recovery
DMKRSP - real spooling output
DMKRSQ - subroutines for DMKRSP
DMKRST - real spooling input
DMKSAV - read or write CP nucleus
DMKSBL - small block letter formatter
DMKSCH - system scheduler
DMKSCN - various scan subroutines
DMKSCO - continuation of DMKSCN
DMKSEP - separator page writer
DMKSEV - 2870 channel logout error recovery

DMKSIX - 2860 channel logout error recovery
DMKSNC - save 370x control program image
DMKSND - cmd: SEND
DMKSNT - saved systems definitions
DMKSPK - spool file deletion
DMKSPL - open and close spool files
DMKSPM - cmd: SPMODE
DMKSPS - I/O routine for SPTAPE processing
DMKSPT - cmd: SPTAPE
DMKSRM - cmd: SRM
DMKSSP - starter system re-configuration routine
DMKSSS - MSS common service routine
DMKSST - more MSS support routines
DMKSSU - more MSS support routines
DMKSTA - main storage initialization
DMKSTD - resident work area for DMKSTP
DMKSTK - schedule CP work to be done
DMKSTP - various scheduler feedback routines
DMKSTR - page migration interrupt handling
DMKSVC - SVC interrupt handler
DMKSYM - symbol table for dumps
DMKSYS - system generation parameters
DMKTAP - tape I/O error recovery
DMKTAQ - (307) continuation of DMKTAP
DMKTBL - terminal translation tables
DMKTBM - more terminal translation tables
DMKTBN - even more terminal translation tables
DMKTCS - real 3800 printer initialization
DMKTCT - more 3800 printer initialization
DMKTDK - T-disk allocation
DMKTHI - cmd: INDICATE
DMKTMR - 370 mode timer simulation
DMKTRA - cmd: TRACE
DMKTRC - trace interrupt handling
DMKTRD - trace I/O operations
DMKTRK - DASD alternate track processing
DMKTRM - 2741 terminal type identification
DMKTRP - cmd: CPTRAP
DMKTRT - cmd: CPTRAP subroutines
DMKTTY - special ASCII terminal handling
DMKTTZ - resident data area for DMKTTY
DMKUCB - 3211 UCB images
DMKUCC - 3203 UCB images
DMKUCS - 1403 UCS images
DMKUDR - user directory I/O routines
DMKUDU - user directory update via DIAGNOSE X'84'
DMKUNT - CCW and CSW translation
DMKURS - real spooling device message writer
DMKUSO - cmd: LOGOFF, FORCE, and DISCONN
DMKUSP - cmd: subroutines for DMKLOG and DMKUSO
DMKVAT - shadow page table processing
DMKVAU - continuation of DMKVAT
DMKVCA - virtual CTC support
DMKVCB - continuation of DMKVCA

DMKVCH - cmd: ATTACH and DETACH
DMKVCN - virtual console SIO simulation
DMKVCP - SNA terminal support
DMKVCR - SNA terminal support
DMKVCT - SNA terminal support
DMKVCV - SNA terminal support
DMKVCX - SNA terminal support
DMKVDA - cmd: ATTACH
DMKVDC - cmd: subroutines for ATTACH and DETACH
DMKVDD - cmd: DETACH
DMKVDE - cmd: subroutines for ATTACH and DETACH
DMKVDG - DASD space allocation subroutines
DMKVDR - virtual device release
DMKVDS - cmd: subroutines for ATTACH, DEFINE, and LINK
DMKVDT - (HPO) ATTACH for 3880-11
DMKVER - process virtual machine SVC 76
DMKVIO - virtual I/O interrupt simulation
DMKVMA - shared segment protection
DMKVMC - VMCF processing
DMKVMD - cmd: VMDUMP
DMKVMI - IPL simulator
DMKVRR - (HPO) V=R virtual machine recovery
DMKVSC - V=R CCW translation checker
DMKVSI - virtual I/O instruction simulation
DMKVSJ - continuation of DMKVSI
DMKVSP - virtual spool I/O simulation
DMKVSQ - continuation of DMKVSP
DMKVSR - continuation of DMKVSP
DMKVST - virtual printer support
DMKVSU - virtual spooling subroutines
DMKVSV - virtual 3800 spooling support
DMKVSW - continuation of DMKVSP
DMKWRM - spool warm start processing
DMKWRN - continuation of DMKWRM
DMKZTD - T-disk clear routine

NOTES

Appendix B

SELECTED CP CONTROL BLOCKS

The table on the following page is a cross-reference showing all the CP control blocks that we have referenced in this book and the chapters containing those references. The chapter names are abbreviated according to this scheme:

1. DSP - dispatcher.
2. SCH - scheduler.
3. TIM - timer management.
4. IMC - inter-machine communication.
5. STO - storage management.
6. PAG - paging.
7. PGM - page migration.
8. IOP - I/O processing.
9. TRM - terminal support.
10. SPL - spooling subsystem.
11. SFR - spool file recovery.
12. CFC - console functions and CP commands.
13. MPA - multiprocessor support.
14. TRT - trace table and dumps.
15. DIR - system directory.
16. MIC - microcode assists
17. GOS - guest operating system support.
18. VMI - virtual memory initialization.
19. CPI - CP initialization.

Table of control blocks and chapters

	DSP	STCH	TIM	IMC	STO	PAG	PGM	IOP	TRM	SPL	SFR	SFC	MPA	TRT	DIR	MIC	GOS	VMI	CPI
ALOCBLOK	.	.	.	.	X	X	.	.	.	.	.	.	.	.	.	.	.	.	X
ALOFBLOK	.	.	.	.	X	X	.	.	.	.	.	.	.	.	.	.	.	.	X
BSCBLOK	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.
BUFFER	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.
CCT	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
CHXBLOK	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.
CONTASK	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.
CORTABLE	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	X	.	.
CPEXBLOK	X	.	.	.	X	X	X	X	X	X	.	.	X	X	.	.	X	.	X
DMKSYSOW	.	.	.	.	X	X	.	.	.	.	.	.	.	.	.	.	.	.	.
ECBLOK	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
FILELIST	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.
IOBLOCK	X	.	.	.	X	X	.	X	X	X	.	.	X	.	X	.	.	.	X
IUCVBLOK	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
MICBLOK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	X
MSGBLOK	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
NICBLOK	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.	.
PAGTABLE	.	.	.	.	X	X	X	.	.	.	.	.	.	.	.	.	.	.	.
PDENT	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
PDSEG	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
PGBLOK	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.
PSA	X	.	X	.	X	.	.	.	.	X	X	X	X	.	.	.	X	.	X
RCHBLOK	.	.	.	.	.	.	.	X	X	X	.	.	.	X	.	.	.	.	X
RCUBLOK	.	.	.	.	.	.	.	X	X	X	.	.	.	X	.	.	.	.	X
RDCBLOK	.	.	.	.	X	X	.	.	.	.	.	.	.	.	.	.	.	.	X
RDEVBLOK	.	.	.	.	.	.	.	X	X	X	X	.	X	X	.	.	.	.	X
RECBLOK	.	.	.	.	X	X	.	.	.	X	X	.	.	.	.	.	.	.	.
SAVEAREA	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	X
SFBLOK	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	.	.
SHQBLOK	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	.	.
SPLINK	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.
SPTBLOK	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.
SWPTABLE	.	.	.	.	X	X	X	.	.	.	.	.	.	.	.	.	.	.	.
TRQBLOK	X	.	X	.	.	.	X	.	X	.	.	.	X	.	.	.	.	.	X
UDEVBLOK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.
UDIRBLOK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.
UIPLBLOK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.
UIUCBLOK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.
UMACBLOK	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	.	.	.
VCHBLOK	.	.	.	.	.	.	.	X	X	X	.	.	.	X	.	.	.	.	.
VCUBLOK	.	.	.	.	.	.	.	X	X	X	.	.	.	X	.	.	.	.	.
VDEVBLOK	.	.	.	.	.	.	.	X	X	X	.	.	.	X	.	.	.	.	.
VMBLOK	X	X	X	X	X	.	X	X	X	X	.	.	X	X	.	X	X	X	X
VMCBLOK	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
VMCPARM	.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
VSPLCTL	.	.	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	.
XINTBLOK	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

NOTES

Appendix C

CP MACROS

The following list is a summary of the macros that are defined in CP's macro libraries. This does not include those macros which are defined locally within a given ASSEMBLE file.

1. ABEND - cause CP to terminate with an ABEND code.

2. CALL - invoke another CP routine as a subroutine. The routine will be called via SVC 8 or via BALR, according to a table contained in the macro definition. A parameter value can be specified for transmission in R2. For AP/MP systems, AFFinity can be specified to cause control to return to the calling processor.

3. CHARGE - handle the CPU timer as an accumulator of a virtual machine's CPU consumption.

4. CLRIO - generate the CLRIO instruction, X'9D01'.

5. CLUSTER - generate a NICBLOK for a remote 3270 cluster controller. This is used in DMKRIO as a part of the system I/O configuration.

6. COUNT - increment a counter using CS logic for AP/MP systems.

7. CPF - generate a DIAG X'8' sequence for executing a CP command from a virtual machine. (This appears to be unused and obsolete.)

8. DECHEX - converts a decimal number (0-15) to its hexadecimal equivalent. This is an inner macro used in the RCHANNEL, RCTLUNIT, RDEVICE, and SYSCOR macros.

9. DECMP - decrement a fullword counter using CS logic for AP/MP systems.

10. ENTER - save registers R0 through R11 into SAVEREGS at the entry point of a routine called via SVC 8.

11. EXIT - load registers R0 through R11 from SAVEREGS and issue SVC 12 to return from a routine called via SVC 8.

12. GOTO - after loading an optional parameter into R2, pass control to the specified routine. R12 is loaded with the routine's address. No return will be made.

13. GRTBLOK - define a GRTBLOK, which contains the device-specific information for each of the models of 3277 and 3278 terminals. This macro also defines many symbols used in the 3270 support routines.

14. HEXDEC - convert a hexadecimal digit into its decimal equivalent. This is an inner macro used by the RCHANNEL, RCTLUNIT, and RDEVICE macros in DMKRIO.

15. IPTE - (HPO - in member 'ISKE') generate the IPTE instruction X'B221'.

16. ISKE - (HPO - in member 'ISKE') generate the ISKE instruction X'B229'.

17. IUCV - generate code to invoke various IUCV functions. For use in a virtual machine, this macro generates the X'B2F0' instruction. For use in CP, this macro generates a CALL to DMKIUA.

18. JPSCBLOK - generate the "journaling and password suppression control block". This is a DSECT unless it is in the DMKSYS CSECT.

19. LOCK - generate code for AP/MP systems to obtain or release a lock word.

20. MAXDV - generate code to place the maximum virtual device address (X'5FF' or X'FFF') into a register. Used in various routines to check the validity of a virtual device address.

21. MSG - generate code to place length, address, and parameter values into R0, R1, and R2. Used in various routines to generate messages that do not have the standard CP heading string (DMK...).

22. NAMENCP - generate a named system entry in DMKSNT for a 370x control program.

23. NAMESYS - generate a named system entry in DMKSNT for a saved system.

24. NAME3800 - generate a named system entry in DMKSNT for a 3800 definition table.

25. PAGTBL - generate a page table for a private or shared segment. (This appears to be unused and obsolete.)

NOTES