File No. S370-20
Order No. GC28-2003-6

Systems

IBM Time Sharing System Concepts and Facilities

This publication provides an introduction to the IBM Time Sharing System (TSS), a general purpose operating system used with IBM System/370 computers that have dynamic address translation. TSS allows many users to have simultaneous access to a computing system. The combination of machine and control program creates a data processing environment for each user which can be utilized independently or shared with other users. Each user operates in a separate virtual address space potentially as big as the addressing capability of the machine.

The design of TSS facilitates program development, because the functions provided support a convenient, interactive programming environment. Programs developed in this environment may be used in production mode without change. Integration of virtual storage with data management permits an innovative and productive approach to data base applications. A feature of TSS is user ownership of data with security and privacy.

This publication is written for managers of data processing installations, system programmers, application programmers, end users of applications, and operators. It is an introduction to the purpose, design, and use of TSS. It contains general descriptions of the control program, task management, and data management; a summary of publications relating to TSS; and information about use of the system to support individual users and subsystem development projects.

There is no prerequisite reading for this publication. However, the reader should have a basic understanding of IBM data processing techniques.

Summary of Amendments

Changes since the last edition (GC28-2003-5) include:

- Division into three sections, (1) an introduction to TSS concepts, (2) a description of external characteristics of interest to application programmers (users), and (3) a description of internal structure of interest to system programmers
- More emphasis on how TSS provides ownership of data, control of access to data, and data set integrity, and their relation to security and privacy of data
- Description of new support for System/370
- Discussion of hardware and software tools which increase programmer productivity
- Explanation of how named, disconnectable segments of virtual storage can be utilized to provide an address space larger than the addressing capability of System/370 hardware
- Description of a facility which makes it possible to run most language processors and a subset of programs written for OS/VS

Seventh Edition (April 1978)

This is a major revision of GC28-2003-5 and makes that edition obsolete. This edition applies to Release 3.6 of TSS/370 and to all subsequent releases unless otherwise indicated in new editions or Technical Newsletters.

Changes or additions to this publication will be provided in Technical Newsletters or, if changes are extensive, in a new edition.

Requests for copies of IBM publications should be made to the IBM representative or branch office serving the reader's locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System – Department 80M, 1133 Westchester Avenue, White Plains, New York 10604. Comments become the property of IBM.

© Copyright International Business Machines Corporation 1967, 1968, 1970, 1971, 1978.

Preface

This publication is intended to give the reader an introduction to the Time Sharing System (TSS), an appreciation of how applications can be developed and used in the TSS environment, and information about the potential benefits which may accrue because of the interfaces, functions, program development aids, and services provided. It is written for people in management who want to know more about how TSS supports the development of complex software subsystems in the areas of interactive computing, information management and retrieval, realtime processing, and batch processing. System programmers, application programmers, and operators may use this manual as an introduction to the system and the related publications. End users of applications who have no concern for the internal structure of TSS may use the information to gain an appreciation of what the computing system does to support their applications.

When TSS was introduced, the concept of a programmer using a terminal to communicate with a computer was of major interest. Accordingly, previous editions of this publication emphasized a conversational mode of use. However, the use of TSS in nonconversational mode is no different from conversational mode, except that instead of a terminal, communication is by means of a data set and there is no attention key. Therefore, in the present edition, a distinction is made between conversational and nonconversational modes only when necessary.

Section 1 introduces TSS and the concepts which are the basis of its design. Section 2 describes the external characteristics of TSS and is intended for application programmers. Section 3 presents the internal structure of TSS and is intended for system programmers. Appendixes A, B, and C summarize TSS commands, macros, and publications.

Because this publication is intended for an audience with varied data processing experience, some terms, particularly those related to TSS, may be unfamiliar to the reader. The first occurrence in each section of terms in the glossary is printed as: ***term.*** Data set names, member names, USERIDs, and source coding examples are printed as: NAME.

There is no prerequisite reading for this publication, but it is recommended that the reader have a basic understanding of IBM data processing techniques and an understanding of what an operating system is called upon to do.

Two publications in the TSS series which describe the system in more detail are *IBM Time Sharing System: System Logic Summary*, GY28-2009 and *IBM Time Sharing System: Data Management Facilities*, GC28-2056.

Contents

Figures

Introduction to TSS

General Description

TSS is a general purpose operating system which was designed for use with IBM computers that have ***dynamic address translation.*** The combination yields a computing system which can serve both ***conversational*** users, who interact with the system and their programs using terminals, and ***nonconversational*** users, who cause programs to be run by the system in a mode which is similar to ***batch processing***. TSS serves many simultaneous users with varied training, experience, and objectives. This includes scientific users, commercial users, ***text processing*** users, programmers, programming support personnel, and users of realtime applications.

The system comprises a ***supervisor*** and ***task*** monitor, ***service programs, user programs, supporting programs,*** and a support ***subsystem.*** The supervisor and task monitor control operation of the system and create the operating environment for users. The service programs perform ***task management*** and ***data management*** in response to user and system requests. Some programs supplied with the system run in the same mode as user programs, providing language processing, link-editing, and functions needed by user programs. The supporting programs are concerned with maintenance of the system. The support subsystem facilitates problem determination, remote maintenance, and system program development.

TSS supports single processor, attached processor, and multiprocessor System/370s.

Design Objectives

This section presents a number of concepts which are the basis of TSS design. The TSS implementation of these concepts is intended to make computers easier to use and to help people be more productive by:

- Reducing the complexity involved in preparing a program for execution. This allows an iterative approach to program development, leaving more time available for concentration on the problem instead of on detail related only to the computing system.

- Allowing users to interact with the computer during program preparation and execution.

- Placing the problem solver in direct association with the computer. Much clerical work associated with problem-solving, such as routine calculations, reducing and plotting data, ***editing,*** and information retrieval, can be carried out more effectively using a computer, especially if the problem solver can be close to the computer.

- Enhancing feedback throughout the design, development, and test phases of a project. This can raise the quality of design and increase the quantity of product (improve programmer productivity).

- Making it more convenient for a user with limited knowledge of computing systems to use the computer.

Resource Sharing

The users of TSS can share resources and data without the involvement of anyone else. The data that a user creates is owned by the user, not the system. TSS is designed to prevent a user from gaining access to the data owned by another user without the owner's explicit permission, which is given by means of a command. The owner can withdraw permission at any time. Privacy is not dependent upon passwords associated with each ***data set*** requiring protection. Access to a data set is controlled by specifications contained in a catalog belonging to the owner. The owner can specify the type of access sharers may have. If a sharer has read-only access to a data set or group of data sets, a data set may be read but not changed; if read/write access, read and changed; if unlimited access, read, changed, erased, and created. Owners can permit each sharer different access, as needed.

Program sharing extends beyond sharing of the data sets in which the programs are stored. If a program is ***parallel reenterable,*** it can be used simultaneously by more than one user. There need be only one copy of the program in main storage and more than one central processing unit (CPU) can be executing instructions of the program. The supervisor, task management, data management, command system, supporting programs, and the ***language processors*** native to TSS are all shared in this manner.

Users can arrange to share portions of their ***address space*** with each other. Using system services, they can synchronize use of shared address space. Because sharing is accomplished by connecting a shared area only to those users requesting it, the sharing is not apparent to other users, and does not infringe on the address space available to them.

Interactive Computing

Interactive computing is characterized by a high rate of unanticipated human decision-making interspersed with relatively short computer processing times. ***Noninteractive computing*** is characterized by preplanning of a relatively long, logically uninterrupted process. Actually, these two cases are ends of a spectrum representing computing activity. Data entry, editing, and data set manipulation can be done interactively, but do not involve much unanticipated decision-making.

TSS design helps to greatly reduce the need for preplanning, which tends to impede progress during the problem-solving or trial phase. The design provides for ***late binding*** of programs and data and ***dynamic allocation*** of computer resources. Much attention is given to keeping a user from getting into situations where the only recourse is to back up and repeat previous work.

TSS design is strongly influenced by the needs of interactive users. This can result in benefits for noninteractive processing too. TSS creates an environment which effectively supports interactive development of programs and subsequent noninteractive execution of these programs in production. The resources of the data processing system are easily accessible. Terminals provide a direct means to monitor and control programs and processing on an individual basis. TSS provides an effective person/machine communication that facilitates step-by-step interaction with programs that depend on immediate human judgement for timely solutions to complex problems.

Interactive computing also strongly fosters the concept of user identity. In between sessions with the system, users surrender custody of their data, but not ownership.

Relatively little training specific to TSS is required to use the system. Productive work can begin after learning (1) the procedure for terminal operation, (2) a control language to communicate with the system, (3) how to use the editor and, in some cases, a problem-oriented language. Because TSS has conversational capabilities, a user acquires operational skills through hands-on experience. The system provides guidance in the form of messages that indicate errors and action taken. Users can interact directly with programs and the system. It is possible to interrupt execution, obtain intermediate results, introduce new data, and change the sequence of execution.

Noninteractive Computing

The term noninteractive computing includes traditional batch processing but leaves room for all types of processing in which humans do not take an active, decision-making role. As interactive computing grows, the need for effective noninteractive processing still remains.

Some programs which users develop conversationally are to be run repeatedly in production. A feature of TSS is that programs and ***command procedures*** developed interactively can be used in nonconversational mode without alteration. The same command language handles both modes. Nonconversational tasks can be initiated by interactive users at terminals.

TSS can also process jobs in a conventional batch mode. TSS supports batch processing initiated centrally and at remote locations. At the central site, card decks are entered and output is obtained using local unit-record equipment. Remote job entry stations, connected to the central site by telephone lines, can be used at distant locations.

Another type of noninteractive processing supported by TSS is ***online computing,*** for which response time is an essential element. An example is control of experiments, where data originates at a test cell and is transmitted to a program for immediate analysis. Also, control signals can be returned to the test cell to operate the experiment.

Time Sharing

In TSS, there is a unique task associated with each user. The system allows many users to share the CPU(s) by dispatching an available CPU for a scheduled amount of time determined by task characteristics. An installation can provide service to users on both demand and scheduled bases. Scheduling strategies are specified in a table which can be set up to provide different classes of service. The scheduler recognizes changing characteristics of tasks, and provides service according to criteria specified in the table for each class of work. Installations can design a schedule table that concurrently defines different kinds of service for various users.

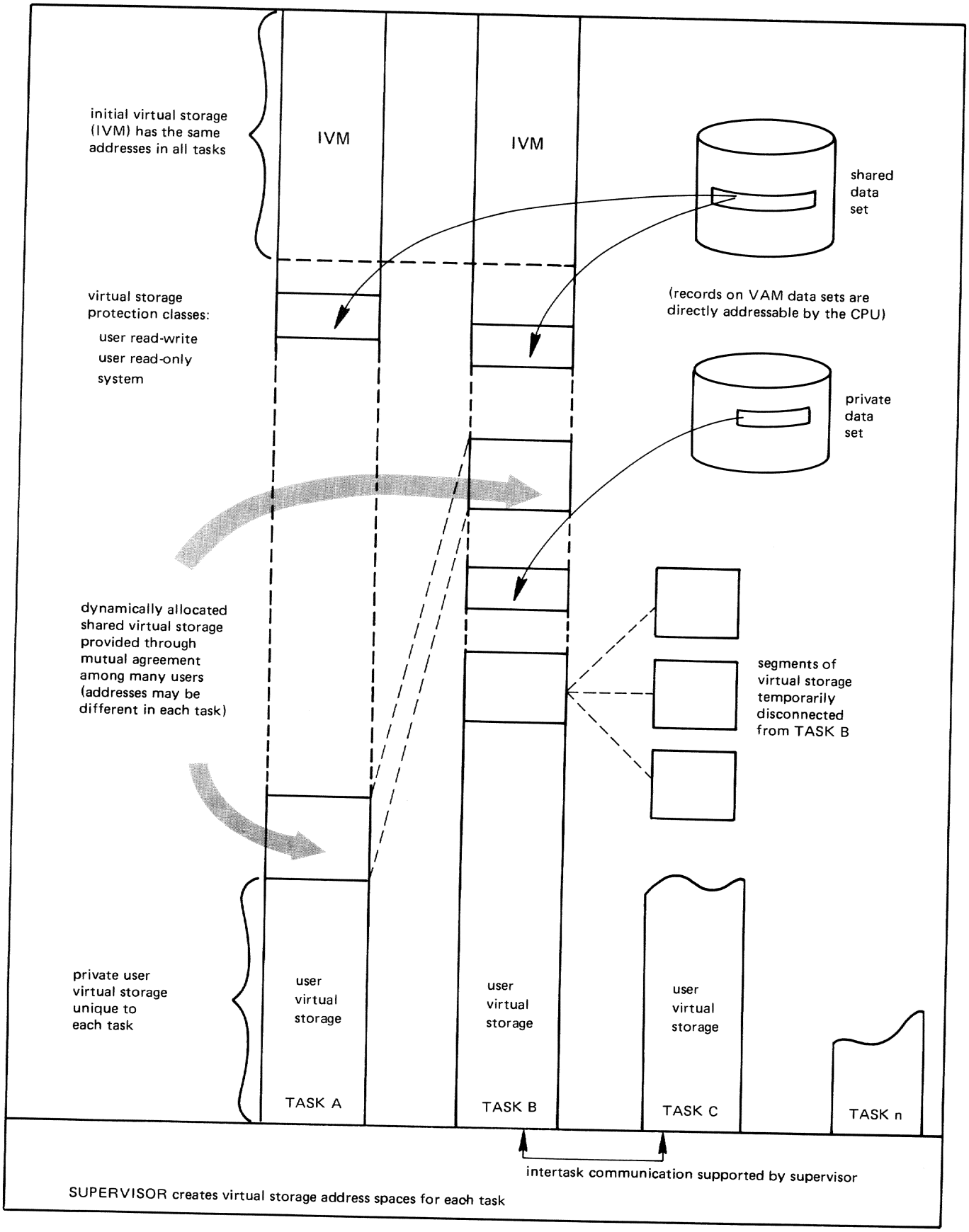

Figure 1-1. TSS Virtual Storage

Virtual Storage

Virtual storage is address space that can be referenced directly by a CPU equipped with a dynamic address translation feature. This address space appears to the user as ***real storage,*** but it is actually a combination of main storage and ***auxiliary storage.*** A virtual address space can be as large as the ***addressing capability*** of the CPU; it is not limited by the size of main storage.

In TSS, each user operates in a separate virtual address space with an independent addressing structure. The sum of all address spaces in use is limited to the amount of auxiliary storage available on direct-access devices.

Virtual storage is implemented by a process called dynamic address translation. Instructions of a program in virtual storage must be brought into real storage before execution. Virtual addresses of instruction operands must be translated to the corresponding addresses in real storage. In TSS, translation is accomplished by a combination of hardware and software outside the user's virtual address space. Virtual storage is managed in blocks of 4,096 bytes, called ***pages.*** Addresses are relocated by a value that is a multiple of the size of a page. When a page is referenced, it is brought into real storage; when it is not actively used and real storage is needed by another task, it is written to auxiliary storage unless an exact copy already exists on auxiliary storage (that is, the page was not modified during use).

This simplifies program design, because the available address space is large. Variable-size data structures are easily handled by managing virtual storage in a manner that leaves enough contiguous storage to accommodate expansion. Complete disregard for locality of reference may lead to heavy use of system resources to accomplish the required ***paging***, but it may be worth the cost for problems in which prediction of address reference is very difficult. Each TSS user can issue commands to determine how much paging is being done during any period of execution, which facilitates selection of appropriate strategies.

The size of a TSS user's virtual storage can exceed the addressing capability of the machine. ***Segments*** of virtual storage may be named and disconnected when not needed. The named segments may be reconnected as needed. Large amounts of virtual storage can be referred to as quickly as the system can switch the translation table pointers to reconnect the disconnected segments. For those applications which do not need to address all their data simultaneously, this may be more convenient than conventional input/output (I/O).

Virtual Access Method

In TSS, the principal method used by system and user programs to gain access to data is integrated with the method used to create virtual storage. The ***virtual access method*** (VAM) ***maps*** data on ***external storage*** to virtual storage. In the TSS context, the term *external storage* means permanent data storage. VAM manages the association of ***logical I/O*** with ***physical I/O,*** freeing programs from concern with the physical ***data base*** in the same way that virtual storage frees programs from the constraints of main storage. With the exception of a small part of the supervisor, programs need not be complicated by the need to satisfy the requirements of storage devices with varying characteristics. Thus, VAM is device-

independent and easily adaptable to various direct-access devices. The supervisor handles paging, data set requests, and error recovery with the same software.

VAM supports sharing of data in a convenient and secure manner and makes possible safe, concurrent update of a data set by multiple users (or programs). Tables in dynamically obtained shared virtual storage contain control information needed to synchronize updates and maintain integrity of the data sets.

For VAM, the supervisor can use the ***storage key*** reference and change bits to eliminate unnecessary I/O operations. ***Records*** adjacent to records just processed may already be in virtual storage and even in real storage or perhaps on a paging device capable of more rapid access than the device on which the records reside permanently. Direct-access storage ***volumes*** used for VAM are formatted in fixed, page-size blocks, which reduces the time for typical processing.

An important design point of TSS is that all storage is ***page-addressable.*** VAM defines dynamically changing relationships between pages in virtual storage and pages on external storage. Using VAM, data on external storage can be directly addressed by the CPU.

Dynamic Loader

Programs to be run under the control of TSS are loaded into virtual storage by a ***dynamic loader.*** The dynamic loader enhances the interactive programming process, because it makes possible late binding of programs. It is integrated with VAM and the paging mechanism, obtaining programs to be loaded from ***members*** of VAM ***partitioned data sets. Address constants*** are not resolved by the dynamic loader unless the page in which they are located is referred to by a program during execution. This automatically eliminates unnecessary processing. Reference is detected by the ***paging supervisor,*** which treats storage unprocessed by the dynamic loader in a special way.

Because programs are not bound to each other until actually needed, large program bases can be utilized conveniently. Loading is dynamic, which means that ***binding*** occurs at execution time. Linkage can be data-directed and subsystems can be open-ended. This facilitates data base applications in which the names of programs appropriate to the processing of elements in the data base are stored with the elements. In this arrangement, data is the driving force. The data and the user direct the problem-solving process. If a new technique (program) is added to the subsystem it can be used to process the data without any impact to the old method and with only a small change required of the user.

Implicit loading is possible, and users can direct substitution of programs by arranging lists of libraries containing various versions of the programs. The dynamic loader also provides a dynamic call-by-name facility for ***explicit loading.*** All ***entry names*** in programs are available for resolution when loading subsequent programs.

Because link-editing is optional, subsystems containing numerous subroutines can be used by many users and yet simultaneously undergo maintenance. Benefit from the system sharing facilities is inherent, because access to programs is under the same control as access to all shared data.

In summary, advantages of virtual storage, VAM, and a dynamic loader are:

- Programs can be designed with little concern for main storage capacity.
- Only actively used portions of programs and data areas occupy main storage; this is managed automatically by the system.
- I/O programming is simplified for both system and user programs; the dynamic address translation hardware and the system control program are integrated in their support of I/O. Programs need not be concerned about the extent of data sets; space allocation is managed automatically. Also, operations are based on logical records, not physical characteristics of storage devices.
- Programs using VAM data sets are not dependent on particular device types or system configurations.
- The dynamic loader eliminates the need to link-edit programs (but it may be desirable to link-edit specific production programs for reasons of improved performance). It is possible to load and unload programs, while testing, without affecting the entire collection of programs.
- If a ***control section*** has the ***public attribute***, it is loaded in a ***shared segment*** of virtual storage. Shared segments have common address translation tables. Requests for the same public control section are satisfied by connecting to the shared segment in which the control section is loaded. The dynamic loader uses the name of the data set from which the ***object module*** was loaded to verify that the control sections match. If a control section with the public attribute is being actively used by more than one task, it is probable that it will be in main storage. Thus, only one copy is needed for several otherwise independent address spaces. This is handled automatically by the system, relieving programmers of the need to take specific action to achieve the economies which result.

Data Management

Those system services which provide for storage and access to data are called data management. They free programmers from details of device programming and lead to efficient utilization of equipment. Data is logically grouped in data sets. A data set is a named collection of related records. The data set name is used whenever the data set is processed. Examples of data sets are: files used by programs, libraries of programs, and, in a special sense, streams of records read from or written to devices such as card readers, printers, punches, and terminals. Programs process data through a logical connection to the data set name rather than to the name (description or address) of a device. TSS provides a catalog which makes it possible to refer to data sets without specifying physical locations or logical characteristics. A cataloged data set may be referred to by name only; the descriptive information is recorded in the catalog.

The term *volume* refers to a standard unit of auxiliary storage such as a disk pack or a reel of tape. The volume serial number identifies the specific disk pack or reel of tape on which the data set resides. A ***public volume*** can be used by many users concurrently, and any user of TSS may

have data on it. Public volumes are owned and managed by the system. Users need not refer to specific volumes or make specific allocation requests related to space. All data sets on public volumes are cataloged; the user need not remember locations of data sets. A ***private volume*** is dedicated to one user at a time. The system operator, aided by the system, authorizes use of private volumes.

All direct-access volumes have standard ***volume labels*** and ***data set labels*** which the system creates and maintains. There can also be user data set labels on volumes formatted for the basic ***sequential*** access method (BSAM). These labels can be processed by user-written routines. Tape volumes may contain (1) standard volume and data set labels, or (2) standard volume and data set labels plus user data set labels, or (3) no labels. Labels on tape volumes with standard labels are created and maintained by the system; user data set labels are processed by user-written exception routines.

TSS for the Application Programmer

This section describes TSS for application programmers. The work performed by application programmers may range from writing programs for a single user to developing an application that supports many users. TSS defines a strict interface between system and the application programmer, who operates on the ***nonprivileged*** *side of the interface. This definition establishes the integrity of the system. The system is designed so that no user can gain access to information belonging to any other user without the owner's permission. This interface has not changed significantly since it was first defined. For example, although the capacity and functional characteristics of the principal data storage devices have undergone major change, programs which use the primary* ***access method*** *continue to function without need for reprogramming while taking full advantage of new device function.*

(System programmers use the system in the same manner as application programmers, except that their work usually produces alterations and extensions to the system. This section should be used by system programmers as an aid to understanding and preserving the system/user interface.)

Access to the System

Prior to using the system, one must get an identification code, called a USERID. The USERID defines a user to the system. The management of the computer center assigns an initial password, charge number, external priority, privilege, authority, and resource usage limits for each user. One who has access to the system is said to be *joined* to the system. When TSS is delivered, it has joined to it a system manager (SYSMANGR), a system operator (SYSOPER0), and an owner of system ***data sets*** (TSS). Using the JOIN command, the system manager can join system administrators to the system. The system administrators, in turn, can join users to the system. SYSMANGR can also join users to the system. This two-level approach allows computer center management to divide responsibility among departments.

The following are JOIN command parameters:

USERID: All processing by the system on behalf of a person is related to a USERID. Ownership, sharing of data and programs, and communication between users is based on the USERID.

PASSWORD: Access to TSS is based on knowing the password to a USERID. Users can change the password with the CHGPASS command.

CHARGE: Specifies an account number against which charges are accrued. TSS includes facilities for collection and presentation of raw data covering use of most system resources. The installation is responsible for accounting and resource allocation according to individual needs.

PRIORITY: Specifies a set of scheduling parameters for ***tasks*** belonging to the user. TSS is supplied with a general-purpose schedule table containing several sets of parameters.

PRIV: (Privilege) Controls which commands a user may issue. A few privilege classes are used by the system and the remainder are available for use by the installation.

AUTH: (Authority) Indicates which supervisor calls (SVCs) may be issued by user-written programs and whether the user may display and/or alter system programs. Authority also controls some special features of data set sharing and affects program loading related to maintaining system security.

RATION: Specifies which one of a set of installation-defined limits is to be applied to instantaneous and cumulative usage of resources, such as ***CPU time*** and ***online storage.*** The system records usage of resources by each USERID and rejects requests for resources that would cause the limits to be exceeded.

BATCH: Specifies that a user may submit batch work to the system at the central computer installation.

RJE: Specifies that a user may submit batch work and receive printed output using a remote job entry station.

The right of any user (except the system manager, the system operator, and USERID TSS) to use the system can be revoked by the corresponding administrator or the system manager. The QUIT command removes a user from the system and provides for appropriate disposition of the user's data sets. Users can be temporarily denied access to the system by reducing certain allowable resource allocations, such as ***connect time,*** to zero.

User/System Communication

TSS has different kinds of users. One kind of user is a person; another user could be a ***subsystem.*** For example, a USERID could be set up to compile and execute student jobs at a university computing center. The person responsible for the USERID associated with the subsystem accounts for and controls the work performed by the subsystem.

LOGON

Equipped with a USERID, a user uses the LOGON command to create a task. (This discussion applies equally to ***conversational*** and ***nonconversational*** use of the system.) The LOGON command is not a command in the normal sense. It has operands, such as password, ***addressing mode,*** and ***control section packing*** options, but its purpose is to cause a task to be created. A task has a unique identification number, called a TASKID. The TASKID is used by the system to keep track of resources utilized by the task. The TASKID is used when communicating between tasks. The task exists until the LOGOFF command is issued.

LOGON causes the system to connect the user to a number of entities. For purposes of computing, the most important is the presence of an ***address space*** for programs and data in which the task receives service from a CPU.

Another important entity provided by the LOGON process is connection to a catalog containing names and information about data sets to which the user has access. The system associates users with their data sets and

those to which they are allowed access by means of the catalog. The catalog as it appears on ***external storage*** is like any other TSS data set, but only system service routines have access to the catalog. The catalog is actually a combination of many data sets. There is a separate data set for each user's catalog and for the master catalog into which a user's catalog is copied for use. This technique provides for redundancy and security, with efficient access.

Each USERID normally owns a data set named USERLIB (user library). The user library can be used to store ***object modules*** and contains a user profile described below. LOGON connects the user's task with a number of system data sets, whose names need not appear in the user's catalog. One of these is a data set in a ***generation data group*** named SYSLIB (system library). The system library, along with the user library, is defined as a ***job library*** (JOBLIB).

Another entity to which the task is connected is the system operator. Users and programs can communicate with the system operator. An example of communication is a request to mount a private tape or disk.

User-Specified Profile

An important function of LOGON processing is to enable each user to set up an environment according to individual needs. During LOGON, a standard user profile (possibly modified by the installation) is used to construct a ***combined dictionary*** for the task. Users can change their copy of this dictionary with DEFAULT, SET, and SYNONYM commands or the SETDV macro, and save the changed copy with the PROFILE command. For subsequent LOGON commands, the system obtains the combined dictionary from the saved user profile. As a final step in the LOGON process, the system executes a ZLOGON command. The system ZLOGON command is null and can be overridden by a user command, with the same name, to cause user-specified initialization to be performed.

Users can customize the system by changing the user profile. Facilities exist for:

- Renaming commands and operands
- Altering the ***default values*** for command operands
- Defining variables, called ***command symbols,*** for use during command execution
- Creating new commands
- Rewriting system messages
- Modifying terminal input and output translation tables
- Redefining the terminal function control characters used to edit input and output
- Saving the changed user profile so that it can be reestablished the next time the user logs on.

Command System

When LOGON is complete, the system reads a ***record*** from SYSIN to get the user's first command. The part of TSS that reads is called the ***command system*** and SYSIN is the data definition name (DDNAME) for the command system input data set. Output is directed to SYSOUT, which is the DDNAME of the command system output data set. (Data sets, DDNAMEs, and the means by which programs access data are described under "Program/System Communication" in this section.) The command system is used to invoke programs, which can read from SYSIN and write to SYSOUT, using a simple device-independent interface.

The fact that SYSIN and SYSOUT are DDNAMEs of data sets is not important to users. For conversational tasks, SYSIN normally is what the user enters and SYSOUT normally is what the system writes back. For nonconversational tasks, SYSIN is a prestored data set containing commands and data; SYSOUT is a data set which is usually printed and erased. Each time a command from SYSIN completes, or a program returns to the command system, another record is read from SYSIN. Commands entered from the terminal are treated in the same way as commands in nonconversational SYSIN data sets. (The task is terminated with a LOGOFF command or by reaching the end of the SYSIN data set.)

Users get work done by making requests to the system in the form of commands. The term *command*, as used in connection with TSS, implies more than a request entered at a terminal which the system must interpret and execute. All processing performed on behalf of a user (conversational or nonconversational) is the result of commands. A program is caused to run by a command or a call from another program. A command can be issued from within a program. Appendix A is a summary of commands supplied with TSS, organized by functional area. Commands belong to certain categories:

- ***Task management*** commands initiate and terminate tasks and display task-related statistical information.
- Command environment commands adjust the apparent characteristics of the system to suit individual needs and specify action to be taken upon interruption of programs and command sequences.
- Terminal control commands affect the operating mode of the terminal related to the handling of records to/from SYSIN/SYSOUT.
- Program execution commands provide the means for testing and running programs.
- ***Data management*** commands help users use data.
- Editor commands create, modify, and delete records in data sets which can be processed by the TSS editor.
- Data ***editing*** commands support entry, display, and alteration of data sets.
- Language processing commands invoke assemblers and compilers, which produce executable object modules from source programs.

- Bulk output commands print and punch data sets locally, print data sets at remote stations, and write data sets on tape for offline printing.

- Operator, manager, and system programmer commands support operation, control, and maintenance of the system.

- Time sharing support system commands, for use by system programmers, enable independent, interactive testing of the system or an individual task.

There is a difference between conversational and nonconversational tasks in one respect; with conversational tasks, a user at a terminal can signal an ***attention interrupt,*** making it possible to examine intermediate results, issue other commands, and cause execution of an interrupted program to be resumed. The command system normally handles attention interrupts, but a ***user program*** can specify an interrupt routine which takes precedence and receives control when there is an attention interrupt.

If a user program uses the SYSIN macro to read from SYSIN, there can be escape to the command system and other programs during a read, without the need to signal attention. The program may in fact be reading from the expansion of a ***command procedure*** definition or from a string of commands obtained from one record of SYSIN. A user-definable break character indicates that the input which follows is intended for the command system and not the user program. A TSS program can detect such escape to the command system.

Interruption of user programs and execution of new ones is made possible by a pushdown stack for the saved status of each program. Also, because there is ample ***virtual storage,*** the command system and user programs do not use transient routines (which typically are loaded into the same storage locations), thereby avoiding problems which could result upon resumption of execution. Furthermore, TSS data management is designed to get each ***save area*** dynamically. This is practical, because a user program can call a routine such as GET, whose execution may be interrupted. During the interruption, another user program in the same address space can use GET without affecting the interrupted GET operation. There are limits: obviously, the state of a data set being processed by an interrupted program could be changed by execution of other programs acting on the same data set.

It is convenient to be able to leave one command environment temporarily and enter another. Horizontal integration (all commands usable at all times) and vertical integration (subsetting of environments) are both realizable with the TSS command system. For example, an application program, such as an editor, can read a subset of commands, unconstrained by the general command syntax. The user can escape from that environment and use any command outside the subset, possibly entering other subset environments.

Editor

The TSS editor is invoked by the EDIT command. It is capable of editing most of the data set formats used by the system. (For other formats, there is another editor, which is invoked with the MODIFY command.) The TSS editor also can be invoked from programs. An example of this is the TSS PL/I F compiler which uses the editor to create source data sets. Editing is ended with the END command which

closes the data set being edited. An EDIT command, issued while editing, closes the current data set and begins editing the data set named in the command.

The TSS editor edits data sets having the format most frequently used by ***language processors*** native to TSS. These data sets have variable-length records with a maximum length of 132 bytes. Each record contains, in order, a 4-byte length field, a 7-byte numeric key, called the ***line number,*** a code byte that indicates the origin of the data (card reader or keyboard), and a data area. Such data sets are called ***line data sets.*** Origin of the data, as indicated by the code byte, is important to programs which accept different continuation conventions for data records. TSS language processors allow free-form input for lines entered at a keyboard, thus eliminating the need for users to space over columns exactly. Line data sets can be made to appear as card-image data sets by the interface that supports execution of OS/VS programs.

A ***region data set*** is an extension of the line data set format. This format is used by the system for command procedures and messages. The keys consist of an 8-byte portion, which defines a ***region*** of the data set, followed by the line number. As with line data sets, the first data byte in the record is the code byte. Records may be up to 256 bytes long. The REGION operand of the EDIT command identifies the portion of the data set which is to be processed by editor commands as if that region were a line data set. Although not presently utilized for system data sets, keys (region name plus line number) can be larger than eight bytes, up to a limit determined by the record length.

The editor optionally maintains a transaction table in which changes are recorded. Additions and deletions are recorded separately. The transaction table facilitates nullification of changes to a data set being edited.

The STET command causes all changes that are recorded in the table at the time the command is issued to be reversed. Additions recorded in the table are deleted from the data set. Deletions recorded in the table are added to the data set. Changes made by the STET command are also recorded in the table; in effect, the addition and deletion portions of the table are switched.

The POST command clears the transaction table of the entries recorded in it, thus making those changes permanent.

The ENABLE command causes each subsequent command that alters data to clear any previous entries to the table, so that only changes made by that command will be recorded. While the ENABLE command is in effect, each command that alters data leaves only the changes it has made in the transaction table.

The DISABLE command cancels the effect of a previous ENABLE command. While ENABLE is in force, the editor is said to be enabled; while DISABLE is in force, it is said to be disabled. When transactions are to be recorded, the editor is initially disabled.

Editor commands are summarized in Appendix A.

User-Written Commands

Users can create their own commands to supplement the commands supplied with the system. A user-written command can issue system and user commands contained within it. A command can also be a program, invoked as if it were a command.

The PROCDEF command can be used to write a command procedure definition (also called PROCDEF). It invokes the editor, which is used to write the PROCDEF. When writing a PROCDEF, provision can be made for parameters which are to be operands of the PROCDEF. When the PROCDEF is issued, statements in it are expanded and filled with these parameters. The parameters can also be used to control execution or they can be passed as operands to commands contained within the PROCDEF.

The BUILTIN command and the BPKDS macro make it possible to invoke a program as if it were a command, obtaining the services of the command system for delivery of parameters. The program is invoked by a command (also called BUILTIN) defined by the BUILTIN command.

User-Modified System Messages

During the course of execution, the system issues messages to SYSOUT. The user can write messages to replace many of those issued by the system. User-written messages are in the user library; system messages are in the system library. A user-written message has the same message identification code as the system message it replaces. The system searches the user message file before searching the system message file to get a message; therefore, messages in the user message file take precedence over messages in the system message file.

The user profile contains a filter value to indicate the *level* of messages desired. Messages are classified by severity and length. There are five levels of severity: information, warning, normal error, serious error, and termination error. By default, all messages except information-level messages are written, but the filter value can be changed. There are five levels of message length: message identification code only, standard messages with message code, standard messages without message code, extended messages with message code, and extended message without message code. By default, users receive standard messages without the message identification code.

An explanation of messages and terms in the system message file can be requested with the EXPLAIN command. This command can be used to clarify a message, words in a message, responses the user may make to a message, and the origin of a message.

Conversational Language Processors

The language processors native to TSS (TSS assembler, TSS FORTRAN, and TSS ***linkage editor)*** can operate in a mode in which statements are analyzed for syntactical correctness, line by line, with errors reported immediately. If an error in a source statement can be corrected during language processing, the language processor can be utilized to update the source program within the context of the language processor, providing

immediate diagnostic aid. The processors are invoked by the language processor controller (LPC). LPC is a facility which can also be utilized by an installation to control locally developed processors.

Symbolic Libraries

Symbolic libraries supplement the capabilities of ***partitioned data sets.*** An example of a symbolic library is an assembler macro library, which can contain macros, DSECTs, and source code. Symbolic libraries can be used with applications other than the assembler.

A symbolic library has a symbolic component and an index component. The symbolic component is either a line data set, containing records grouped into ***parcels*** by a unique header plus name, or a region data set, with regions as parcels. The index component is produced by the SYSINDEX command and contains pointers from names and aliases to all portions of the symbolic component that are to be used in library references. The symbolic library search routine (SYSEARCH) may be called by user programs to locate parcels of a symbolic library using the index created by SYSINDEX.

Copying Data Sets

The CDS (copy data set) command can copy a data set or a ***member*** of a partitioned data set. Also, it can be used to renumber the lines of a line data set. The organization of the copied data is determined by the definition of the output data set. A ***sequential*** or ***indexed sequential*** data set may be copied into a partitioned data set, and a member of a partitioned data set may be copied out of the partitioned data set, becoming a sequential or indexed sequential data set. Provided that it has valid keys in ascending sequence, a sequential data set or member may be copied into an indexed sequential data set or member. An indexed sequential data set or member can always be copied into a sequential data set or member. A user with unlimited access to a data set that resides on direct-access storage can optionally specify that the data set or member be erased after it is copied.

VT (VAM-to-tape), TV (tape-to-VAM), and VV (VAM-to-VAM) copy data sets on a ***page*** basis, which is more efficient than copying by logical records, because the internal structure of the data set is ignored. The data sets must reside initially on direct-access storage in VAM format. VT writes a VAM data set on tape as a physical sequential data set in a format meaningful only to the TV command. TV reads the data set written by VT and recreates the VAM data set. VV copies VAM data sets.

Defining Data Sets for Programs

A convenient feature of TSS is the way that it provides logical connection between programs and data sets, minimizing the number and complexity of steps that users must take. For example, a user should not be burdened with space allocation on ***volumes.*** It is sufficient that users know the names of the programs they run and the names of the data sets to be processed.

DDEF Command

The DDEF command associates a DDNAME with a data set name by creating a ***control block,*** called a job file control block (JFCB). The JFCB can be accessed by user programs to obtain information about the data set. The system uses the JFCB in conjunction with another control block, called the ***data control block*** (DCB), to perform I/O operations on

the data set. The DCB is located in the user program, or a subroutine that the user program calls, and contains the DDNAME. The JFCB exists for the duration of the task or until a RELEASE command specifying the DDNAME is issued. Languages, such as FORTRAN and PL/I, treat I/O at a higher level than that just described. In other words, the programmer does not need to get involved with the system control blocks. The programs produced by these compilers are used with I/O subroutine libraries which interface with TSS.

Besides establishing a logical connection between programs and data, the DDEF command can be used to define the requirements for system resources needed by a data set. The DDEF command can also define a VAM partitioned data set as a job library, from which programs can be loaded by the ***dynamic loader.*** The JFCBs associated with data sets defined with the JOBLIB option are chained together to form the ***program library list.*** This list is also known as the JOBLIB chain. In the case of basic sequential access method (BSAM) data sets, the concatenate option of the DDEF command makes it possible to read several different data sets as one data set. The DDEF command can be used to supply information, via the JFCB, which is placed in the DCB by the system when the data set is opened.

TSS analyzes the requirements for the data set at the time the DDEF command is issued and, for ***private volumes***, issues mount messages, if necessary, to the system operator. When a ***private device*** is allocated to a task, a device reservation and a volume reservation are made. The volume reservation can be released without losing the device reservation, which allows successive volumes to be mounted. If the required space cannot be allocated, or the specified volumes cannot be mounted, the user receives notification. Private devices not associated with volumes, such as graphic displays, are handled within the same framework.

Nonconversational tasks wait until the system is able to satisfy the requirement for private devices before processing begins. The SECURE command is used to communicate a list of the kinds of devices and the quantity of each that will be needed. For a full description of the DDEF command, see the *Command Systems User's Guide.*

RELEASE Command

The RELEASE command reverses the action of the DDEF command and disposes of the JFCB that was created, freeing the DDNAME for other use. RELEASE is also used to free data sets from concatenation and to close and remove data sets from the program library list. Any programs loaded from a job library that is released are unloaded by the RELEASE command. Release of a DDNAME associated with an open data set results in that data set being closed. The volume reservation for private volumes can be released without releasing the device reservation, or both device reservation and volume reservation may be released.

Program Library List Control

A program in TSS can consist of one or more related object modules. All executable programs in TSS are stored in object module form in program libraries that are in the form of partitioned data sets. A program can consist of several object modules which reside in different libraries. The linkage editor and the dynamic loader can retrieve all required object modules if the libraries containing them have been appropriately defined.

The program library list is created and initialized with entries for USERLIB and SYSLIB(0) during LOGON. Job libraries can be added by the DDEF command and removed by the RELEASE command. The library at the top of the list always receives all object modules resulting from language processing. The user library is at the top of the list unless special action is taken. When a job library is defined with a DDEF command which specifies the JOBLIB option, that job library is placed at the top of the list. The JOBLIBS command can be used to rearrange the program library list, as desired.

Although it is not necessary to link-edit programs in TSS, there are situations in which link-editing is desirable. It is possible to conserve space in virtual storage and on external storage by combining related ***control sections,*** which can also reduce the ***working set*** of executing programs. The program library list can be used in conjunction with the linkage editor to define:

- The library that is to receive the link-edited object module
- The sequence in which libraries are to be searched by automatic call if the linkage editor must search for object modules to complete the link-edited object module

For example, if no other library is specified, the output of the linkage editor is stored in the library currently at the top of the program library list. If another library is specified at the time the linkage editor executes, that library receives the link-edited object module. The library can be the user library, any of the current job libraries, or a special library defined by a DDEF command without the JOBLIB option.

During link-editing, the library or libraries containing the object modules to be included in the link-edited object module are either specified in INCLUDE statements in the link-editor source program, or in the program library list. The object modules whose libraries are identified by INCLUDE statements are placed in the output module at the time the INCLUDE statement is processed. Those object modules required in the output module, but whose libraries are not defined by INCLUDE statements, are obtained after all statements in the linkage editor source program have been processed. They are retrieved by automatic call, using the program library list for the search.

Developing, Testing, and Running Programs

The main thrust of the TSS design and implementation is to make the person/machine interface highly interactive. This can enhance programmer productivity and also make it easy to execute in production mode, programs which were developed interactively. The program control system (PCS), the command system, and the dynamic loader are the principal TSS user program development aids. They are available at any time for investigation of production programs; it is not necessary to reassemble or recompile programs to enter a test mode.

The program event recording (PER) feature of System/370 is available to TSS users. This feature provides for automatic detection of various events such as alteration of, or reference to, virtual storage or machine registers. The PER hardware feature efficiently monitors execution of

user programs for these events. It is possible to specify a PCS statement that is to be executed on occurrence of a specified event such as an unintended store into a variable. An appropriate statement would be the STOP command, which would cause execution to be suspended immediately after the improper store operation. By displaying the program instruction counter it is possible to determine the operation which caused the unwanted overwrite. Each TSS user can utilize the PER feature independently. The feature is activated only when the CPU is executing the code of a user for whom PER is active.

Writing Programs

Facilities are available in TSS which affect the preparation of source programs to be tested and run under control of TSS. These facilities reduce the amount of work related to testing and running that is needed during program preparation and thus improve programmer productivity. The programmer can concentrate on the application, because tools for testing and running are available. ***Late binding*** of objects and ***dynamic allocation*** of resources help to free the programmer from concerns other than the immediate needs of the application.

The program development aids available in TSS eliminate the need to include debugging statements in the source program. This simplifies program writing, because many functions, previously source-coded, are available for use after language processing. For example, coding I/O statements to display intermediate results for test purposes is not necessary. Debugging statements can be implanted by PCS in the executable code that is loaded into virtual storage and do not become a permanent part of the object module in external storage. When execution of the program has been verified, the program is immediately ready for use in production, free from the potential risk associated with removing debugging statements or the inefficiency of not removing them.

Native TSS language processors (and the OS ASM H Program Product) can optionally produce an internal symbol dictionary (ISD) in the object module. An ISD makes it possible to use the names of statements and variables that are defined by the source program in PCS statements. This is especially significant to a FORTRAN programmer, who prefers to work with names in the source program and not machine addresses. ***External names*** are always retained in the object module for use by PCS commands, unless specifically deleted. When PCS displays or dumps variables, the type and length is often known, particularly when the variable name is in an ISD.

Loading Programs

An important aspect of language processing in TSS is that the output of a language processor (the object module) is in a form which, as far as the programmer is concerned, is directly executable. This is also true of OS/VS language processors running in the TSS environment. This is made possible by service provided by the dynamic loader and, in the case of OS/VS object modules, the ***object data converter*** (ODC). The dynamic loader eliminates the need for specific programmer action (link-editing) after language processing and before program execution. This increases interactiveness, because it is possible to change an object module without affecting other modules to which it is logically linked. ODC can operate in a mode logically equivalent to the OS/VS linkage editor, if desired. ODC is automatically invoked after running an OS/VS language processor under control of TSS.

Two types of object module linkage are possible: implicit and explicit. Object module use of a V-type ***address constant*** which refers to an external name constitutes a request for implicit linkage. When a previously undefined external name is encountered during loading, the dynamic loader attempts to resolve the reference. The dynamic loader determines if the name is defined in a module already loaded. If not, the dynamic loader searches the active job libraries for the name and, if the name can be found, loads the module containing the definition of the name. Succeeding modules may, in turn, reference new names. Loading continues until all references that can be resolved are found. Explicit linkage can be used to avoid unnecessary loading. A program may contain references to many subprograms, only a few of which are needed for a particular execution. In this case, the name of each required object is passed to the dynamic loader during execution.

A counterpart to the dynamic loader is the unloader, which reverses the action of the dynamic loader. Both are usable by the command system as well as user programs.

Program Control System

PCS creates a hands-on environment, which fosters increased programmer productivity in connection with the task of testing and running programs. PCS commands are issued in the same environment as other TSS commands. As such, they can also be issued from within PROCDEF commands and user programs, conversationally or nonconversationally, as can any TSS command.

PCS commands can be combined into command statements. There are three types of statements: immediate, dynamic, and conditional. The commands in an immediate statement are executed at the time the statement is issued. Dynamic statements are stored until control passes through a specified location in a user program or until the PER event being monitored occurs. Immediate and dynamic statements can be conditional. A conditional statement includes at least one IF command which is used to determine if the remainder of the statement is to be executed.

PCS commands can be used to:

- Display and dump data areas and instructions within a program, specifying these items by the names used in the source program, or by indication of the displacement from a known location and a length, or by indication of an absolute address and a length.
- Modify data areas and instructions within a program, specifying items as described for display and dump.
- Indicate locations within a program at which execution is to be started or stopped, specifying locations as described for display and dump.
- Indicate locations within a program at which PCS commands are to be automatically executed.
- Specify events to be monitored by the PER hardware feature such that when the events occur, PCS commands are automatically executed.

- Establish logical (true/false) conditions that control the action of PCS statements.

- Load and unload programs and subroutines.

- Perform arithmetic computations, using specified variables and the contents of data areas in user programs.

PCS statements consist of directives, operators, variables, and constants. The PCS directives are AT, BRANCH, CALL, DISPLAY, DUMP, GO, IF, LOAD, QUALIFY, REMOVE, SET, STOP, TRAP, and UNLOAD. Each directive designates a PCS command. The action of each PCS command is summarized under "Program Execution Management Commands" in Appendix A. Arithmetic, logical, or relational operators are used to form expressions. Variables are designated by external names, internal symbols, command symbols, absolute storage locations, or machine register numbers. Constants are one of six types: integer, character, hexadecimal, floating point, address, binary.

When referencing subscripted arrays (as with FORTRAN), individual elements may be specified. Subscripts can be arithmetic expressions. Synonyms for PCS command names and operands may be used. %CSECT and %COM are two special symbols that may be used to refer to the unnamed assembler language control section and the FORTRAN blank common block, respectively. FORTRAN statement numbers are defined in ISDs and are usable in PCS statements. FORTRAN statements without numbers can be referenced by relation to the preceding numbered statement. A counter, which may be referred to with the special symbol %, is associated with each dynamic statement and is incremented by 1 for each execution of the statement. The value of the counter may be displayed or dumped and can be used in forming expressions. A contiguous group of variables is specified by concatenation of symbols defining the range of locations containing the variables.

Program Test

The tools provided in TSS for program test utilize the computing system to perform functions not realizable with conventional hands-on machine time. To obtain maximum benefit from these tools, all programs should be planned for interactive execution even though they may be intended to be run exclusively in batch mode. (As mentioned previously, TSS program execution in interactive mode is identical to that in batch mode.)

Programs in TSS remain loaded unless specifically unloaded. Therefore, if it is desired to rerun portions of the program or the entire program, it should be ***serially reusable.***

During development, some subroutines of a TSS program may be left incomplete. For example, coding to perform limit checking or to handle unlikely error conditions can be deferred until exact needs are better known. Such code can be temporarily simulated in TSS with dummy statements. PCS AT commands can be implanted in these statements, allowing manual simulation of the missing function. Similarly, subroutines outside of the program may not yet be written. Again, AT commands can be used to cause control to be returned to the command system so that the function of the subroutine can be manually performed.

A TSS user program executes in ***problem state,*** is normally interruptible, and is loaded in an address space that is independent from the address spaces of other users, except for any virtual storage ***segments*** shared by mutual agreement. It is possible to signal attention during execution of one program, execute another program, signal attention during its execution, and so on. When a program runs to completion, a GO command causes resumption of the previously interrupted program.

It is convenient to scatter dummy statements throughout a program at points where it would be logical to stop and examine intermediate results. The names or statement numbers become operands of AT commands. Each use of an AT command causes the instruction at the location specified to be overlaid with a supervisor call instruction (SVC). This is one of the reasons why code should not modify itself or treat instructions as data.

There is always the possibility that a program may go into a loop. If this happens, the loop condition could be recognized by failure to reach an AT statement within a reasonable period of time. An attention interrupt from the terminal will cause execution to be suspended. If a loop is suspected, the TRAP command can be used to trace the loop and display pertinent variables. If a STOP command is part of the TRAP statement, execution proceeds one instruction at a time.

A principle of PCS and the command system is that the user may signal attention at any time, determine the point in the program where execution was interrupted, and resume execution (with the GO command). Signalling attention does not affect, other than to suspend, execution of the program. In those cases where user programs handle attention interrupts, it is possible to use AT statements in the attention routine.

In the event of program interrupt, PCS prints the location of the interrupt, indicates, if possible, the name of the control section in which the interrupt occurred, and gives control to the command system. Batch jobs terminate unless the user has taken specific action to handle the termination. It is possible to examine machine registers, the instruction counter, and any data addresses involved, determine the cause of the interrupt, make corrections, and resume execution.

Dumps are seldom needed during development of programs using TSS. This saves paper and time. Nonetheless, a dump may be desired. Because TSS commands (including PCS commands) are easily imbedded in programs written in any language, a programmer may choose to handle abnormal conditions in production programs with PCS. Such DUMP commands lie dormant and are called into use only when needed.

Control of Execution

Use of PCS is not limited to testing and debugging programs. PCS can be the means by which a user controls execution. Programs in different address spaces can be monitored and controlled using PCS to display and alter fields in shared virtual storage. An example of the type of program that can be run effectively, using PCS to control execution, is a mathematical model of a real process. The user can interrupt the program periodically, inspect intermediate results, alter constants (perhaps to adjust the rate of convergence of a calculation), and cause execution to resume. It is also possible that all of the input/output can be handled by PCS.

Program/System Communication

The preceding information in this section presented how *users* communicate with the system. The following explains how *programs* communicate with the system. Programs use data; the system manages data. Programs call for service; the system provides support. Getting and freeing virtual storage, checkpointing portions of virtual storage, checking virtual storage class, setting up for interrupt handling, waiting for events, obtaining the time of day, setting an interval of time for an event to occur, communicating between tasks, providing connection between address spaces, loading other programs to be called, and obtaining measurements of system utilization related to the work being performed are examples of the calls for service that can be issued by TSS programs.

Several libraries containing macros and DSECTs are supplied with TSS. The macros provide programs with linkage to the entire spectrum of system function. There are macros for nonprivileged user programs, ***privileged*** virtual storage programs, ***supervisor*** programs, recovery management system programs, and the independent utilities. The DSECTs describe every system control block in detail. Appendix B is a summary of the TSS macros, organized by functional area. The macros belong to two categories:

- Data management macros provide the means by which programs obtain input/output services.
- Program management macros provide the means by which programs are loaded, get storage, link to one another, service interrupts, interact with the command system, communicate with SYSIN/SYSOUT (with the user), communicate with the operator and the system log, invoke commands (which may be user programs), and utilize system-oriented information.

Data Management for Programs

One part of program/system communication concerns the storage and retrieval of data. Data management provides for the identification, storage, retrieval, sharing, copying, and erasing of data sets. It controls transfer of data between virtual storage and ***secondary storage*** devices. Supplied with TSS are programs which:

- Read data
- Write data
- Overlap reading, writing, and processing operations
- Read and verify volume and ***data set labels***
- Write data set labels
- Position volumes to the proper record
- Detect error conditions and correct them when possible
- Provide exits for user-written error and label routines

TSS data management facilities:

- Permit the user to store and retrieve data using storage facilities managed by TSS
- Free the user from concern with specific hardware configurations
- Permit the user to defer many specifications, such as device type and blocking factor, until the program is in execution
- Permit interchange of programs and data among installations
- Allow users to concentrate their programming efforts on the application and not the specifics of device programming
- Provide standardized methods for handling input/output and related operations
- Provide flexibility for including support for new or improved devices
- Provide for effective error recovery and recording

Data Set Names

A fully qualified data set name is a series of one or more simple names, called qualifiers. Each represents a level of qualification. For example, the name RUNOFF.CF.FRONT consists of three qualifiers. Starting from the left, each qualifier may be considered a category within which the next qualifier is a unique subcategory. This structure for data set names facilitates cataloging data sets and granting or obtaining access to groups of data sets. Fully qualified names are used by the DDEF command to set up a logical connection between programs and data.

A partially qualified data set name identifies a group of data sets, and omits one or more of the rightmost components of a data set name. The group of data sets referred to includes all that have qualifiers identical to those present in the partially qualified name. Partially qualified names are used in several commands when it is convenient to refer to the specified data sets as a group; for example, erasing a group, deleting it from the catalog, or specifying that others may share it.

These rules govern the choice of data set names:

- Each qualifier consists of from one to eight alphameric characters; the first must be alphabetic.
- A period must be used to separate qualifiers.
- The maximum number of characters (including periods) in the data set name is 35, which allows a maximum of 18 qualifiers, assuming each is a single character.

The system prefixes each name with the 8-character USERID followed by a period. Use of the USERID qualifier is restricted to the system, which is a major factor in establishing data security. Because every data set name is necessarily qualified by a unique USERID, every data set in the system is unique.

Catalog

The catalog is used for filing data set descriptions within TSS. Once a data set is created and cataloged, it can be located by name alone. Data sets reside on direct-access storage or tape. The identification of these volumes is available in the system catalog.

The system catalog is organized into a hierarchy of indexes:

- The highest level index, called the master index, contains USERIDs, one for each user joined to the system. The master index is maintained by the system and updated by JOIN and QUIT commands given by the system administrators or the system manager.

- The collection of indexes subordinate to each USERID in the master index is called a ***user catalog.*** The first index in the user catalog corresponds to the USERID. Each of the remaining indexes corresponds to a level of qualification in the data set name.

When a data set is cataloged, the required indexes are established in the user catalog, in accordance with the fully qualified name of the data set. An index is established for each level of qualification. The master index points to the highest index level of the user's catalog. This index, and each index thereafter, points to the location of the next lower index. The lowest index level points to the data set control block, which points to the pages of the data set. In the case of tape volumes, the lowest index level of the user's catalog also gives the order or sequence number of that data set on the volume, relative to the beginning of the volume.

Generation Data Groups

A generation data group (GDG) is a collection of successive, historically related data sets such as similar payroll data sets that are created every week. Cataloging such data sets with unique data set names would cause inconveniences that can be avoided. The system can assign numbers to individual data sets in a chronological collection, thereby allowing the user to catalog the entire collection under a single name. A GDG is created by the CATALOG command. It is possible to distinguish among successive data sets in the collection without the need for assignment of a new name to each data set. Because each new data set is normally created from the preceding one, it is called a generation, and the number associated with it is called a generation number. The user can refer to a particular generation by specifying, with the common name of the group, either the relative generation number or the absolute generation name of the data set.

Relative Generation Numbers: At any time, the relative generation number of the most recently cataloged data set in any GDG is 0. The relative generation numbers of previously cataloged data sets in the GDG are negative integers that indicate relationships to the latest cataloged generation. New data sets for the GDG are created by using positive integers as relative generation numbers.

Example: If payroll data sets are organized in a generation data group named `PAYROLL`, the most recent generation would be referred to as `PAYROLL(0)`. The preceding generation would be `PAYROLL(-1)`, the one before that would be `PAYROLL(-2)`, etc. A new generation would

be defined as PAYROLL(+1). When this new generation is cataloged, it becomes PAYROLL(0), and the old PAYROLL(0) becomes PAYROLL(-1). Thus, adding a generation changes the relative generation numbers of all data sets in the GDG.

Relative generation numbers depend upon the position of the data sets in the GDG. If a name is removed from the GDG, the relative numbers of the data sets shift accordingly.

Absolute Generation Names: To each data set in a GDG the system assigns an absolute generation name of the form GxxxxVyy, where xxxx, is an unsigned four-digit decimal generation number, and yy, an unsigned two-digit decimal number, indicating the version of a particular generation. Appending the absolute generation name to the name of the GDG provides a unique name for the data set.

Example: If 0001 is the generation number initially specified for generation data group DIVISION.PAYROLL, the first generation is DIVISION.PAYROLL.G0001V00, and the next generation is DIVISION.PAYROLL.G0002V00.

The DDEF command is used to create a new generation in a generation data group. The system develops the generation and version numbers as follows: The generation number is obtained by adding the positive relative generation number specified in the DDEF command to the previous generation number; the version number is set to 0. Thus, if the present generation is G1384V03 and the incrementing factor specified is the relative generation number (+2), the new generation is G1386V00. The system does not automatically create nonzero version numbers; if replacement of an existing generation and change of version number is desired, the CATALOG command is used. The system changes the catalog entry for the named generation. Thus, if a new data set that replaces the generation named G1386V00 is to be cataloged, it may be named G1386V01, which replaces G1386V00 in the GDG when the system changes the catalog entry. The data set that is to be replaced will be erased automatically if it resides on public storage, but not if it resides on private storage.

Each data set in a GDG has a unique fully qualified name. This name consists of the group name, a period, and the low-order qualifier GxxxxVyy. This allows 26 characters (including periods) for high-level qualification of the group name.

Data Set Security and Sharing

TSS is designed to prevent a user from accessing data sets belonging to other users without specific permission. The catalog is the repository of access control information about each data set. If users permit others to access their data sets, such permission is recorded in the catalog.

Cataloged data sets may be shared or nonshared. This affects the way the system processes the data set. Processing a shared data set incurs more overhead because of the additional control structure imposed. A data set is nonshared unless the owner issues a PERMIT command which allows it to be shared.

Access to a shared data set is in one of the following modes:

- *Read-only:* The sharer may read the data set, but cannot change it in any way.
- *Read/write:* The sharer may read and write the data set, but not erase it.
- *Unlimited:* The sharer may read, write, erase, and create a data set or generation.

The owner uses the PERMIT command to designate which users can share a data set and to indicate the type of access. The PERMIT command also allows the data set owner to change any access authorization previously given. For each level of qualification in the data set name, there can be a different list of sharers. Each sharer can be allowed a different type of access.

An owner may permit all of his data sets to be shared. To do this the owner specifies *ALL in a PERMIT command. Sharers must pick a qualifier for referring to data sets in the owner's catalog. The qualifier chosen becomes a prefix to the owner's data set name (or index) as perceived by the sharer. To have shared access to all of an owner's data sets, the sharer specifies *ALL for the owner's name. Similarly, groups of data sets with names having common high-order qualifiers can be shared by specifying partially qualified names in the owner's catalog.

Each time a PERMIT command is issued, the owner's catalog is updated with information on who can share which data sets, and with what level of access. The PERMIT command can indicate that all users or individual users (by USERID) have access to groups of data sets. Access to groups of data sets is given by referring, in the PERMIT command, to a partially qualified name. If a sharer has unlimited access to the group, a new data set can be created by the sharer but it will be owned by the owner of the group.

To attempt to gain access to a data set a sharer issues a SHARE command. For example, a data set owned by USERID MOHR, named RUNOFF.CF.FRONT, is to be shared and called RUNOFF.START in the catalog of USERID REYNOLDS. REYNOLDS issues:

```
SHARE RUNOFF.START,MOHR,RUNOFF.CF.FRONT
```

The SHARE command creates a name in the sharer's catalog and links that name to the owner's data set. If the owner's catalog does not allow access, the link is incomplete and inoperative. The name is still put in the sharer's catalog, but a diagnostic message appears on SYSOUT. The system will prevent access until the owner issues a PERMIT command for the data set. In this case the owner could have issued:

```
PERMIT RUNOFF.CF.FRONT,REYNOLDS,RO
```

The names that sharers choose to use for the shared data set have no effect on the owner's use of the data set or any other sharer's name for the data set. A sharer's catalog entry (the name chosen by the sharer) is

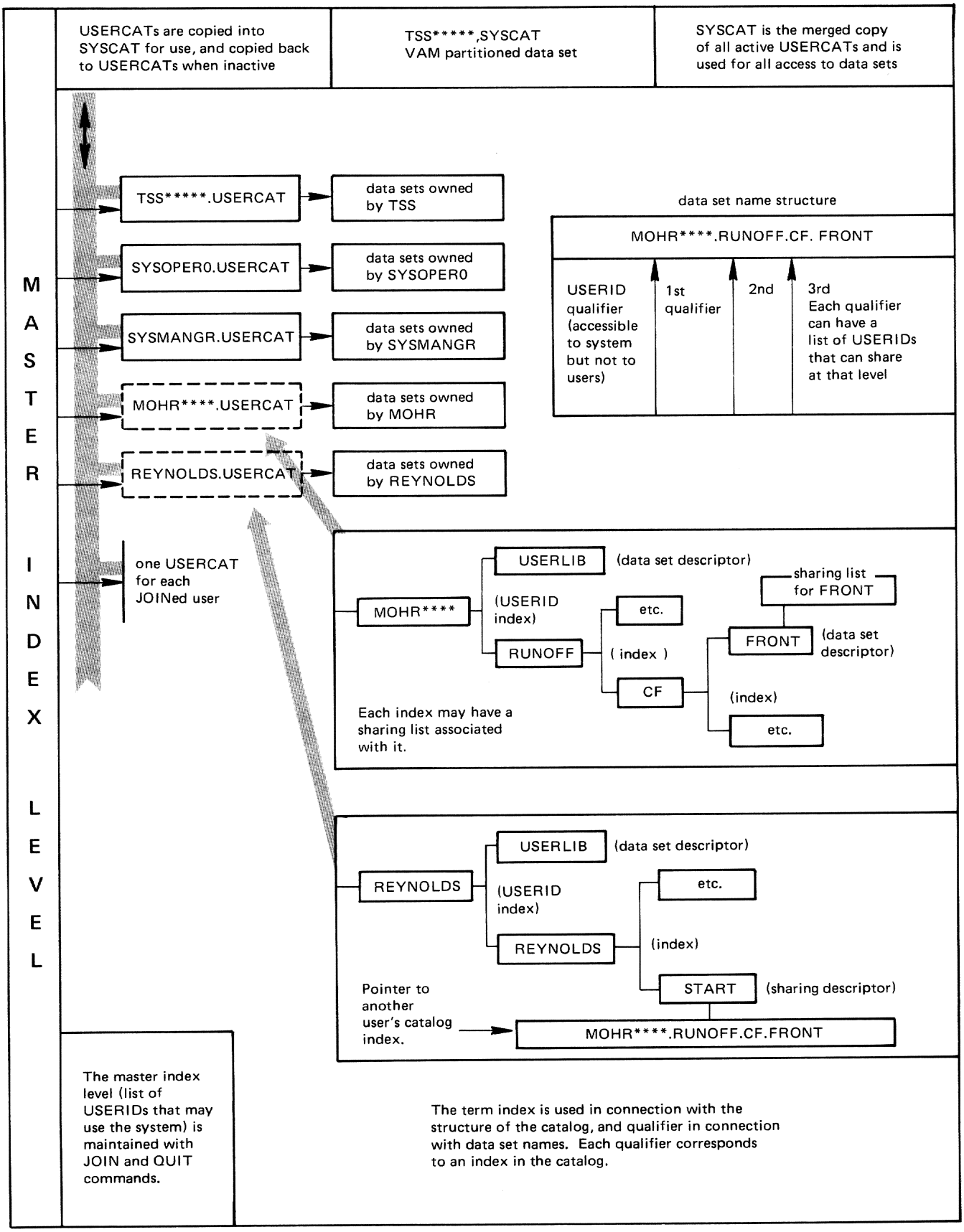

Figure 2-1. TSS Catalog Structure

not removed if the owner erases or uncatalogs the data set. Sharers can use the DELETE command to remove sharing names from their catalogs.

With VAM, it is possible to have safe, concurrent access to a data set by more than one task (user). To prevent several tasks from updating the same record at the same time, interlocks are maintained for a shared data set while it is being used. The two types of interlocks are: read and write. Depending on the organization of the data set, interlocks operate at a page level or a data set level.

A read interlock prevents other tasks from writing into the data set. Multiple read interlocks can be established, permitting several tasks to read simultaneously. Tasks attempting to set a read interlock are made to wait, if a write interlock is set.

A write interlock prevents any task, other than the task that set the interlock, from using the data set. Only one write interlock can be set on a data set; neither read nor write interlocks can be set until the write interlock is reset. The system suspends execution of a task waiting for an interlock until the interlock is released.

Public and Private Volumes

TSS data sets can be on either ***public volumes*** or private volumes. Generally, it is best to put data sets on public volumes. Public volumes are always mounted and available. When a private volume is requested, the system must determine if it can honor the request, based on current requirements for devices of a suitable type. If the system cannot assign a private device to a user's task, one of two actions occurs, depending upon the operational mode:

- A conversational task issues a message to the user during the execution of the DDEF command or macro if a suitable device cannot be assigned or if the required volumes cannot be mounted immediately. The user can wait, or cancel the request.

- A nonconversational task waits until the required number of private devices is available.

Permanent and Temporary Public Storage

In TSS, the user can specify whether a data set on public storage is permanent or temporary. A data set on permanent storage is retained until the user erases it. A data set on temporary storage is guaranteed to be retained only for the duration of the task by which it was created. Temporary data sets may be optionally erased when they are closed or when the task in which they are defined executes LOGOFF. By releasing the data definition prior to LOGOFF, the user may retain a temporary data set, but it is subject to erasure through the action of a system maintenance utility if it is not being used by any task.

Unit-Record Devices

Conventional unit-record I/O equipment (printers, card readers, and punches) cannot be referred to during program execution, unless an installation elects to make these devices available to user tasks. These devices are normally handled by a system task, called BULKIO.

BULKIO reads cards and catalogs the data sets which result, for the proper USERID. It also prints and punches data sets. Printing and

punching takes into account user requests for particular forms (types of paper and cards) and various setup conditions for the printer. Data sets which are printed and punched are accessed directly by the BULKIO task, rather than being spooled, thus eliminating the need to copy a data to process it. Such data sets need not be shared data sets, but if they are, access to them is possible while they are being printed or punched, subject to the read and write interlocks discussed above.

Data Set Organization

The data set organization defines the way that records of a data set can be accessed. In TSS, the most advantageous way to process data on direct-access storage is to use the ***virtual access method.*** The basic sequential access method (BSAM) is used for tape data sets. For various reasons, including need for interchange with OS/VS, BSAM can be used for data on direct-access storage. In situations where devices are to be supported in a way that requires special channel programming, the I/O request access method (IOREQ) is used. Unit-record and remote job entry (RJE) equipment is supported by a system task (BULKIO) which uses the multiple sequential access method (MSAM). MSAM can also be used by other tasks which have access to unit-record devices.

Virtual Access Method

VAM data sets are specifically organized for efficient processing within TSS. VAM is the primary access method in TSS. Data sets with VAM organization reside on direct-access storage, except when they are written on tape with the VT command, which provides for export-import. Users create, read, and process VAM data sets on the basis of logical records. The system organizes VAM data sets using page-size physical records. The page is the unit of transfer between direct-access storage and TSS virtual storage. The system also ensures that only the required pages of a data set are in virtual storage.

VAM data sets have these organizations:

- Virtual sequential
- Virtual indexed sequential
- Virtual partitioned

In a *virtual sequential access method* (VSAM) data set, the logical order of the records is determined solely by the order in which the records were created. The user presents records, one at a time, to the system. These logical records are organized by the system into page-size physical records, which are written to external storage. After each logical record is presented, the system provides a ***retrieval address*** for the record. The retrieval address may be saved for later use. After the data set has been created, the records can be read back in the order of their creation by requesting records one at a time. Using retrieval addresses, it is also possible to read and update records of the data set in a random order, thus obtaining direct access to any record. VSAM data sets can be interlocked on a data set basis only.

In a *virtual indexed sequential access method* (VISAM) data set, the logical records are organized in ascending collating sequence, based on a key contained in each record. The key may be a control field in the data, such as a part number, or it may be an arbitrary identifier in each record.

In addition to the logical records, VISAM data sets contain a page directory and locators that relate the keys to physical addresses of the records in the data set. In this context, the term *physical* refers to an address which is relative to the beginning of the data set without regard to the internal structure of the data set. The page directory is set up when the number of data pages in the data set exceeds one. There is one key entry in the directory for each data page in the data set, except for the first page (physical page 0). Each key entry contains the lowest key on the page plus the *logical* and *physical* (relative) location of the page in the data set. The directory can consist of one or more pages, depending on the size of the data set. In each page of the data set there is an ordered set of locators, one locator per record. Each locator specifies the physical location of the record in the page. Locators are placed sequentially (lowest key first) at the bottom of the data page, in ascending order. Records may or may not be logically sequential on a page, but locators are in sequential order.

New data pages are added to the physical end of the data set, even though they may logically represent insertions. By adding pages to the end and maintaining a translation mechanism (the directory), the need for overflow pages is eliminated. This greatly reduces the amount of reorganization required to maintain adequate performance in spite of numerous insertions. The user can optionally specify that a certain percentage of space be left in each page for expansion of the data set.

Because records in VISAM data sets have logical and physical relationships, the user can request the access method to:

- Retrieve or create records whose keys are in ascending collating sequence.

- Retrieve or create records whose keys are in any order. Processing is slower than if it were being done in collating sequence, because a search is required to locate the position of each record.

- Add records with new keys. The access method locates the proper position for the new record making possible subsequent retrieval in a sequence determined by the keys.

- Delete records. The access method updates the page locators (and the page directory if necessary) and frees the space that was being used by the deleted records.

- Update records in the data set (record length can be changed).

VISAM data sets are interlocked either on a data set basis or a page basis, depending on how the data set is opened.

A *virtual partitioned access method* (VPAM) data set is used to combine data objects into a single data set. Each object becomes a member, and each member is identified by a unique name. The member name may consist of from one to eight alphameric characters; the first character must be alphabetic. The partitioned organization allows the user to refer to either the entire data set or to any member, using a name consisting of the fully qualified data set name suffixed by the member name in parentheses.

Example: A VPAM data set named MATHLIB, whose members consist of mathematical subroutines, such as SQRT, ATAN, and COS, could be referred to in any of these ways:

Name	Description
MATHLIB	library of subroutines
MATHLIB(SQRT)	square-root subroutine
MATHLIB(ATAN)	arc-tangent subroutine
MATHLIB(COS)	cosine subroutine

Reference to individual members is possible, because there is a directory which keeps track of each member. As members are added, deleted, or changed, the directory information (member location, size, attributes, etc.) is updated by the access method. Space made available through deletion or contraction of members is immediately available for reuse. Also, members may expand, and if more space is needed outside the data set, it is automatically acquired.

A VPAM data set can be composed of VSAM or VISAM members or a mixture of both. Users can assign additional names, called aliases, to each member, and subsequently find a member on the basis of either the member name or any of its aliases. The partitioned data set organization is ideally suited for storage of libraries of programs or other groups of data that are frequently referred to together. Interlocking of VPAM data sets is done on a member basis. The rules of interlocking depend on the organization of the member (VISAM or VSAM).

Basic Sequential Access Method

BSAM data sets are typically used for interchange with operating systems other than TSS, but can be processed by TSS. The logical records in these data sets are organized solely on the basis of their position relative to the beginning of the data set. When these records are processed, a block of one or more logical records is the unit of transfer to and from the device involved. BSAM data sets can reside on disk or tape.

BSAM data sets can be concatenated, that is, automatically processed successively as a single data set. Concatenation of extensions to a data set already defined is accomplished with the DDEF command. The system provides for exit to a user program during the transition from one data set to another in the case where the characteristics of the data set differ. Concatenation applies only to data sets opened for input. Up to 255 data sets may be concatenated. The system performs volume switching without need for user program intervention. User label exit routines are executed for each data set, as requested.

Queued Sequential Access Method

QSAM makes it possible to access records in blocked or unblocked BSAM data sets without the need to write blocking and deblocking routines. Also when QSAM is used, I/O operations are automatically buffered by the system. Each time a program issues GET or PUT, a logical record is read or written.

Statements used to invoke QSAM are coded at a higher level compared to BSAM. A program using QSAM is more likely to be independent of the type of device on which the data it accesses is stored. QSAM can be used with BSAM data sets on disk or tape.

IOREQ Access Method

IOREQ is an access method in which user-written channel programs are executed by the system. The programs consist of virtual channel command words (VCCWs) for an I/O device. The access method and the system translate virtual addresses known to the user to real addresses needed by the channels. The system manages channel scheduling, inboard errors, error recording (optional), presentation of channel and device ending status, and sense operations in the event of unit check or unit exception. User interrupt routines may be used to handle asynchronous interrupts from the I/O devices.

Allowing users to write channel programs which refer to virtual storage has important consequences related to maintaining data security. To provide for security, the system checks the validity of all data areas referred to by the channel programs. Also, allocation of devices for use by programs that use IOREQ can be strictly controlled.

Multiple Sequential Access Method

MSAM is an extension of BSAM, applied to devices such as card readers, card punches, line printers, and RJE stations. In order to use these devices without placing unnecessary load on the CPU, channel commands are chained together by the access method so that one I/O operation processes many records.

Record Formats

TSS data management requires format specification for the logical records in a data set. Four types of record format are used:

- *Format F* (fixed-length) is specified for data sets whose logical records all have the same length.
- *Format V* (variable-length) is specified for data sets containing logical records that vary in length and contain the length as part of the record, following system-defined conventions.
- *Format U* (undefined-format) is specified when the records are of varying length but do not contain system-defined length information.
- *Format D* (variable-length ASCII[1] tape) is specified for ASCII tape data sets containing variable-length logical records. Format D may be specified only as a DCB subparameter of the DDEF command. (Other pertinent DCB subparameters should be specified for ASCII tape data sets with record format D.)

Details on the specification of record format and other record-oriented information, such as physical attributes, for each type of data set organization are described in the *PL/I Programmer's Guide*, *FORTRAN Programmer's Guide*, and *Data Management Facilities*.

[1]American National Standard Code for Information Interchange, ANSI X3.4-1968.

Specifying Data Set Characteristics

In order for a data set to be processed, a system program or a user program needs a specification of the characteristics of the data set such as record length and record format. This information can be made available from a variety of sources.

For a new data set (one to be created), the information is obtained from the DCB used when the data set is first opened. The information may have been moved there from the control block (JFCB) created by the DDEF command. The program can specify data set characteristics using the DCB. Commands and programs can obtain information from the JFCB or directly from the user.

For an old data set (one that exists), the catalog or the data set label can be used to obtain the required information. After a data set has been opened, the DCB associated with it contains a merge of the available information.

The DCB contains various types of information:

- DDNAME operand of the DDEF command corresponding to the data set to be processed
- Data set organization
- Record format information (type, length, etc.)
- Device-dependent options
- Exit addresses

  SYNAD: synchronous error exit address, for transferring control to a user-supplied routine if an uncorrectable I/O error occurs

  EODAD: end of data set address, for transferring control to an end-of-data routine when end of an input data set is detected during processing

  EXLST: (BSAM) exit list address, for transferring control to a user-supplied routine for creating or verifying user data set labels on tape and direct-access volumes, or for modifying the DCB during OPEN processing

- Working storage used by the access method routines

Besides being a command, DDEF is also a macro. Using the DDEF macro, a program may define a data set and specify whatever characteristics are appropriate. *A program can define data sets during execution,* without any preplanning by the user. This is a feature of TSS which significantly reduces the amount of programming needed to develop applications.

An assembler language program can add to or modify the contents of a DCB. Restrictions on modification of the DCB are stated in the publication *Assembler User Macro Instructions.* The DCB macro can be used to

generate a DCB, filled with specified information, in the assembled code. Once a field in the DCB is filled in this way, it will not be overlaid at the time the data set is opened.

During OPEN processing, information from the JFCB is used to complete the DCB. Any field that is empty during OPEN processing, and for which the JFCB is a valid source, can be filled from information supplied by DDEF.

Modification of particular DCB information saves restatement of all required information each time a program is run. To facilitate DCB modification, only those fields needed for program execution should be assembled into the DCB. Other fields should be left empty for subsequent fill-in. During execution of a program, fields will be filled in. The system keeps track of the initial condition of the DCB. Once the data set is closed, the DCB is restored to its pre-OPEN state. When the data set is opened again, the system repeats the fill-in process.

Identification of TSS FORTRAN Data Sets

Data sets to be processed by programs written in FORTRAN are identified by a data set reference number that appears in an I/O statement. Data set reference numbers are in the range 0 to 99. FORTRAN I/O uses the data set reference number from an I/O statement to construct a DDNAME of the form FTxxFyyy, where xx is the data set reference number, and yyy is a file sequence number used to differentiate data sets on the same volume.

The task definition table (TDT, a list of current JFCBs) is searched for a matching DDNAME and if one is found, FORTRAN I/O builds a DCB, puts the DDNAME in it, and opens the data set. After the DCB is opened, FORTRAN I/O examines it to see if enough information is present to process the data set. If not, default information is filled in to allow processing.

If an I/O statement does not contain a data set reference number, or if the data set reference number refers to a logical unit for which there is no JFCB, FORTRAN I/O assumes that SYSIN or SYSOUT is to be used.

Identification of TSS PL/I Data Sets

A data set to be processed by a program written in PL/I is identified by association with a file specified in the program. The association is between a DDNAME in the PL/I program and the JFCB created by a DDEF command. In the absence of definition of certain types of files, SYSIN or SYSOUT will be used.

The PL/I OPEN statement allows the user to specify characteristics of the data set. PL/I I/O creates a skeleton DCB for the data set using the file attributes explicitly declared in the program and merges them with those implied by the OPEN statement, completing the DCB as much as possible. Next, TSS data management is called to open the data set and provide additional information which can be acquired from the catalog and the JFCB. When the DCB fields have been filled by OPEN, PL/I I/O provides defaults for any fields still unfilled. If a conflict exists, the PL/I UNDEFINEDFILE condition is raised.

A file could be opened with the statement: OPEN FILE(MASTER), where MASTER is the file name defined in the DECLARE statement. The user may optionally specify a DDNAME by using the TITLE option in the file DECLARE statement. If the TITLE option is not used, MASTER is taken to be the DDNAME of a currently defined data set.

Identification of TSS Assembler Data Sets

Data sets processed by assembler language programs are identified by the DDNAME that appears in the DCB. The information assembled in the DCB, plus the information supplied from other sources, is merged as described above.

System Services for Programs

The other part of program/system communication is concerned with how programs obtain service from the system. The following description is written in terms familiar to assembler language programmers, but the information should also be useful to persons who program in higher-level languages, because it shows the services that can be obtained through the use of subroutines.

Services available to programs written in higher-level languages are described in the particular language definition. The compiled code calls a subroutine package appropriate to the operating system environment being used. When standard subroutine packages do not supply all services needed, additional subroutines, callable from user programs and usually written in assembler language, can be developed.

The OBEY facility of TSS makes it possible for programs to invoke any command. Thus, functions available at the command level also can be utilized by higher-level language programs. For each language, there is a subroutine that can be called which causes the string of characters passed as a parameter to be obeyed as a command. Because the TSS command system can invoke programs as commands, OBEY provides a connection between programs written in different languages for which a direct linkage is not defined.

A TSS user program operates in an address space which is unique to each user. The ***program status word*** (PSW) used for dispatching tasks in a virtual address space specifies problem state, which prevents execution of ***supervisor state*** instructions. The TSS supervisor supports two states of program execution in virtual storage, analogous to the supervisor and problem states of System/370. Each TSS task has associated with it a virtual PSW (VPSW) which specifies either ***nonprivileged state*** or ***privileged state.*** System programs can execute in privileged state and thus have access to data unavailable to user programs. Programs executing in privileged state can invoke functions which are not allowed to user programs. TSS is designed to prevent a user from causing instructions to be executed that would allow access to information belonging to any other user without the owner's permission.

Linkage from user programs to the system is accomplished either by branching directly or by issuing a supervisor call (SVC) instruction. Branching does not change the state (privileged/nonprivileged) of a task. In the case of the SVC, supervisory services are invoked and a change of

state is always involved. It is the responsibility of each supervisory service to validate any parameters associated with the SVC and carry out the requested action. The security of the system depends on supervisory programs checking all requests to prevent user programs from accessing privileged information or obtaining unauthorized services.

In order to relieve the programmer of details associated with calling for system services and in order to facilitate extension of services without need for alteration of source programs, calls to the system should be coded using macros. A macro is a group of statements written in a procedural language which, when expanded, generates source statements based on the parameters specified with the macro instruction. For example, to get a 100-page virtual storage work area, the statement

```
GETMAIN PAGE,LV=100
```

generates the coding needed to set up the proper parameter list and the instructions for a call to the supervisor.

Programs may utilize most TSS command processors by direct call, using macros. When a command is invoked through a macro entry point, it issues return codes instead of messages. The calling program can take appropriate action based on the return codes. Thus, advantageous aspects of ***interactive computing*** are also available to programs.

Emphasis in TSS on dynamic resource allocation, late binding, and a minimal need for preplanning is important, because thorough support of these concepts greatly reduces the amount of programming needed in the development of applications. Often, conversion of a subsystem from another operating system to TSS involves the removal of code, because function needed by the subsystem is present in TSS. This is particularly applicable to data sharing. TSS data management provides sharing control which relieves the need to program data security measures at the application level.

Many of the macros described in Appendix B have counterparts in other operating systems. The short descriptions in the appendix are intended to give a general idea of the facilities provided. In some cases, the facilities are unique to TSS or deserve special mention. The following topics are examples of cases where TSS differs from other operating systems:

Communicating with Users

Communication between a system program or a user program and a TSS user is through an interface which provides a standardized access to SYSIN and SYSOUT. For example, the command system reads commands from SYSIN and command processors use SYSOUT for output, completion messages, and error messages. The interface, common to conversational and nonconversational tasks, is called GATE and is a part of the TSS telecommunications access method (TAM II). In the case of an interactive user at a terminal, SYSIN is normally the keyboard and SYSOUT is normally the printer or display screen. In a nonconversational task, GATE uses VAM data sets for SYSIN/SYSOUT. A program, in connection with its use of GATE, need not be concerned with the nature of SYSIN and SYSOUT or whether the task is operating in a conversational or nonconversational mode. GATE processes input/output statements related to SYSIN/SYSOUT in a manner appropriate to terminal type and task mode.

Programs written in higher-level languages utilize GATE indirectly through linkage to a subroutine library. Language definitions typically include provision for input/output using a system facility in the absence of data set specification. Even when programs intend to read or write data to a data set, depending on the implementation of the language, it may be possible to direct the operations to SYSIN/SYSOUT. Assembler language programs use macros, such as GATRD, GATWR, and SYSIN, for input/output with SYSIN/SYSOUT.

Nonconversational jobs involve execution of a data set prepared in advance or entered as a batch input deck. Cards entered at the central site or a remote job entry station are cataloged in the catalog of the USERID specified in the job. This data set is erased automatically when the job completes. The SYSOUT data set is usually printed and erased but may be retained for further processing.

GATE, under control of the user, edits the data it processes. For example, there can be conversion of characters entered in lower case to upper case, use of characters, such as backspace, to correct previously entered data, use of tab characters on input to save entering blanks, and use of tab characters on output to improve the effective rate at which output can be transmitted to the terminal.

Each user can modify system handling of the terminal. For example, users can specify data and control characters according to individual needs, avoiding any translation or editing. Users can specify input and output translation tables which replace the standard translation tables. Also, function codes for character deletion, line deletion, carriage return suppression, line continuation, and the command system break character can be assigned to particular characters, as desired.

Terminals and transmission lines are prone to data errors. TAM II performs error recovery, keeping the user in control. This technique minimizes the amount of retyping that is needed. Users can vary details of correction to suit individual needs. TAM II also performs input and output buffering. Users can activate and deactivate input and output buffering, as desired.

Communicating with Terminals

TAM II can also be used by TSS user programs to communicate with terminals other than SYSIN and SYSOUT. The same GATE interface that applies to SYSIN and SYSOUT applies to terminals that are logically connected to user application programs. Thus, application programs can be as device-independent as system programs. The publication *Multiterminal Task Programming and Operation* explains how to use TAM II to communicate with terminals. The same macros used for communication with SYSIN/SYSOUT can be used in support of application programs. These macros are summarized under "SYSIN/SYSOUT Communication Macros" in Appendix B. Programs which control terminals other than SYSIN/SYSOUT need a way to handle interrupts in order to dispatch work. TAM II provides a means for dispatching user programs upon receipt of specific interrupts in addition to the ordinary means described below under the topic "User Interrupt Control."

Getting and Freeing Virtual Storage

Programs often need to get additional working storage during execution. The GETMAIN macro is used to add virtual storage to an address space. Virtual storage is freed using the FREEMAIN macro. In TSS, maximum virtual address space for each task is not provided until it is needed; as a result, the amount of ***real storage*** needed for the ***page tables*** is smaller and supervisor overhead is reduced. More importantly, this leaves address space free for use when connecting to the virtual storage of another user.

Users connect to shared virtual storage by loading control sections from shared job libraries with the dynamic loader. The control sections have the ***public attribute*** and can be read-only or read/write. The dynamic loader loads from job libraries, which are data sets. The access protection mechanism is the same as for any data set. Users need not learn another model of sharing.

User programs may test an address to determine if it is assigned, and determine the most restrictive protection class of a specified number of contiguous halfpages of virtual storage. This is done with the CKCLS (check class) macro.

Mapping Data to Virtual Storage Using VAM

A practical feature of TSS is that data on secondary storage can be ***mapped*** directly to the address space in which a user program operates. Less programming is required and often significant processing steps can be bypassed when the conventional model of I/O is not used. With conventional I/O, logical records are read from external storage, moved to a user area for processing, and then written back to the permanent storage. TSS VAM makes a quite different mode of processing possible.

For example, suppose an application involves access to a data structure 10 megabytes in size, and that the data structure is designed so that elements can be located by computation, in addition to direct sequential search. An implementation using VAM could be something like this: The data set containing the data structure has record format U (Undefined). The record length is 1 megabyte. The program gets a 10-megabyte work area using the GETMAIN macro and issues 10 GET macros which logically moves the data into the work area.

If the work area starts on a page boundary, which can be easily arranged, VAM will not physically move the data from external storage to the work area. Instead, the entries for the page tables related to the work area are set up to point to the data on external storage. The process of reading in 10 megabytes of data is almost instantaneous, because all that needs to be done is to update tables; no data need be read from external storage. As execution of the program proceeds, only pages of data referenced by the CPU will be brought in from external storage.

When the CPU references a page in the work area that has not been previously referenced, the hardware signals a translation exception. This interrupt, for a page in external storage, is no different than an interrupt

for a page that must be brought in from ***auxiliary storage.*** Access to a page in virtual storage may be more rapid than access to the page in external storage, depending on the frequency of use relative to other demands on main storage and auxiliary storage.

If the contents of a page in the work area are changed, the CPU sets the change bit in the ***storage key.*** When the application elects to write the data structure back to external storage, it positions access to the data set at the beginning and issues 10 PUT macros. This requests the supervisor to write the data. The supervisor can determine if a page in the work area has been changed and avoid writing any page which is unchanged. Thus, capturing changes to the 10-megabyte data structure can proceed quite rapidly, depending on the number of pages which have been changed. It is not necessary for the application to keep track of which pages have been changed; this is done by the system.

Large portions of virtual storage can be captured for later use with the CSTORE macro. CSTORE causes virtual storage to be written to external storage in object module format. The storage can be restored quickly and efficiently with the dynamic loader because, in TSS, loading is a process of mapping data on external storage to virtual storage. The areas of virtual storage which are copied into object modules by CSTORE need not be executable code. The advantage of saving a data structure in object module format is that it may be named in a way that is meaningful to programs which address the data by name through the dynamic loader.

Named, Disconnectable Segments of Virtual Storage

In the case of a System/370 computer, the amount of virtual storage that can be addressed at any one time is 16 megabytes. For those applications which require more than 16 megabytes to run, a facility is available in TSS which allows users to disconnect segments of virtual address space not being used. These disconnected segments are given names which can be used to reconnect the segments when access to the data is needed. Groups of segments are reserved (defined) with the RSVSEG macro. Disconnection is accomplished with the DISCSEG macro and connection with the CONSEG macro. RELSEG releases the segment reservation (definition) and DELSEG allows for deletion of disconnected segments which are no longer needed.

The example given above in "Mapping Data to Virtual Storage Using VAM" can be extended: If the data structures are somewhat independent, a number of them can be read in at once. As data from each structure is needed, the application issues the disconnect/reconnect sequence and each data structure becomes addressable as quickly as the pointers in the ***segment tables*** can be switched. It is possible to checkpoint changes made to the structures efficiently, because the process of disconnecting the segments does not affect the page tables which record the effect of changes. Thus, a task may have more than 16 megabytes of data in virtual storage, but only 16 megabytes may be addressed by the CPU at one time.

Loading and Linking Programs

The dynamic loader is used to load programs into virtual storage. It is also available for the purpose of loading data. This facilitates use of

data-directed linkages. Data base applications can potentially reference many programs and data elements, yet not be constrained by the need to load unused items. Programs are thus freed from many details of managing access to the data elements.

The LOAD macro causes a program to be mapped to (loaded into) virtual storage. The CALL macro is used to load and transfer control to a program. A call can be explicit (loading occurs only if the call is executed) or implicit (loading always occurs). The ARM macro, together with groups of address constants and flags generated by the ADCON macro and described by the ADCOND DSECT, can be used to achieve additional control of loading.

The DELETE macro is used to cause a module to be unloaded. When loading is achieved by explicit call, unloading causes the module to be totally disconnected from the rest of the loaded program. This allows substitution of subroutines without unloading the rest of the application.

Standard linkage is supported by the SAVE, RETURN, and BLIST macros.

User Interrupt Control

As stated before, each TSS user operates in unique virtual storage address space. Virtual storage is initialized with a set of programs that provide most of the services required by users. The programs in the address space are managed by an interrupt-driven supervisor, called the task monitor. The task monitor resides in the address space and is distinct from the supervisor, which is not in the address space. The task monitor receives interrupts as do other supervisors used with System/370: defined storage areas contain ***old PSWs*** and ***new PSWs.***

Interrupts are communicated to a task by the supervisor using a process analogous to hardware status switching. The interrupts processed by the task monitor are:

- Program
- SVC
- External
- Asynchronous I/O
- Timer
- Synchronous I/O
- Data set ***paging*** error

Interrupts of each type are identified by codes. In the absence of an interrupt processor for a code, a common diagnostic routine is called which sends an appropriate message to SYSOUT. If the task is conversational, corrective action can be taken. If the task is nonconversational, it is abnormally terminated.

TSS provides macros that permit the user to control task interrupts by specifying interrupt routines. The SIR macro is used to specify and

activate an interrupt routine. The DIR macro is used to delete an interrupt routine. User routines can service program interrupts, SVC interrupts (codes unused by TSS), external interrupts, asynchronous I/O interrupts, timer interrupts, and synchronous I/O interrupts. The SPEC, SSEC, SEEC, SAEC, STEC, and SIEC macros are used to initialize the control blocks needed for interrupt processing.

When a specified interrupt type occurs, execution of the running program is suspended, the task status (registers and VPSW) is saved, and control is passed to the user interrupt routine. The user program has access to a control block which includes information about the interrupt. The user program can also inspect and alter the saved status, including the VPSW. For example, SVC routines typically alter the instruction counter in the saved VPSW to branch around parameters. (To preserve security, the privileged indicator can not be altered to cause redispatch in privileged state.) Using the information in the control block, the interrupt routine can respond properly to the interrupt.

When the interrupt routine has concluded its processing, it returns control to the task monitor by using the standard system return linkage. The task monitor evaluates pending interrupt status according to priority. Ultimately, control is returned to the interrupted program at the point indicated by the VPSW saved at the time of interrupt. Using the INTINQ macro, interrupt routines can inquire if interrupts are pending for other user-specified interrupt routines. It is also possible to determine whether specified interrupts have occurred, before they are accepted.

An interrupt routine uses the SAI macro to indicate whether it may be interrupted. The task monitor saves and inhibits interrupts if the routine is not to be interrupted. Pending interrupts will be enqueued until they can be processed. If interrupts are not disabled, interrupts of higher priority will interrupt a lower priority routine. The RAE macro is used to restore the previous inhibit state and enable interrupts.

A System/370 instruction, Monitor Call (MC), facilitates analysis of program execution. The execution of an MC instruction depends on the contents of a machine control register. Sixteen classes of monitor calls are defined. In TSS, eight of these classes are reserved for nonprivileged user programs. When a class is enabled, and a monitor call for that class occurs, a program interrupt results. It is possible to specify a routine to handle this interrupt. MC, together with a user-written data reduction program, can be used to trace execution of a large subsystem.

A method for detecting and handling a subset of program interrupts which is more efficient than using interrupt routines dispatched by the task monitor is provided by use of the PIREC macro. PIREC is used to test an address to determine if a program interrupt will occur upon reference to the virtual storage pointed to by the address. When PIREC is used, normal processing for program interrupts resulting from invalid addresses is bypassed.

Servicing Attention Interrupts from SYSIN

Normally, user interrupt routines do not replace a system interrupt routine. However, to service attention interrupts from SYSIN, the user

must disable the system routine. This is done with the USATT macro. The SIR and SAEC macros are used to set up linkage to the user routine. The CLATT macro is used to enable the system routine. It is not necessary to delete the user interrupt routine (DIR) if control is to be relinquished only temporarily. Also, when exit is to the command system by such means as the CLIC macro or by reaching a PCS AT statement, the command system regains control of attention interrupts until execution of the user program resumes.

A second way to establish a user SYSIN attention servicing routine is provided by the AETD macro. AETD generates a table containing addresses of routines that are to be given control when the user signals attention a specified number of times. Typically, an attention routine identifies itself by writing a unique message to SYSOUT and waits for input. If attention is signalled again, the next routine gains control. A user may cause one of five different attention interrupt servicing routines to be given control, depending on the number of times attention is signalled. User routines invoked through use of AETD become a part of the system attention routine and remain a part of it until deactivated with the AETD macro.

AETD may be specified by a program that has been invoked to service an attention interrupt which occurred during execution of another program, (as defined by an AETD issued by that program) without causing the first AETD to be overridden. AETD macros can be issued at up to 10 such levels. However, if more than one AETD is issued at one level, only the last is recognized. When a program that has issued an AETD returns control, the entry specified by that program is deleted.

It is possible to leave blank the name for a certain attention routine number. In that case, when attention is signalled, the command system will be invoked to read from SYSIN. When the entry is not blank, the specified routine is given control. When it returns control, execution of the interrupted program resumes.

If the attention is signalled several times before the command system can process the first interrupt, and if no routine is active, all but the last attention is ignored. If a routine is active, each attention routine up to the number corresponding to the number of times attention was signalled is given control and, except for the last such routine, is immediately interrupted by the next queued interrupt. As each routine exits, the next lower routine is given control until it in turn exits; finally, the user program that was first interrupted resumes execution.

If SIR, SAEC, and USATT are used to establish a SYSIN attention routine, a user can specify different priorities for the servicing of attention interrupts in relation to other types of interrupts. If AETD is used, the user has no control over this type of priority specification.

If the user attention routine loops, and if the routine was established via SIR, SAEC, and USATT, there is no way to get out of the loop except to ask the system operator to force termination of the task. If the routine is established with AETD, a user can signal attention until control is given to the command system.

Timer Maintenance

In TSS, each task has eight interval timers unique to the task and accessible to user programs. (An additional eight timers per task are available to privileged programs.) The STIMER and TTIMER macro instructions set and test these timers.

The correct way for a program to determine calendar time in TSS is to use the REDTIM macro. REDTIM expands to an SVC allowing the supervisor to provide the time as a 64-bit fixed-point binary number which is the number of microseconds since the beginning of March 1, 1900. This date is the standard epoch for TSS.

The System/370 instruction Store Clock (STCK) is useful for timing brief intervals, but use of the REDTIM macro is recommended for time-of-day, because not all installations and operating systems set the time-of-day clock based on the same epoch. A program which uses a system service to obtain the time is less dependent on the operating environment.

The EBCDTIME macro can be used to convert time from the format of the REDTIM macro into various EBCDIC formats. System time can be translated into any combination of years, months, days, hours, minutes, seconds, and tenths and hundredths of seconds. When time is not specified with the EBCDTIME macro, local time is assumed. Use of EBCDTIME relieves programs from the need to take daylight savings time and leap year into account.

Interfacing User Programs with the Command System

TSS users can write programs which have the properties of system-supplied commands. The user commands which result, can be used in PROCDEF commands along with system commands, from which they are indistinguishable. The BPKDS macro is used to generate the necessary linkage information and parameter storage areas in the user program. The BUILTIN command is used to put the command name in the user's procedure library (a member of the user library) and to define a name that the command system can use to find the ***entry point*** defined in the expansion of the BPKDS macro in order to invoke the user program.

Services of the command system are available to user programs. The GDV macro allows user programs to obtain the value of a default from the combined dictionary of the task. The GETDV macro provides the value of any entry (default, synonym, command symbol definition) in the combined dictionary. The SETDV macro sets the value of any entry in the combined dictionary.

The OBEY macro can be used to cause a command to be executed from within a user program. Execution proceeds as if there were an attention interrupt followed by command entry from SYSIN. After command execution, user program execution resumes at the point of interruption, namely the first instruction after the OBEY macro. The command can be any valid command, including commands which cause execution of another user program. Programs written in higher-level languages utilize OBEY by calling system-supplied subroutines which issue the OBEY.

Using the PAUSE, COMMAND, CLIC, and CLIP macros, programs can yield control to the command system temporarily, causing the next record

to be read from SYSIN as a command. Resumption of execution is accomplished with the GO command. The EXIT macro terminates execution of a program, returning control to the command system. The ABEND macro can be used to indicate abnormal termination of a program and, depending on the severity specified, to cause the old task to be replaced with a new task.

Communicating with the Operator and the System Log

The TSS operator is very important to the success of an installation and does much more than watch a terminal for messages about putting paper in a printer or mounting disk and tape volumes. The operator monitors the entire operation of the system and is expected to detect trends and conditions that require attention. Users communicate with the operator, either verbally or with commands.

The operator's terminal is the same as any user's terminal in that the operator is logged on (USERID SYSOPER0). The operator USERID is accorded certain command privileges concomitant with operating the system. A user program communicates with SYSOPER0 by using the WTO, WTOA, and WTOR macros. Using these macros, the program can cause messages to be written to the SYSOPER0 terminal. The operator's reply to a WTOR is made available to the program.

The operator owns a generation data group named SYSLOG (system log). The system operator task writes the WTx messages, along with other log information, into the most current generation of the system log.

System-Oriented Macro Instructions

Some TSS macros perform system-related functions. These are most frequently used in support of subsystems. AWAIT is used to place a task in an inactive state. The task will be made active again upon receipt of an interrupt. Upon receipt of the interrupt, the task can take whatever action is appropriate. VSEND is used to send a message to another task. Receipt of a VSEND is associated with a task external interrupt. Applications can use AWAIT and VSEND to achieve multitasking where each task is a different TSS task. (It is also possible to perform multitasking within a single task, utilizing the services of the task monitor.)

USAGE makes available to a program the accumulated resource statistics for the current task as well as the total usage for the USERID since the last time the installation reset the totals. XTRTM extracts and examines the total accumulated CPU time.

HASH generates a *hash* value for a name according to the algorithm used throughout the system. LPCEDIT and LPCINIT can be used to make the services of the TSS editor available to user programs, for example, language processor controllers. LIBESRCH can be used to determine the location of a particular object module in the job library chain.

CHDERMAC and CHDVAL are useful when writing macros. CHDERMAC is a convenient way to generate error messages for conditions detected during macro expansion. CHDVAL is useful when determining the type code of a parameter during expansion. CHDPSECT is the means by which macro writers and coders of source programs in assem-

bler language cause generated code to be placed in the proper control sections as an aid in making TSS programs ***parallel reenterable.***

OS/VS Supervisor Services

Programs running under control of TSS which use parameter lists and calling conventions defined for OS/VS are supported by a subset of OS/VS services. The subset has been defined based upon the requirements of those OS/VS licensed programs which TSS supports. For a list of licensed programs supported for use in the TSS environment, see the *Command System User's Guide.*

Licensed programs are not altered to make them run under control of TSS. A utility program is used to obtain object modules from the volumes on which the programs are distributed. The object data converter converts the distributed OS/VS object modules into TSS object modules.

To provide a suitable interface for execution of the licensed programs, a certain degree of OS/VS simulation is implemented in TSS. OS/VS SVC functions, such as GETMAIN/FREEMAIN, are simulated at the functional level. The sequential, indexed sequential, direct, and partitioned access methods are logically simulated. Data records are maintained in TSS data sets and processed internally in a manner which simulates OS/VS data set characteristics.

TSS for the System Programmer

This section describes TSS for system programmers. The purpose of the section is to explain where various functions reside, identify the interfaces, and describe support facilities and features of TSS. The publication System Programmer's Guide *describes the interface presented to system programs. The publication* System Logic Summary *is intended for the reader interested in more detail.*

The intent of TSS design is to provide all the functions needed by ***subsystem*** developers, but sometimes, modification of the system is necessary. The structure, interfaces, and programming tools of TSS facilitate safe, effective system modification. To the extent that modifications can be made by addition, they are not dependent on internal characteristics of the TSS control program and are less likely to be adversely affected by maintenance and functional enhancement of the system.

Subsystems

Except for a discussion of the time sharing support system (TSSS), this section emphasizes structure, interfaces, modularity, management of change, methods of modification, and potential for extension. However convenient the facilities that are supplied, the factor of greatest concern to the system programmer is, "How can I make the system do what my installation needs?"

TSS provides a matrix in which subsystems may be easily embedded. Subsystem implementation is aided by a common, device-independent, terminal input/output handler, a smooth interface with the ***command system,*** sufficient ***virtual storage*** for generous workspaces, automatic ***external storage*** allocation, flexible and efficient ***sequential*** and ***indexed sequential access methods,*** and controlled program and data sharing. TSS supports subsystem programming at the same interface as end user programming.

The terminal handler and the interface to the command system allow a subsystem developer to take advantage of all facilities of TSS user profile control. These facilities make it possible for subsystem users to tailor the appearance of the subsystem according to individual needs. This capability need not be programmed in each subsystem.

The large ***address space*** available in TSS simplifies programming, because it is not often necessary to impact program design with overlay structures. The dynamic ***paging*** environment facilitates implementation of efficient subsystems. Unused workspace need never occupy ***real storage,*** yet is available instantaneously, without action by the program. Use of virtual storage ensures that no penalty (in terms of dedicated real storage) need be paid for the mere availability of a subsystem.

Automatic external storage allocation and flexible access methods allow data objects to be referenced by name, without concern for physical limitations of storage devices. This reduces the complexity of subsystem logic which would otherwise be needed to treat the conditions that arise as data objects expand and contract.

From a performance and usability standpoint, TSS facilities for dynamic sharing of programs and data encourage the subsystem developer to write reenterable programs. In this way, multiple users of a subsystem cause only a single copy of the program to be loaded. This sharing is managed by TSS automatically, relieving the subsystem of logic for control of sharing.

System Program Structure

The TSS operating environment consists of application programs and service routines operating under the control of supervisory programs. Supervisory programs are organized into components which function at different levels. The levels are distinct and the interfaces between the levels have been comparatively stable.

Distinct levels and stable interfaces facilitate management of change. Evolution in ***dynamic address translation*** architecture, great increases in the capacity of direct-access devices, and transition from terminals based on typewriter technology to cathode-ray tube displays were changes external to TSS which have been accommodated. Changes internal to TSS have resulted from many years of experience supporting TSS users as their capabilities and needs have grown. TSS has a clearly defined system program structure which facilitates management of change.

TSS is concerned with allocation of resources among competing needs. System designers and system programmers are involved with defining algorithms to apportion the finite resources available (storage, devices, control units, channels, and CPUs) among ***tasks*** (users) whose aggregate demand may greatly exceed the amount of resources available. TSS design enables users to work with a virtual computer, whose appearance is much simpler than that of a real computer.

Virtual Computers

The virtual computer created by TSS for a user is realized by a combination of hardware and software. A specific combination may be thought of as a *level* of virtual computer. Within TSS there are five levels of virtual computers. To each level, the levels below it appear to be hardware.

Level 4 (numbering from 0) is the environment in which users interact with TSS through the command system and ***language processors.*** Thus, to a user, programming solely in FORTRAN, the system may appear to be a FORTRAN computer.

Level 3 is the environment in which the language processors and ***user programs*** operate. This environment is characterized by program execution in ***problem state*** (defined by System/370 architecture) and ***nonprivileged state*** (defined by TSS architecture). TSS is designed to prevent level 3 programs from wresting control of the real computer or obtaining unauthorized access to data belonging to the system or users of other virtual computers. A level 3 program can address only that portion of virtual storage assigned to the user of the program or having a user ***storage key.*** Only problem state instructions can be executed, and the program may obtain service for only a safe subset of SVCs.

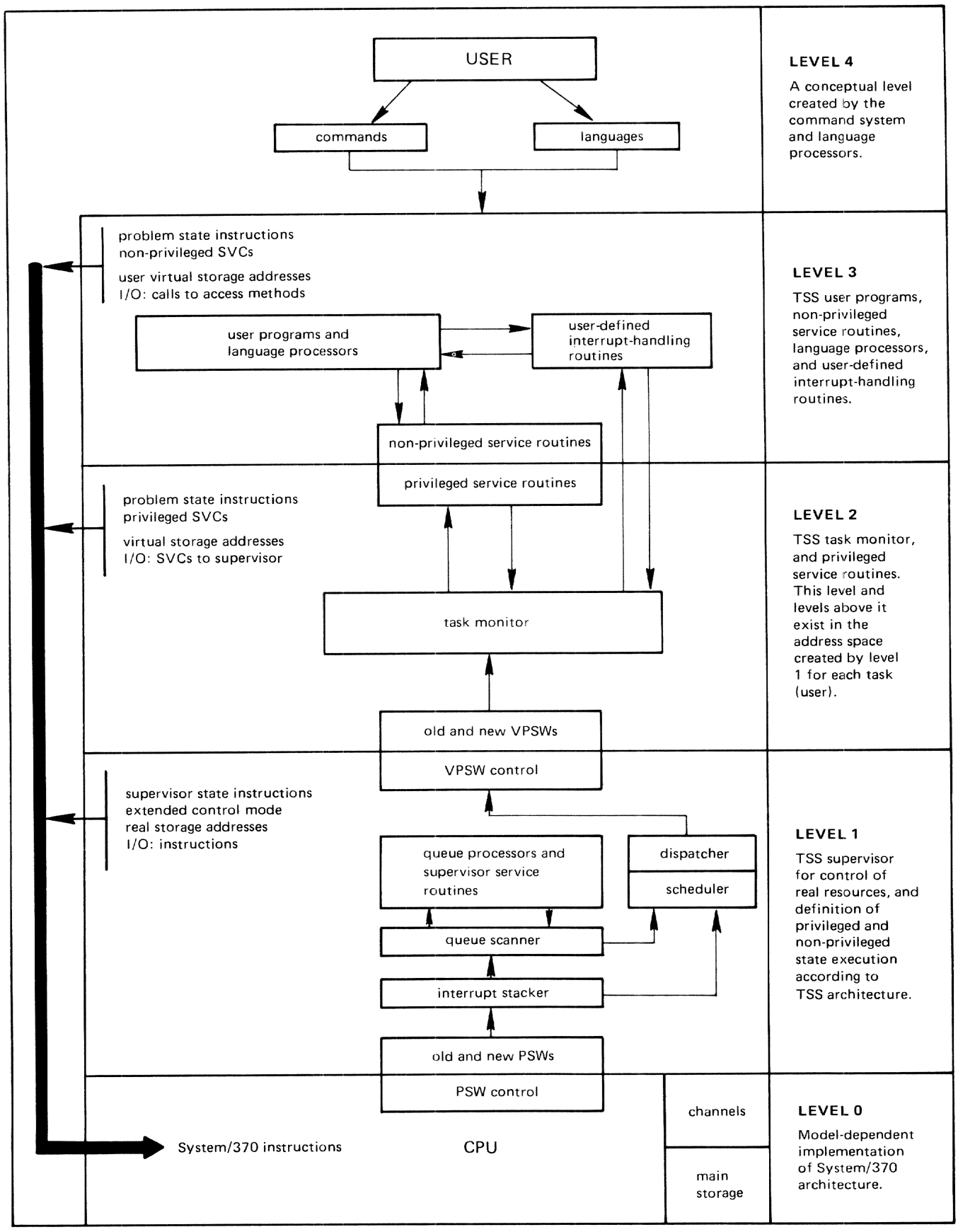

Figure 3-1. TSS Program Structure

Level 2 is characterized by program execution in hardware-defined problem state and software-defined ***privileged state.*** Only problem state instructions can be executed, but level 2 programs can obtain service for any SVC defined in TSS. A level 2 program can address the entire address space in which it executes, including level 3, but not that of lower levels.

Level 1 is characterized by program execution in ***supervisor state*** (defined by System/370 architecture). A level 1 program can address any part of real storage and any I/O device and may use any instruction. Programs and ***control blocks*** in level 1 are not addressable by programs in higher levels. This prevents a buildup of dependencies within application programs related to the specific level 1 implementation. The programs in level 1 are collectively referred to as the ***supervisor*** and are not subject to time-slicing.

Level 0 is the implementation of the hardware architecture. The address space or ***control storage*** of this level is not normally addressable by programs in higher levels. Level 0 controls the two states of the real computer, problem and supervisor, by the setting of a bit in the PSW as defined by System/370 architecture.

Levels of Protection

For purposes of discussion, the computer created by level *n* programs is defined as a level *n+1* computer. Level *n* programs run on level *n* computers. TSS architecture defines three execution states: supervisor, problem-privileged, and problem-nonprivileged. The terms *privileged state* and *nonprivileged state* are important concepts. These states are defined by TSS architecture and are the two possible states of level 2 computers. A level 2 program operates in the same address space as a level 3 program, but level 3 programs cannot access or damage level 2 programs or data. Programs in level 3 are dispatched with a key which does not match level 2 storage.

In the address space created by level 1 programs, fixed storage locations are reserved for PSWs associated with each type of interrupt which can be presented by level 1. Because these PSWs are in a virtual computer, they are called VPSWs. Level 1 controls the two states of the virtual computers it creates by the setting of a bit in the VPSW. The virtual computer created by level 1 operates in a manner that is analogous to the real computer created by level 0. When interruption of the level 2 computer occurs, the ***current VPSW*** is stored in the location reserved for the ***old VPSW*** and the VPSW in the location reserved for the ***new VPSW*** becomes the current VPSW. Level 2 computers always run in problem state.

System Protection

Basically, the system is protected from individual user tasks by the use of separate address spaces for the tasks. The supervisor executes with dynamic address translation turned off, whereas user tasks run with translation on. TSS tasks operate in an address space that contains most of what are normally considered supervisor functions such as access methods, task interrupt control, and device allocation. Therefore, the chance that a task malfunction can affect the system as a whole is reduced.

Task Protection

Within a task, there is a distinct separation between supervisory functions and user functions, enforced by the storage keys and ***protection keys.*** ***Nonprivileged*** programs (level 3 programs) have limited access to ***privileged*** (level 2) virtual storage. The user is given both read/write and read-only areas which provides protection from some kinds of damage caused by improper program execution. Improper storage reference causes program interrupts.

Data Protection

Each user is known to the system by a unique USERID. *Data is owned by individual users, not TSS.* This is a basic principle of protection in TSS.

Partitioning of Function

Virtual computers created by level 1, as far as problem state instructions are concerned, follow the hardware architecture, with the exception of the location and format of the old and new (V)PSWs. The functions of supervisor state instructions are replaced by a range of SVCs which provides supervisor services to level 2 programs. This includes support of VPSW loading, control of interrupts through masking, I/O, timer control, and authorization of connection to the address spaces of other level 2 computers.

The interface between level 1 and level 2 has great practical consequence. As a result, system programmers involved with modification of TSS should understand and preserve the interfaces between levels.

Implementation of the program product language interface (PPLI) is the creation, in level 3, of a virtual control program. When PPLI is activated, programs intended for OS/VS operate as if they were executing on a real control program. A significant portion of the user interface presented by OS/VS is emulated in TSS.

Level 1 programs relieve programs in level 2 and above from much detail. For example, the ***virtual access method*** (VAM), which is entirely in level 2, has a single logical window through which data passes, free of the details of device, control unit, and channel programming. For some other access methods, it is necessary for level 2 programs to specify virtual I/O channel programs to be executed, but this is of secondary importance. In this case, level 1 programs provide better support for I/O operations than a real computer. I/O scheduling is performed by level 1 programs, taking into account all tasks in the system.

Another example is the management of all address spaces by level 1 programs. Level 2 programs have no concern with algorithms for management of multiple address spaces or operation of paging devices. Resources of the real computer are managed by level 1 programs. Privacy of data in a level 2 computer is enforced by separation of the address space from all others.

The system program structure of TSS is intended to confine hardware and software errors to individual virtual computers. Requests for service that potentially impact many tasks can be validated by level 1 programs, significantly reducing the chance for global system failure. Problem determination, using TSS, is simplified because it is possible to operate a level 2 computer incrementally, while other level 2 computers (users) proceed at a normal rate. Also, in most cases, programs and data in a

specific level 2 computer can be replaced by test versions without affecting the entire system.

Address Space Map

In TSS, there are two separate storage maps of interest. The first represents storage occupied by the supervisor and its associated data areas. This is level 1 storage. The second represents level 2 storage. The supervisor initializes a separate level 2 computer for each TSS user. The term *level 2 computer* corresponds exactly to the term *task*. Each TSS task is initialized with a prototype copy of initial virtual storage.

The acronyms IVM (initial virtual memory) and VM (virtual memory) will be used in this publication to refer to level 2 storage. Appearing rarely in documentation and occasionally in code, RM (real memory) and RC (real core) are used to refer to level 1 storage. RM and VM also appear in the syntax of system programmer commands. Sometimes the supervisor is called the resident monitor and is also called the *resident supervisor* in older TSS publications. In discussions of where function resides, programs are "in real core," "part of IVM," or "loaded into VM," etc. In the discussion of address space maps which follows, all addresses are hexadecimal and quantities are decimal.

Initially, real storage contains nothing and there is no virtual storage anywhere. Programs and system tables needed to initialize the address spaces reside in ***data sets*** on direct-access storage. The data sets have specific names known to a program, called STARTUP, which is selected by the system operator after the computer load key is pressed.

STARTUP uses the system data sets to get the data needed for real storage (RM) and initial virtual storage (IVM). STARTUP loads the supervisor (RESSUP) into real storage. STARTUP loads IVM into a special virtual storage which is kept available as a prototype for each task. This prototype is effectively read-only and is used by all newly created tasks. When a ***page*** of nonshared IVM is changed by the activity of a task, it becomes unique to the task. Thus, the process of initializing the virtual storage of a task does not involve copying the entire IVM.

There are tables within RESSUP and IVM which must be initialized with a description of the system hardware configuration. STARTUP determines the number of CPUs, amount of real storage, and the operational paths to I/O devices, and updates the tables in RESSUP and IVM correspondingly.

After STARTUP has initialized the various address spaces, it creates the operator task. At this point, STARTUP has completed its function. Depending on the setting of an installation-selectable option, control is first given to TSSS for the purpose of setting parameters needed during system execution, or directly to TSS so that LOGON of the operator can be completed. The operator's task uses the terminal from which START-UP was selected, for SYSIN/SYSOUT and is like all other ***conversational*** tasks, except that it has operator command privilege.

Real Storage

The first page (4,096 bytes) of real storage is called the prefixed storage area (PSA). Each CPU in the system must have a separate first page in order to present interrupt status uniquely. Therefore, one page from somewhere in real storage must be reserved for each CPU. That page is addressable by all CPUs in the system and therefore must be

avoided, except for its use as a PSA. In System/370, the PSA location is defined for each CPU by the contents of a register which the operating software can load. This register serves as a prefix for CPU and channel storage references to locations 0-FFF.

The supervisor is loaded near the PSA. Due to provision for partitioning, real storage is not necessarily contiguous. Therefore, it is possible that RESSUP will not be adjacent to the PSA. Another reason for lack of contiguity could be the detection of defective storage locations. Before STARTUP is selected, an analysis of real storage is performed so that use of defective storage can be avoided. The size of RESSUP is approximately 300 kilobytes. This includes the nontransient portion of the resident support system (RSS), which is a part of TSSS. RSS normally is inactive, and much of it is contained in a special virtual storage built by STARTUP. When RSS is activated, real storage becomes part of this virtual storage. RSS pages the nonresident portion of its virtual storage independently from the supervisor.

The remainder of real storage is utilized according to need. The greatest use is for pages of virtual storage for TSS tasks. A small amount is used for control blocks and work areas needed by the supervisor. In order to allow for dynamic partitioning of real storage, the supervisor and control blocks that can be predicted to have a lifetime longer than a few seconds are placed in adjacent storage areas. Thus, when it is desired to remove a CPU and some storage from the system, contiguous real storage can be obtained relatively quickly. A map of RESSUP ***control sections*** is produced by STARTUP, providing addresses and version identification for all control sections and ***entry names.*** This map includes entries for the special RSS virtual storage.

The various types of real storage are protected with specific storage keys. Programs in level 2 and higher cannot address any level 1 storage because of the separation provided by the dynamic address translation hardware. Level 1 programs run in supervisor state with a PSW key that grants access to any part of real storage, but channel programs need not run with a universal access key. Therefore, the supervisor performs I/O using the key for the area involved with the operation. In this way, a class of supervisor I/O programming errors is detectable by the hardware. Each of the following has a different key: supervisor code, supervisor work areas, pages used for level 2 storage, pages used for level 3 storage, pages being used for paging operations, and pages held in reserve. This use of keys is possible in TSS, because storage keys are not required for multitask management.

Virtual Storage

The first two pages (8,192 bytes) of virtual storage are the interrupt storage area (ISA), functionally analogous to the PSA. The storage key for the first halfpage (address 0-7FF) allows nonprivileged (level 3) programs read/write access to the ISA. This facilitates communication between user programs and system programs without need for a base register. The remainder of the ISA is used by privileged (level 2) programs. It contains the new VPSWs and is store-protected from user programs. The ISA is part of IVM.

IVM is allocated downward (addresses decreasing), beginning with the highest address in VM. The highest page is not actually used. The ***page tables*** are set to cause a protection interrupt for any reference to this

page, providing a means for detection of a branch to an address in the range FFF000 to FFFFFF. Such a branch could occur, were a program to use the value that the ***dynamic loader*** stores in ***address constants*** that can not be resolved. The size of IVM is approximately three megabytes. STARTUP allocates space in ***private segments*** for control sections which are private to each task. Private IVM typically consists of control sections which change during execution and therefore can not be shared with other tasks. Read-only control sections can be public. Space for public control sections is allocated in ***public segments.*** This means that more than one task can use the same read-only code in main storage. STARTUP also produces a map of IVM control sections, providing address and version identification for all control sections and ***entry points.***

Beginning with address 2000, VM is unassigned and available for allocation in response to requests for storage. Requests for allocation are satisfied, working upward (addresses increasing) for nonprivileged storage and downward for privileged storage. The map of VM changes whenever programs are loaded and unloaded dynamically. The command system and dynamic loader provide users with the addresses of entry points.

Shared Virtual Storage

Sharing of virtual storage between address spaces (tasks) is accomplished in TSS on a ***segment*** basis. A virtual address consists of three parts: segment number, page number, and displacement. This sharing basis results from the two-level nature of the dynamic address translation hardware. For each segment of virtual storage there is a different page table. The page tables which make up a given address space are pointed to by a ***segment table.*** Sharing of virtual storage occurs when the segment tables for different address spaces have entries that point to one or more common page tables. User programs (level 3) achieve virtual storage sharing by use of the dynamic loader.

The address of each control section loaded by STARTUP is the same in all tasks. This applies to both private (nonshared) and public (shared) control sections. This can not be said for control sections loaded by the dynamic loader. This is because each task has a different history of loading. There can be no guarantee that the address of a specific ***shared segment,*** available for loading in one task, will be available in another task. Therefore, the address of a shared control section in one task can be different from the address of the same control section in another task. This difference is confined to the segment portion of the address. The number of the page within the segment is identical in all tasks which share the control section.

TSS users can share virtual storage without assistance from installation support people. All of the protection provided by data set ownership applies. The dynamic loader, which loads ***object modules,*** provides the connection between address spaces.

Object modules in TSS, including those converted from OS/VS, are structured to facilitate dynamic loading in a shared virtual storage environment. Object modules are made up of one or more control sections (CSECTs). CSECTs have entry points. Associated with each

entry point is a ***V-value*** which represents the address of the entry point. The V-value is used by any process that ***binds*** the output of language processors into an executable program.

Programs consist of fixed portions containing instructions and constants, and variable portions containing external addresses, work areas, and parameters. A CSECT consisting of fixed portions can have the attributes *read-only*, and *public*, so that it can be shared among different address spaces. (The TSS FORTRAN compiler can produce object modules with these attributes.) A ***public attribute*** causes the dynamic loader to use previously loaded copies of a CSECT to satisfy load requests, provided that the object modules are in the same shared ***job library.*** (A CSECT can be read/write and public, permitting an efficient exchange of information between tasks.)

In TSS, a special value is associated with entry points, and an additional attribute of CSECTs is defined, to facilitate use of dynamically loaded, read-only, shared CSECTs. In addition to a V-value, all entry points are assigned an ***R-value*** by the dynamic loader. A control section may have the *PSECT* attribute and in that case is called a PSECT (private CSECT). A PSECT can be used to store task-specific, variable information. (Programs written for operating systems which do not provide the combination of a dynamic loader and shared virtual storage are usable in TSS without regard to R-values and PSECTS.)

The purpose of R-values and PSECTs is to provide a convenient way to share programs among tasks executing in different virtual address spaces. For this kind of program sharing, control is passed to a location in a read-only, public CSECT. The program must establish addressability to its PSECT. The program can obtain addressability to itself, but there can be no address constant for the PSECT within the (read-only) CSECT, because the address of the PSECT may be different for each task. Therefore, the called program must depend on information passed to it. The information is supplied by the caller in the 19th word of a ***save area.*** The caller need not know the name of the PSECT, only the name of the program called. The dynamic loader stores the R-value of the called entry point in an R-constant located in the 19th word of the save area. Thus, a read-only, public (shared) CSECT can establish addressability to its PSECT.

TSS uses a save area which is one word longer than that used in OS/VS. This not a matter of great concern. Programs operating in the OS/VS environment created by PPLI are isolated from TSS calling conventions. Only when OS/VS programs are modified to call TSS programs that depend on the extra save area word, is it necessary to use a longer save area.

Where Function Resides

The system programmer needs to know where system programs and system data reside. Lists of names are scattered throughout the system and the documentation. There are lists of commands, modules, control sections, macros, DSECTs, and ***default values.*** The names of these objects usually can be found in some form of directory. For example, the names of all ***members*** of SYSLIB(0) may be obtained with the POD? ***(partitioned organization directory)*** command. Command names can be related to

object modules by inspection of ***regions*** in the SYSPRO member of SYSLIB(0). The publications *System Logic Summary*, *System Programmer's Guide*, and the program logic manuals associate name or function with program.

TSS is delivered with a complete set of source programs and macros, which corresponds to the object code. Maintenance distributions are accompanied by updates that correspond to the previous source programs. The new source programs and any changed macros or DSECTS are also supplied. It does not take long to get an accurate listing of any TSS program.

The function of each system data set supplied on the system residence ***volume*** (Volume ID: TSSRES) is as follows. Names are given as they appear on TSSRES. The first qualifier is the USERID. As with all data sets, this qualifier is not accessible to users.

The first group of names is presented in the order that the data sets are used when making TSS operational.

TSS*****.SYSIAM.DSTSSRES
(system independent access method): data set which contains the utility support system (USS). USS supports loading and execution of programs, independent from time sharing operation. USS is loaded by a program, called PRELUDE. PRELUDE is activated by hardware initial program load (IPL). The system operator selects STARTUP when it is desired to run TSS. As will be described later, STARTUP can build a quick-start data set on other direct-access volumes. If that is done, a copy of SYSIAM.DSTSSRES is written on the same volume as the quick-start data set and is called TSS*****.SYSIAM.DSxxxxxx, where xxxxxx is the volume ID of the receiving volume.

TSS*****.SYSUTL
(system utilities) is a ***generation data group*** (GDG) from which USS reads the most current data set to load the independent utilities.

TSS*****.STARTUP
(system STARTUP utility): GDG containing the program used to initialize the various address spaces needed for time sharing operation. The version of STARTUP to be used is loaded from the most current data set in the group.

TSS*****.SYSCCB
(system configuration control block): GDG from which STARTUP reads the most current data set to obtain the paths to printer(s), the number of CPUs, and the number of pages of main storage.

TSS*****.SYSIVM
(system initial virtual memory): GDG from which STARTUP reads the most current data set to obtain the modules for IVM.

TSS*****.RESSUP
(resident supervisor): GDG from which STARTUP reads the most current data set to obtain the modules for RESSUP.

TSS*****.RSSSUP
(resident support system, supplemental programs): GDG from which STARTUP reads the most current data set to obtain the modules for RSS. Modules in this data set are used in the special virtual storage paged under control of RSS.

TSS*****.SYSCAT
(system catalog): data set containing all ***user catalogs*** currently in use.

TSS*****.SYSSVCT
(system saved catalog table): data set used as a directory of user catalogs. This data set contains the status of each user catalog, indicating whether or not the USERCAT data set and the corresponding member of SYSCAT are synchronized and which is current.

TSS*****.USERCAT
(TSS user catalog): data set containing the catalog for USERID TSS.

TSS*****.SYSUSE
(system user table): data set which contains the USERIDs of all users authorized to use the system. SYSUSE contains attributes of the USERIDs, their allowable rations, and accumulated resource usage statistics.

SYSOPER0.USERCAT
(SYSOPER0 user catalog): data set containing the catalog for USERID SYSOPER0. The first task to log on when TSS is made operational is the main operator task, SYSOPER0.

TSS*****.SYSMAC
(system macro library): GDG containing definitions of all macros and DSECTs necessary to support normal nonprivileged user assemblies. The most current data set in this group is defined for each user automatically.

TSS*****.MACNDX
(macro library index): GDG in which each data set is an index to the corresponding SYSMAC. The most current data set in this group is defined for each user automatically.

TSS*****.SYSLIB
(system library): GDG containing the system library. The most current data set in this group is defined automatically by the system for each user as a JOBLIB. SYSLIB(0) contains object modules, the system procedure library, the system procedure dictionary, the system prototype profile, and other members needed for system operation. The dynamic loader can load privileged modules for nonprivileged users from SYSLIB(0), but only authorized USERIDs may store members in the SYSLIB. New generations of the SYSLIB can be created without restarting the system.

SYSOPER0.USERLIB
(SYSOPER0 user library): data set containing the user library for USERID SYSOPER0.

SYSOPER0.SYSLOG
(system log): GDG of data sets containing a log of communications with the operator task. The data set with the highest generation number contains the most recent log.

SYSOPER0.SYSBWQ
(system batch work queue): data set which contains the queue of work for the batch monitor and BULKIO task.

The second group of names represents data sets supplied on TSSRES but not needed to make the system operational.

TSS*****.SYSERP
(system error recording print) is a GDG containing programs used by the EREP command to format and print the error recording log. The most current data set contains the current versions. SYSERP also contains a member, ERPMAC, a macro library needed to assemble the programs in the library.

TSS*****.ASMMAC
(assembler macro library): GDG containing definitions of macros and DSECTs used to support assembly of system programs. The most current data set in this group, by convention, corresponds to the system object code.

TSS*****.ASMNDX
(assembler macro library index): GDG in which each data set is an index to the corresponding ASMMAC.

TSS*****.GENMAC
(generation macro library): GDG containing definitions of macros used to support system generation. The most current data set in this group, by convention, corresponds to the system object code.

TSS*****.GENNDX
(generation macro library index): GDG in which each data set is an index to the corresponding GENMAC.

TSS*****.UTLMAC
(utility macro library): GDG containing definitions of macros used to support assembly of STARTUP, PRELUDE, USS, and the independent utilities. The most current data set in this group, by convention, corresponds to the system object code.

TSS*****.UTLNDX
(utility macro library index): GDG in which each data set is an index to the corresponding UTLMAC.

TSS*****.USERLIB
(TSS user library): data set containing the user library for USERID TSS.

TSS*****.SOURCE.SYSGEN
(source for system generation): data set equivalent to the source used to generate the ***starter system*** supplied on TSSRES.

TSS*****.SYSGEN.MODULE
(system generation module): data set equivalent to the module used to generate the starter system supplied on TSSRES.

TSS*****.APGEN
(apply generation): data set which, when executed, applies the contents of SYSGEN.MODULE to the system.

SYSMANGR.USERCAT
(SYSMANGR user catalog): data set containing the catalog for USERID SYSMANGR.

SYSMANGR.USERLIB
(SYSMANGR user library): data set containing the user library for USERID SYSMANGR.

TSS*****.FORLIB
(FORTRAN library): data set containing object modules for the TSS FORTRAN compiler and run time library.

TSS*****.SCRIPT.FORTRAN
(FORTRAN linkage editor script): data set which is used to link-edit the modules in FORLIB and is supplied to facilitate maintenance.

TSS*****.VSSLIB
(VSS library): data set containing object modules for the virtual support system (VSS) component of TSSS.

TSS*****.SCRIPT.VSSLINK
(VSS linkage editor script): data set which is used to link-edit the modules in VSSLIB and is supplied to facilitate maintenance.

TSS*****.SCRIPT.PPLI.xxxxxxxx
(program product language interface script): data sets used to install the licensed programs supported by TSS, where xxxxxxxx identifies the specific program.

System Generation

There is not much that need be said about SYSGEN in the TSS environment. The process of generating an operational system does not involve selection and link-editing of programs. All programs in TSS are available all the time. Furthermore, because external storage (data set) allocation is automatic, no consideration need be given to prediction of data set size. There are, however, some parameters which affect system operation that can be specified in a SYSGEN. Most installations adjust these parameters by other means such as TSSS.

All that remains is to specify the number of CPUs, size of real storage, and the I/O configuration. Specifications needed to do a TSS SYSGEN are prepared as a source data set from which the assembler produces an object module. The ***linkage editor*** is used to copy the resultant tables into the appropriate system data sets. As an installation's configuration changes, subsequent SYSGENs can be prepared concurrent with production. It is possible to generate a system which will be run on a different configuration than is used for the generation process. The publication *System Generation and Maintenance* explains how to specify, create, maintain, and modify an installation-adapted TSS.

System Maintenance

An important principle which guides TSS maintenance activity is that *the system is created from source code*, not object code. When the system is delivered, the residence volume is in executable form, but maintenance is based on updates to source programs. This principle applies to the specification of TSS control blocks. Control blocks are described by DSECTs. Every bit in every byte in every control block of TSS is covered by specifications in the system DSECTs. A scan program can be used to document use of control block items. Information from the scan makes it easier to extend control blocks, because all affected programs can be identified. IBM-supplied maintenance packages include updated source programs and matching update control information.

Maintenance packages are called program temporary fixes (PTFs). In TSS, PTFs are not really temporary. A prospective PTF is verified to be consistent with system architecture. All PTFs that have been released have been permanently incorporated in the system. From time to time, PTFs are combined into releases. These releases are equivalent to systems updated with all PTFs.

The command system and linkage editor are used to modify the system according to ***scripts***, supplied with the PTF. Scripts execute as batch jobs and cover most maintenance. On rare occasions, manual action is required for maintenance. Installations which have local changes in an area affected by a PTF can modify the script accordingly. When the system maintenance job is complete, a system shutdown followed by a startup completes application of the PTF.

The majority of the activity of STARTUP is ***link-loading*** control sections from RESSUP, SYSIVM, and RSSSUP. STARTUP can be given a list of data sets, called ***delta data sets***, which contain object modules consisting of control sections intended to replace control sections of same name in the system data sets. Delta data sets are a means to dynamically modify the contents of IVM, RESSUP, and RSS. Changes can thus be tested without permanently updating the system. Contained in the system data sets, and also replaceable using delta data sets, are loadlists which specify control sections to be link-loaded. STARTUP uses the first occurrence of a control section name to satisfy the requirements of the loadlists.

The output of the link-load process can be saved. It is not necessary to link-load every time the system is started. One copy of STARTUP output can be saved on each direct-access volume, in a data set, called the quick-start data set. This allows the operator to load other copies.

Data Base/Data Communication

Many characteristics of the ***TSS application*** are typical of a ***data base/data communication*** (DB/DC) application. There are facilities for definition, creation, retrieval, update, restart, recovery, and reorganization of the data objects being managed. Support exists for recovery from damage to the data base due to read/write errors, physical damage, inadvertent erasing, and user error.

The TSS application (time sharing) is supported by facilities and system services which are also available to subsystem developers. That which is beneficial to the TSS application applies to subsystems. DB/DC facilities are an integral part of TSS and need not be added on via application

coding. In TSS, management of the space on direct-access storage is performed by the system, not support people, and all data set sharing is controlled with a common, protective locking structure that allows update by multiple processes/users. Also, when a terminal first connects to TSS, the user can select the TSS application or any of the subsystem applications. Communication with the terminal is supported by the same access method in both cases.

TSS Data Base

A TSS data base consists of all data sets in the catalogs of all users. A TSS user perceives data sets as in public storage or on ***private volumes.*** A private volume, by definition, is administered by individual users, although the actual assignment of physical media may be administered centrally. Installation management must administer ***public volumes*** and is ultimately responsible for the integrity of public storage.

The characteristics of the TSS system data base are:

- Every user has a catalog which contains the names of data sets owned by that user, information regarding who may share the data sets, and sharing names by which a user accesses other users' data sets. Two copies of the user catalog information exist when the catalog is being used. One becomes a member of the system catalog and the other is the backup copy (USERCAT), which is a separate data set. The backup copy is brought up to date when the user logs off or when the system is shut down. The system data set SYSSVCT indicates which catalog is valid and whether the catalogs are synchronized.

- Every VAM data set on public storage is cataloged at creation time.

- Every data set has an owner, and the owner's USERID is a part of the data set name.

- In addition to the catalog entry, every data set has a data set control block (DSCB). The DSCB contains the data set name, information describing the data set, and a map of the data set. The DSCB is pointed to by the catalog entry.

- Public storage volumes are identified by a relative volume number. This number, together with the relative page number of a page on the volume, forms a pointer which is used to specify the location of each page of the data set. The catalog entry contains a pointer of this form, with the addition of a slot number, which points to the initial DSCB for the data set. This enables the DSCB to be accessed directly, without a search.

- All public storage volumes (and private VAM volumes) are formatted in page-size blocks. Each VAM volume has a page availability table (PAT) which is pointed to by the ***volume label.*** The PAT identifies the use of each page on the volume: PAT page, data page, DSCB page, paging page, available page, etc.

These characteristics reveal that there is redundancy of control information. Furthermore, each data set has an owner. These facts are used in the implementation of ***service programs*** and the execution of strategies necessary to recover from catastrophic error. It is possible to:

- Rebuild a PAT from unique check information in the DSCB pages.
- Rebuild and verify a user catalog from information on public volumes or the system catalog.
- Check the consistency of the catalogs, the DSCBs, and the PATs and take corrective action where necessary.

In the environment TSS provides, users depend on the availability of their data. When an installation is faced with damage to its data base, these characteristics can be of much help in restoring operation as soon as possible and with minimum loss.

The characteristics of public storage have another important consequence. Installations can implement a facility to back up the data base, erase temporary data sets from public storage, and migrate unused data sets to ***offline storage.*** Because TSS keeps track of reference and change dates, backup can be limited to those data sets which have been changed since the last backup. Because all access to public storage data sets passes through the catalog, such a facility can intercept requests for data sets in offline (migrated) storage and automatically restore the data sets to public storage or invite the user to issue commands to do so. The necessary changes to privileged system programs to implement offline storage are typically additive; this kind of system extension does not lead to interdependencies with the TSS control program that are difficult to control.

TSS Data Communication

The TSS telecommunications access method (TAM II) provides a basis for exchange of data between the central system and remote terminals. TAM II is well-suited for this because of its modular design and convenient functional interfaces.

For the same reason that TSS ***data management*** isolates programs from storage device characteristics, the design of TAM II provides programs with device-independent access to communications lines and terminals.

For both the TSS application and subsystems supported by TSS, GATE creates a virtual terminal in the sense that programs can communicate without concern for actual terminal features. GATE handles differences such as output device line length, number of lines per frame, and insertion of idle characters to allow for carriage return. A program may specify functions such as page eject (appropriate to batch SYSOUTs). GATE converts these into appropriate action based on the terminal type. For example, a page eject control written to SYSOUT results in a skip to a new page on a batch output. On a display device, page eject causes the screen to be cleared in readiness for the next line of output. On a typewriter terminal, page eject results in spacing up three lines.

GATE also allows programs to write to and read from terminals with a minimum of "help." This is desirable, because some terminals emulate other terminals or are actually computers. GATE does not prevent transmission of characters which ordinarily have no meaning to the terminal but which may have meaning in special instances.

Because of its modular structure, convenient interfaces, and distribution of function among the TSS program levels, TAM II simplifies addition of support for communications devices. In fact, TAM II could be the basis of development of support for I/O devices not normally regarded as communications devices. In some situations, use of TAM II instead of IOREQ might be better because of its lower overhead. TAM II provides basic communications services between TSS tasks (the TSS application) and subsystems (user-written applications) and terminals.

The objectives of TAM II are to:

- Establish, control, and terminate access between tasks or subsystems and communications lines.
- Move data between tasks or subsystems and communications lines.
- Maintain a device-independent and data-independent interface between tasks or subsystems and communications lines.
- Establish and maintain a well-defined interface between device-dependent TAM II modules and other routines of the TSS control program.
- Allow tasks and subsystems to share communications controllers, lines, and terminals.
- Handle device-dependent and device-independent requests interchangeably.
- Provide input and output buffering, transparent to the user or system program, but under direct control of the terminal user.
- Provide reliability, availability, and serviceability aids to assist with maintenance and extension of device and functional support.
- Place the terminal environment under user control.
- Provide subsystems with support for a priority sequence of processing interrupts.

TAM II consists of four components, two in the supervisor (level 1) and two in IVM (level 2). The supervisor components are the real terminal access method (RTAM) and a set of device control modules (DCMs). The IVM components are the virtual terminal support system (VTSS) and a set of format control modules (FCMs).

RTAM controls all interaction between the DCMs and the supervisor. One interface is between the supervisor and RTAM and the other between RTAM and the DCMs. A DCM is a line controller. It is the responsibility of the DCM to get the data to and from the terminal. Ideally, a DCM need not contain any code related to the type of device on the communications line. A DCM is typically table-driven from a device control library (DCL).

The following are functions of a DCM:

- Do final validation of all I/O requests.
- Build the required channel programs and initiate their operation.
- Maintain line control during periods when no data is being transmitted.
- Perform initialization required when connecting a terminal to a task, whether the connection is initiated from the terminal or from the task.
- Set up device-dependent information in system control blocks.
- Handle all device-dependent interrupt status that is presented, except channel end/device end, and all *program controlled interrupt* (PCI) chaining requests.
- Provide error recovery for abnormal end conditions.
- Handle device-dependent timer routines.
- Provide simple output ***editing*** capability for supervisor messages to the terminal user.
- Determine length and type of input and check users' input for user and hardware function requests (cancel, attention, etc).

VTSS provides the interface between task or subsystem programs and TAM II. Implementation of the virtual terminal concept is the responsibility of VTSS and the associated FCMs. An FCM removes device control information from the data on input operations. On output, an FCM adds device control information to the data so that it is appropriately presented on the terminal. An FCM can be set up to handle a class of devices or access methods. For example, ***nonconversational*** tasks use VAM data sets for SYSIN/SYSOUT instead of terminals. For these tasks, VAM is used instead of RTAM, but user programs need not take this into account.

The following are functions of an FCM for output:

- Edit data according to a user function table.
- Do any block or ***record*** formatting required.
- Handle physical line length limits and required control character sequences.
- Translate EBCDIC data to line code.
- Invoke the correct I/O routine.
- Check return codes from RTAM and set up correct return codes for the program using TAM II.

The following are functions of an FCM for input:

- Translate line code to EBCDIC data.
- Remove any block and record format headers.
- Delete any device control characters.
- Edit input data according to a user function table and move it to the specified input area.
- Check return codes from RTAM and set up correct return codes for the program using TAM II.

The following are also the responsibility of an FCM:

- Perform control functions by either continuing the calling sequence or explicitly executing the request.
- Maintain correct sequence and buffer links, for buffered requests in virtual storage.
- Handle commands that relate to manipulation of the virtual terminal.
- Perform any special initialization required for connecting a device.
- Perform any special processing required for disconnecting a device.

Each TSS user can specify default values for settings of various parameters used by FCMs to control the operation of the terminal. These defaults can be saved using the PROFILE command. Separate characteristics can be stored for each terminal type. Profile information is stored in a member of the user library, called SYSFCL (from format control library). An FCL entry contains all the information and work areas needed by the TAM II to support the user terminal.

A terminal user initiates communication with TSS either by "dialing in" (originating a telephone call on the switched network) or signalling attention on a permanently connected terminal. The user enters LOGON and a USERID to select TSS, or, BEGIN and an application name to select an subsystem. A subsystem indicates to TSS the application name by which it will be known, by using the MTT command or macro. The other way for subsystems and terminals to be connected is for the subsystem to initiate connection by specifying the terminal address or telephone number.

System Support Facilities

TSS includes system programming tools that facilitate development and test of system programs. One is a support system, integrated with TSS, yet effectively independent of main system logic. Another is the means to collect data needed for performance analysis.

Time Sharing Support System

TSSS is a subsystem that operates within TSS and is very nearly independent of TSS. TSSS is essentially a development and maintenance tool that is operated from a terminal. Its purpose is to provide a system programmer with the capability to control the execution of system programs, gather data for analysis of system program errors, and apply corrections while TSS is running.

TSSS has access to all programs, tables, and control blocks of real, virtual, and external storage. It is essential to provide strict controls as to who can use TSSS, and when. The TSSS user can access and modify all data in the system. As delivered, use of TSSS is limited to persons with access to the main computer console, and users with system programmer authority. Individual installations can easily change the criteria.

TSSS has two components which operate in different modes. The resident support system (RSS) can be used from the main computer console only. RSS is invoked either by pressing the console interrupt key or by execution of an RSS SVC. When RSS is in control, normal TSS operation is quiesced. Time sharing operation resumes after an RSS RUN command has been issued or after the service requested by the SVC has been performed.

The virtual support system (VSS) can be used from any terminal supported by TSS. VSS executes separately in each task. VSS can be connected by use of the VSS command from the terminal or the RSS CONNECT command from the operator console. Once VSS is connected, any ***attention interrupt*** causes entry to the VSS command mode. When VSS is in control, only the operation of the related TSS task is quiesced. A null input in VSS command mode signals an attention interrupt to the TSS task and returns control as if a VSS RUN command had been issued.

TSSS uses a self-contained, interpretive language processor, with separate but basically identical versions in RSS and VSS, that reads commands and performs requested operations. Use of the TSSS command language is identical for RSS and VSS. The publication *Time Sharing Support System* describes TSSS and its capabilities. It also describes the TSSS command language, defining the functions of the language elements and the language syntax. The publication presents requirements for correct use of TSSS. The methodology of using TSSS is the same as that for PCS. The techniques applied to writing and testing programs in the TSSS environment, are similar to those used in the TSS environment, except that TSSS is independent from the TSS command system and cannot be controlled from within a PROCDEF or use the internal symbol dictionary (ISD) in a TSS object module.

TSSS and PCS are similar. PCS is intended for use with user programs. Its usefulness in connection with privileged system programs is limited to displaying and setting areas. It cannot be used to control execution of privileged programs. TSSS has greater functional capability than PCS, because its operation can be applied to programs in levels 1, 2, and 3.

As with PCS, an AT command can be used to cause TSSS to be invoked when control passes through a specified program instruction. There are very few instructions of TSS in which an AT statement may not be

planted. It is possible to plant an RSS AT in all instructions of the supervisor except for a few instructions in the interrupt stacker and the programs which comprise the system internal performance evaluator (SIPE). A VSS AT may not be placed in VSS or some areas of the task monitor, but an RSS AT can be placed anywhere in VM code.

TSSS commands can be grouped into command statements. There are three types of statements: immediate, dynamic, and conditional. The commands in an immediate statement are executed at the time the statement is issued. Dynamic statements are stored until control passes through a specified location in the user program. Immediate and dynamic statements can be conditional. (A conditional statement includes at least one IF command which is used to determine if the remainder of the statement is to be executed.)

TSSS commands can be used to:

- Cause TSSS to be invoked (under strict control) in support of the entire TSS or an individual TSS task.
- Define symbols and allocate any storage needed in support of TSSS operations.
- Display and dump data areas and instructions, specifying these items with reference to ***external names,*** by indication of the displacement from a known location and a length, or by indication of an absolute address.
- Collect information by moving data into an automatically managed collection area.
- Modify data areas and instructions, specifying items as for display and dump.
- Indicate locations at which execution is to be started or stopped, specifying locations as with display and dump.
- Indicate locations at which TSSS commands are to be automatically executed.
- Establish logical (true/false) conditions that control the action of TSSS statements.
- Perform arithmetic computations, using specified variables and the contents of data areas.

TSSS statements consist of directives, operators, variables, literals, system symbols, and constants. The TSSS directives are AT, CALL, COLLECT, CONNECT, DEFINE, DISCONNECT, DISPLAY, DUMP, END, IF, PATCH, QUALIFY, REMOVE, RUN, SET, and STOP. Each directive designates a TSSS command. The action of each TSSS command is summarized under "Time Sharing Support System Commands" in Appendix A. Arithmetic, logical, or relational operators are used to form expressions. Variables are designated by system symbols, external names, or absolute storage locations. A literal in the TSSS command language is an item of immediate data in the input stream. There are three kinds of literals acceptable to TSSS: decimal integer, hexadecimal, and character.

Only one type of data is classified as a constant in the TSSS command language: the address constant.

TSSS creates a hands-on environment for system programmers. Typically, TSS installations provide round-the-clock service, without interruption. If there were no TSSS and if hands-on time were the only way to test system programs, management would probably restrict change, seeking to increase availability and achieve stability. The presence of TSSS as a tool eases this conflict. The partitioning of function into different levels in TSS, and the fact that most programming takes place in a level where errors are not of global consequence, are additional factors that make the TSS system programmer's job easier.

System Internal Performance Evaluator

SIPE is a performance and system test tool intended for use by installation management. SIPE consists of two components. One is a program in the supervisor which collects and records data about system operation and the other is a set of data reduction programs. The supervisor portion is very nearly independent from TSS routines. During STARTUP execution, the operator is given a chance to bypass SIPE initialization. If initialization is bypassed or if SIPE is not active, SIPE places no load on the system. If recording is active, approximately five percent of CPU capacity is used for SIPE. SIPE records are written on unlabeled tape volumes in a special format that is recognized by the reduction programs. The amount of ***CPU time*** and tape channel activity needed for SIPE depends on the amount of information being collected and recorded.

The type of information collected by SIPE is controlled by the contents of a table. The table contains a switch for each type of SIPE event that can be recorded. Judiciously placed throughout the TSS control program are ***hooks,*** each of which is identified to SIPE by code number. A SIPE hook in real storage consists of two instructions. The first instruction executes an instruction in the PSA. The executed instruction in the PSA is either a no-operation (NOP) or the SIPE SVC, depending on whether SIPE is active. The second instruction is a NOP which specifies the hook code number. A SIPE hook in virtual storage consists of a SIPE SVC and a parameter list.

If SIPE is not active, two NOPs are executed when control passes through the hook. Thus, the cost of permanently including SIPE hooks in the supervisor is very small. The location of the hooks has been carefully selected to yield the most beneficial information. Individual installations are free to add hooks to the supervisor and virtual storage programs. When control passes through a SIPE hook and SIPE is active, the SIPE routine receives control and determines if the hook is armed. If so, SIPE buffers the information specified in the hook. When the buffer is full, a new buffer is used and the old one is written on tape.

The objective of SIPE is to collect data without seriously affecting system operation. Depending on the amount of information desired, there can be distortion of system operation due to the extra instructions needed to move the collected data into the SIPE buffers. This is usually not serious and the loss in system performance during SIPE recording is justified by the benefits that result from analysis of the data. It is not unusual for a TSS installation to run SIPE occasionally during normal production time.

Regular comparison of SIPE data can lead to identification of significant trends.

Examples of some of the items that can be recorded by SIPE are I/O initiation, interrupts, CPU dispatch, main storage allocation, ***auxiliary storage*** allocation, task creation, task dispatch, task status change, ***time slice*** end, task deletion, and page relocations. SIPE also periodically records information from system control blocks.

Each data reduction program is designed to fill a specific requirement. Examples are pictorial representations of main storage and drum storage as a function of time, and statistical summaries showing processing time, I/O activity, main storage usage. The data that is collected on tape is time-stamped and many events can be related to the activity of specific tasks. There is considerable potential for interesting research on resource control algorithms using the large amount of data obtainable with SIPE.

Dynamic Measurement Statistics

To provide the TSS application programmer with task-oriented resource usage measurements, statistics are continuously gathered for each task in the system. These statistics can be made available by the use of three pseudo-commands:

& (used in pairs to bracket commands) Display statistics for the bracketed commands.

% (used as a prefix to any command) Display statistics for prefixed command only.

@ Display statistics accumulated since LOGON.

The gathered statistics are also available to privileged programs on a system-wide basis. This allows continuous monitoring of system load.

Design Features

The TSS features which are described below are either unique to TSS, or can serve as the basis for extensions related to an installation's specific needs. The publication *System Logic Summary* describes many other design features of TSS which are not described here.

Supervisor

In the TSS context, the term *supervisor* is used to describe the programs which execute in supervisor state and are resident in main storage. Using the terminology explained under "System Program Structure" at the beginning of this section, these are level 1 programs. The supervisor is responsible for allocating real system resources, for performing services in response to requests from level 2 programs, and for responding to real hardware interrupts. The supervisor also maintains the status of control blocks which describe the resources allocated to each task. The supervisor consists of an interrupt stacker, a queue scanner, queue processors, a scheduler, a dispatcher, error handling and service subroutines, and system control blocks (tables).

Although the majority of function utilized by TSS users comes from programs outside the supervisor, the ability of TSS to provide many users with access to a computer and its data base in a safe and effective manner is due to the supervisor. Two aspects of the supervisor are

presented: management of the CPU resource and management of the storage resource.

The design for CPU management is strongly influenced by the need to effectively classify and process the various and frequent requests that occur in a multiprocessing, time sharing environment. To support sharing of real resources with many users, the supervisor classifies interrupts according to priority. When an interrupt occurs, the interrupt stacker receives control and enqueues a record of the interrupt on an appropriate queue. In order that a high-priority interrupt be accepted and enqueued quickly, most of the supervisor executes enabled for interrupts. Only very small portions execute with interrupts disabled. All entry points to the supervisor are in the interrupt stacker. Because the intervals during which TSS does not accept interrupts are very short, TSS is well-suited to the support of realtime applications.

When a CPU is interrupted while executing instructions for a user's task, it is necessary to save the status of the task (machine registers, PSW, etc.) for use when the task is dispatched again. If a CPU is executing supervisor instructions, the interrupt is enqueued and execution of the interrupted supervisor process resumes. Some interrupts caused by tasks require very little supervisor execution and can be handled immediately by the interrupt stacker, thereby eliminating the need to perform a full save of task status. Long, low-priority supervisor processes are typically handled in several short queues, which means that no allowance need be made for high-priority interrupts that must take over a CPU from low-priority processes. There are other interrupts, such as machine checks, that occur very rarely and require special processing and immediate handling.

The logical element built by the interrupt stacker and enqueued on an appropriate queue is called a general queue entry (GQE). A GQE either describes an interrupt or represents a request for supervisor work. The following are the supervisor queues, highest priority listed first:

- timer interrupt queue,
- drum request queues,
- drum interrupt queue,
- channel interrupt queue (all nondrum I/O interrupts),
- auxiliary storage allocation request queue,
- main storage allocation request queue,
- supervisor page-in request queue,
- device request queues, (all nondrum I/O requests),
- SVC interrupt queue,
- program interrupt queue,
- private page migration request queue,

- shared page migration request queue,
- real timer interrupt queue,
- vary request queue.

These queues are in a list of queues, called the scan table. Each I/O device is represented in the scan table in the order of the symbolic device address that was specified during system generation. In its capacity as the primary means of sequencing work, the queue scanner recognizes priorities for queue processor routines according to position of the queues in the scan table. Work which can be expected to have the greatest effect on total system performance is processed first.

Most supervisor work is performed by queue processors and their associated subroutines. Each queue processor is responsible for performing the work scheduled for it, as specified by the GQEs chained to its queue. A queue processor may be used by more than one queue. For example, all nondrum device queues use one processor. Typically, after the first GQE on a queue is processed, control is transferred to the queue scanner. This allows a higher priority queue to be serviced before any remaining GQEs are processed. In some cases, a queue processor searches its chain of GQEs and a GQE is selected for processing based on a specially defined priority. In other cases, all GQEs are removed from a queue, a chain of GQEs is maintained within the processor, and all the GQEs are processed.

When a GQE is created, it is given a list of the queues to which it will be sent. When one queue processor has completed its work for a GQE, the GQE is moved to the next queue until the list is null, at which point the work is complete and the GQE is deleted.

The queue scanner searches the queues for work, beginning with the highest priority queue. When work is found, it is given to the queue processor for that queue. When there is no more work, the queue scanner calls the internal scheduler. If the internal scheduler determines that work must be initiated to bring a task into execution, it will enqueue this work on the scan table and branch to the queue scanner. Finding no need to enqueue work, it branches to the dispatcher. The dispatcher, operating in conjunction with subroutines and a table that specifies scheduling parameters, gives control to a task (level 2 computer).

TSS was designed for a system with more than one CPU. More than one supervisor process can be executed simultaneously and supervisor processes can be in execution while tasks are in execution. Each CPU can execute instructions of the interrupt stacker, queue scanner, queue processors, dispatcher, supervisor subroutines, and tasks. The queue processors are ***parallel reenterable.*** To prevent more than one CPU from processing a GQE, the queue scanner marks a queue *busy* before control is passed to the queue processor for the queue. As soon as the GQE is removed from the queue, the queue is no longer busy and the queue can be processed by an available CPU. If a queue processor can not tolerate parallel execution or must prevent further processing of GQEs on the queue before exiting, it *suppresses* the queue. For example, when I/O is initiated on a device, the queue for that device is suppressed which prevents other GQEs representing requests for the same device from

being processed until the I/O for the first GQE is complete. Also, resource allocation queue processors suppress their queues when the resource they control is in short supply. When the resource is available in normal amounts, the queue is unsuppressed.

For some realtime applications, it may be necessary to add a queue to the scan table for a process which must be given a high priority. The design of the TSS supervisor makes it reasonable to add such function without seriously affecting supervisor interfaces.

The design for storage management is influenced by the needs characteristic of a multiprocessing, time sharing environment. Two types of storage are managed by the supervisor: main storage and auxiliary storage. Auxiliary storage is used for pages no longer needed in main storage and is divided into two classes: drum (high speed) and disk (low speed). Storage management algorithms for both types are interrelated and are discussed together.

The storage allocation parameters of a TSS task are specified by the schedule table entry for the task. Scheduling is described below under "Task Scheduling." Examples of the parameters are real storage limit, auxiliary storage limit, and flags to identify the algorithms to be used by the allocation routines.

When a task is regarded as a candidate for use of the CPU and storage its real storage limit is used to reduce the count of estimated available storage, even though the limit will not be reached immediately.

At time slice end, the estimated storage available counter is incremented by the task's real storage limit, even though the pages will not be purged immediately. This increase predicts that storage will become available, and therefore, operations to bring in pages for another task can begin, while the pages to be purged are being written. Prior to dispatching a task, some of the pages are read in a ***blocked page set,*** but most are brought in later on a demand basis. Because tasks do not reach their limit of pages immediately and because not all tasks reach their storage limit before time slice end, a dynamically calculated factor is added to the value taken from estimated storage available. The result is used to determine whether another task can be brought in. This keeps a slight pressure on the main storage resource to avoid underutilization.

When a task reaches its limit of storage and needs another page, one of two actions can be taken. Either its time slice will be ended and its pages removed from main storage and it will be rescheduled for another time slice (typically with a larger storage limit) or, if a flag is on in its schedule table entry, a page stealing algorithm will be invoked to steal pages from the task while it runs.

This algorithm uses a percentage value specified in the schedule table entry (typically around 80 percent), establishing a scan threshold. When the calculated number of pages is reached, all storage key reference bits for the task's pages are reset and the task is allowed to continue. When the limit is reached, all the reference bits are scanned and reset again. If enough pages were found unreferenced between the time of reaching the scan threshold and the time of reaching the maximum, these unreferenced pages are removed until the number of pages is below that determined by

the specified stealing percentage. The pages are removed from the task, the task is allowed to keep running, and the stealing process continues each time the limit is reached. If enough pages can not be found, the task is treated as if stealing were not allowed. The parameters governing the algorithm are contained in the schedule table entry for the task and, because the task transfers from one entry to another according to immediate conditions, can be chosen automatically according to the type of work the task is doing.

When a page is removed from a task, it is simply released, if unchanged, or if changed, written to a high-speed drum if one is available. If a page is unchanged and is determined to be a frequently used page and is resident on disk, it will be moved to a drum so that it may be read into storage more rapidly on subsequent use. Also, at time slice end, a set of frequently used pages is identified. This set forms the blocked page set referred to above, thus avoiding a large number of translation exceptions at the start of a time slice. The size of the blocked page set is typically eight pages. When a page is released to the list of available pages, information about its previous use is retained so that an I/O operation can be avoided if that page is needed again before the contents of the main storage page have been used for another purpose.

If satisfying a page allocation request would reduce the number of available pages below a threshold value, storage has been overcommitted and action will be taken to obtain more pages. Shared pages in main storage are scanned to find pages not recently used. If this does not yield sufficient main storage, all of the tasks occupying storage are checked to see if any are of a lower priority than the requesting task. If one is found, it will be forced to end its time slice, making its pages available. If none can be found, the page request will be discarded and the requesting task will be forced to end its time slice.

The following description of auxiliary storage management algorithms applies when one or more drums are used. If there are no drums in the configuration, the only thing of interest is the selection of the disks and locations on the disks for auxiliary storage pages. Typically, active tasks need more main storage and high-speed auxiliary storage than is available.

At time slice end, a task's changed pages are written to auxiliary storage. Frequently referenced pages are identified and become part of the blocked page set mentioned above. Requests to write pages may include an indication of preference for use of drum or disk. The blocked page set may be divided between drum and disk according to parameters specified at each installation. All pages blocked to drum along with the page table pages indicate drum preference. Pages blocked to disk indicate disk preference. Preference is not indicated for the other pages, but if enough drum space is available, they are written to drum.

When available drum space falls below a specified threshold, migration of pages from drum to disk begins. This migration can take place while the task is receiving service from a CPU. In order to choose tasks for migration, a scan is made of all tasks, to find those which exceed their fair share of drum pages. The fair share is a calculated value, but can be overridden by a value in the schedule table entry for the task (see below). Tasks that exceed a fair share of the drum are subject to having their pages migrated to disk until the number of their pages on drum is reduced

to the fair-share value. Frequently referenced pages are not moved from drum.

If the available drum space falls below another threshold, which is lower than the migration threshold, only writes that indicate drum preference actually go to drum. All unspecified requests go to disk until the migration has freed up drum space.

If a task is inactive for a long time, all its drum pages are moved to disk. This time may be specified by an installation and is usually in the range of a few minutes.

Task Scheduling

Scheduling of the CPU resource in TSS is controlled by a table-driven scheduler. This approach allows flexibility in controlling factors that are always subject to change. By making the scheduler table-driven, many classic algorithms may be simultaneously incorporated. Also, each installation may design scheduling algorithms without the need to reprogram. It is possible to modify the table during system execution. In TSS, the scheduling function is contained in just a few programs, which simplifies learning about the scheduler and modifying it. Task scheduling and selection is performed by an internal scheduler, a dispatcher, the entrance criteria subroutine, and the rescheduling subroutine.

The system maintains a list of active tasks and a list of inactive tasks. The ***active list*** is divided into the ***eligible list*** and the ***dispatchable list.*** Eligible tasks wait for entry into the dispatchable list. Tasks in the dispatchable list occupy real storage and are either in execution or waiting for the CPU. The ***inactive list*** contains those tasks waiting on long-delay stimuli, such as an interrupt from a terminal. The internal scheduler controls movement of tasks from the eligible list to the dispatchable list.

The entrance criteria subroutine is called by the internal scheduler to verify or deny a task's eligibility to be moved from the eligible list to the dispatchable list. The internal scheduler also determines the order of the dispatchable list. The dispatcher selects the task to be executed. The rescheduling subroutine is called when a task reaches time slice end (normal or forced) and controls movement of the task from the dispatchable list to the eligible list. It also moves a task from the eligible list to the inactive list or vice versa and is used to delete the task from all lists. This subroutine determines the reason for the time slice end. Based on the reason, it switches the task to the proper level.

The schedule table resides in main storage (level 1). The table consists of a number of entries, each referred to as a *level* or *schedule table entry.* Some terms are given below without explanation just to give an idea of the type of parameters possible. The abbreviation TSE is used for time slice end. For a fuller discussion of the schedule table see the *System Logic Summary.* Each level has six classes of fields:

- Control of the order in which tasks move from the eligible list to the dispatchable list (priority, delta-to-run, recompute scheduled start time)

- Limits of task demands on a per time slice basis, used to determine when TSE will be reached (CPU time per quantum, number of quanta,

real storage limit, maximum translation exception count, short-term task wait time, preempt flag)

- Transitional fields to indicate which level will be used next upon reaching each type of limit or event (Pulse SVC, TSE SVC, real storage limit, terminal wait, short-term wait, TSE while holding lock, low-core forced TSE holding lock, waiting on lock, terminal write-only operation, low-core forced TSE, next steal level)

- Control of the order of the dispatchable list (maximum translation exceptions per quanta)

- Control of the page stealing algorithm (steal request flag, steal threshold percentage)

- Override of a task's fair share of drum pages

The transitional fields are like branches in a program. When a user is joined to TSS, one of ten *external priorities* is specified. Each task that logs on is initially started at a level which is determined by the external priority. If the task is conversational, the number of the initial level is equal to the external priority. If nonconversational, it is the external priority plus ten. As the task executes, it moves from one level to another, within one set of levels. Some installations have installed a command which allows a user to change the level of the task in which the command is issued.

Task Monitor

The task monitor is to the level 2 computer as the supervisor is to a level 1 computer. Interrupts are presented to the level 2 computer by the supervisor in the same way that the hardware presents interrupts to the supervisor. Within the supervisor, the dispatcher selects a task which is to receive CPU service. Before actually dispatching the task, the task interrupt control subroutine is called to see if there are any pending interrupts to be processed by the task. If so, the current VPSW is stored in the old VPSW location corresponding to the interrupt, and the corresponding new VPSW becomes the current VPSW. This is analogous to the way level 0 programs handle the real hardware.

The basic function of a control program is to control real resources and to provide services to tasks. In TSS, control program functions are separated into a resident portion which controls the real resources and a nonresident portion which provides services to the tasks. The nonresident portion resides in the level 2 address space, which, being virtual storage, does not permanently occupy main storage. The task monitor and its associated service routines act as if they constitute a resident supervisor for each task. The task monitor is very much like the OS/MVT supervisor as regards its functional capability.

The analogy which compares the interface between the supervisor and the real hardware, on the one hand, and the interface between the task monitor and supervisor on the other, is quite extensive, but it is not exact. A difference is that the interface has been extended to facilitate, among other things, internal sharing of programs and data and intertask communication.

The task monitor and most system service routines operate in the privileged state. Programs belonging to the ordinary user execute in the nonprivileged state. This protects privileged programs from damage due to actions of nonprivileged programs. The separation is enforced with the storage keys. (Protection between users is enforced by use of different virtual storage address spaces.)

The dynamic loader can discriminate between authorized and unauthorized requests to load privileged modules based on identity of the data set from which the program is loaded and the authority of the user attempting the load.

Finally, protection is accomplished through validation of requests for supervisor service. In response to a task-related SVC, the supervisor creates an interruption of the task which is processed by the task monitor, which, in turn, invokes privileged system service routines. These routines determine if the request is valid. The reason for communicating between problem and privileged state via the supervisor is that only the supervisor can execute instructions which alter the PSW protection key field.

Many TSS users do not become involved consciously with the services of the task monitor. When a task logs on, various system routines are automatically specified to handle interrupts. For example, the command system receives control and performs a read from SYSIN. It also handles attention interrupts from conversational tasks.

The task monitor consists of privileged service programs that receive and process task oriented (programmed) interruptions in a prescribed sequence and on a priority basis via queueing, scanning, and dispatch mechanisms. As in the supervisor, a queued interruption represents an element of work. Such an element may be deferred for reasons of priority, efficiency, or protection against recursion.

The topic "User Interrupt Control" in the section "TSS for the Application Programmer" indicates the kinds of service provided by the task monitor. Using these services, it is possible to write multitasking applications within a TSS task. Other such applications use intertask communication to synchronize and coordinate processes.

Virtual Access Method

VAM involves several different aspects of direct-access storage usage in TSS, including volume layout, physical transfer of data between main storage and external storage, internal data set structure, and service routines. The integration of virtual storage and data management, which is a major feature of TSS design, offers many advantages and conveniences. In this context, the term *access method* has a broader meaning than when used in connection with designs that make external storage accessible only in an indirect fashion.

One of the most important design points of TSS is that main storage, virtual storage, and external storage, should be addressable as sets of pages, numbered from zero. This would be impossible for some classes of I/O devices, such as card readers and printers. For other classes of I/O devices, such as tapes, ***page-addressability*** would be possible only in a restricted sense. Main storage can be addressed randomly, and therefore may be regarded as page-addressable. Dynamic address translation has page-addressability as the basis of its design. Direct-access storage can

be addressed by page and is well suited for external storage. Thus, the three main types of storage used in TSS, main storage, virtual storage, and external storage, are either inherently, or by convention, page-addressable.

VAM, in the broadest sense, defines dynamically changing relationships between pages in virtual storage and pages in external storage. This enables data on external storage, including programs, to be directly addressed by the CPU via dynamic address translation. Direct addressability of external storage, made possible by VAM, contributes to efficient operation of the system and provides conveniences related to system development.

The sets of pages represented by main storage, virtual storage, and external storage devices need not be dense. Pages may be missing from main storage, because individual storage units have been assigned for other purposes such as maintenance. Virtual storage is organized into segments, leaving gaps in the address space. Direct-access volumes may have missing pages because of surfaces which have become defective. In all cases, a page that exists can be uniquely identified by specifying the device on which it resides and the relative page number on that device.

Whether the "device" is a direct-access volume, a collection of one or more consecutively numbered segments in virtual storage, or a collection of main storage units, the form of a page address is the same. For example, the only difference between one type of disk device and another is the number of pages each type holds, not the way in which an individual page is addressed. This homogeneity of external storage addressing enables device independence for data on external storage.

Viewed externally, a VAM data set is a collection of logically contiguous pages, numbered from zero. All structure within a VAM data set, including indexes and partitioned organization directories, is specified in terms of pages, numbered from zero, that are in the data set. VAM defines and maintains the ***mapping*** of pages of a data set to one or more direct-access volumes. Because the format of a page address is constant across all direct-access volume types, data set pages may be allocated arbitrarily from an arbitrary set of direct-access volumes which need not all be the same type. VAM data sets may grow and contract freely, because additional pages may be allocated from any volume with available space. Because a data set need not be constrained to occupy physically contiguous areas on a direct-access volume, a page in a data set that is currently unused need not occupy a physical page. For example, a deleted or replaced member of a VAM ***partitioned data set*** does not occupy physical space. Thus, direct-access storage is conserved without the need for ancillary operations such as data set compression. Similarly, when an entire data set is erased, space it occupied is immediately available for allocation to other data sets, because VAM imposes no contiguity requirement.

When a direct-access volume is formatted for use with VAM, its entire surface (with the exception of the track containing the volume label and the IPL bootstrap) is formatted with page-size blocks. Space allocation and data set control blocks are stored in page-size blocks and therefore can be made directly addressable by virtue of their being mapped into

virtual storage. The efficiency of VAM is thus extended to maintaining allocation records.

All physical transfer of information to and from VAM volumes is accomplished in page-size blocks. Therefore, the system never needs to execute a formatting write operation on a VAM volume. This increases the availability of direct-access device control units, because they are never busy erasing to the end of a track. Also, indexed and partitioned directories are structures internal to the data set, not physical entities on direct-access tracks. Therefore, these structures can be expanded and contracted, as necessary, simply by allocating additional pages to the data set. This independence of direct-access volume format from the internal structure of the data sets stored on it is an important factor in the effectiveness of VAM.

Direct-access storage devices are also used in TSS to store (page) virtual storage that is not immediately needed by programs in execution. The direct-access volumes used for this purpose are formatted as VAM volumes. Also, public storage volumes may have areas reserved for paging. This commonality of direct-access volume formats permits one set of I/O routines to perform all physical transfers of information between main storage and external storage.

Another advantage of using common volume formats and I/O routines for both auxiliary and external storage is the ease with which virtual storage pages can be transferred between the two. For example, consider a program stored in a data set on external storage. When it is to be loaded, the appropriate pages of the data set containing the program are mapped into a task's virtual storage. Later, if portions of the program are to be paged out, it is possible that the pages may be written on an auxiliary storage device which has a shorter access time than the device used for external storage. Also, when a task needs to transfer a page of its virtual storage to a data set, it can be done simply by transferring the page from auxiliary storage to external storage. This is another example of keeping system overhead low when transferring data between the three main storage types: main storage, virtual storage, and external storage.

The preceding discussion related to those aspects of VAM which increase system efficiency and flexibility. The following shows how the underlying system design, particularly shared virtual storage, helps VAM. When a VAM data set is being processed, a table is constructed which contains, for each data set page, the external storage page (if any) associated with that data set page. This table is called the relative external storage correspondence table (RESTBL) and is in shared virtual storage provided by the operating system. When a data set is shared by more than one user, VAM must prevent multiple tasks from changing or adding the same record, and individual tasks from changing a record that another task is reading. This is accomplished by adding information concerning the use of each page and the type of access to the RESTBL. By placing the RESTBL in virtual storage that is shared by all current users of the data set, one set of common status information is available to control concurrent access to the data set. An ordinary user program obtains the advantages of protection without specifically taking any action.

Appendix A: TSS Commands

This appendix presents TSS commands of general interest, organized by functional area. TSS commands are fully described in publications such as *Command System User's Guide, System Programmer's Guide, Manager's and Administrator's Guide, Operator's Guide,* and *Time Sharing Support System.* In the descriptions that follow, the term *job* means a nonconversational task or a bulk output request. Commands identified by *, issued by the system operator, system manager, or system programmer, can perform additional functions or are not limited to the USERID of the task from which the command was issued.

Task Management Commands

&	Save the current task-related performance data. Upon the next use of the & command, subtract previously saved data and write the results on SYSOUT. The & command is used in pairs to identify the measurement period.
%	Write the task-related performance data for the command prefixed by the % command on SYSOUT.
@	Write the task-related performance data accumulated since LOGON, on SYSOUT.
ABEND	Terminate the current task, and replace it with a new task.
ABENDREG	Display the contents of the general registers and the last instruction location corresponding to the most recent ABEND.
BACK	Convert a conversational task to a nonconversational task, which will use the specified data set as SYSIN.
BLOCK *	Block dispatch of a job.
CANCEL *	Cancel a job.
CHGPASS	Change the LOGON password of the USERID issuing the command.
EXECUTE	Initiate a nonconversational task, which will use the specified data set as SYSIN.
EXHIBIT *	Display the status of a job or present a list of active tasks.
JOBS *	Display a list of the jobs in the system.
IPL?	Display the time of the last system startup.
LOGOFF	Terminate the current task.
LOGON	Create a task and perform initialization based on the profile stored for the specified USERID.
STATUS *	Display the status of a specified job or type of jobs.
SUMMARY *	Display a summary of job statistics.

TID?	Display the TASKID for a specified USERID or job.
TIME	Set a CPU time limit for the current task.
TIMINGS *	Present system performance as measured in jobs, CPU time, percent elapsed time, and seconds per job.
UNBLOCK *	Reverse the effect of a BLOCK command.
USAGE *	Display the resource utilization statistics.
ZLOGON	A user-written procedure, ZLOGON is executed by the system for the user during each LOGON.

Command Environment Commands

BUILTIN	Define a command which can then be used to invoke a user-written program as a command.
DEFAULT	Add, replace, or delete entries to the default portion of the task's combined dictionary.
EJECT	Skip to a new page of SYSOUT for nonconversational tasks or space three lines on the terminal for conversational tasks.
EXIT	Bypass execution of the current program or command, and execute the next command in the source list.
EXPLAIN	Provide additional information on messages, terms in messages, the origin of messages, and expected responses for prompting messages.
GAV	Display all entries in the task's combined dictionary.
GDV	Display the value for a default in the task's combined dictionary.
GOTO	Branch forward in a PROCDEF.
GSV	Display the synonym value associated with a specified term.
INPUT	Add a data set to the stack of secondary SYSIN data sets. (Secondary SYSIN data sets are used when the default for SYSIN is S.)
INPUT?	Present the DDNAMEs and DSNAMEs in the secondary SYSIN stack.
JUMP	Take the next command in a secondary SYSIN data set from a specified record.
KEYWORD	Display the names of operands of a command.
MCAST	Change the function control characters in the task's combined dictionary.
MCASTAB	Alter the terminal input and output translation tables.
NEWMLF	Flush the stack of saved messages obtained from USERLIB(SYSMLF) to guarantee use of updated messages.

OUTPUT	Add a data set to the stack of secondary SYSOUT data sets. (Secondary SYSOUT data sets are used when the default for SYSOUT is S.)
OUTPUT?	Present the DDNAMEs and DSNAMEs in the secondary SYSOUT stack.
PRMPT	Write a specified message on SYSOUT. PRMPT is used when testing new messages or when a repeat of a previous message is desired to obtain more information by means of the EXPLAIN command.
PROCDEF	Invoke the editor for writing or modifying a user command procedure.
PROFILE	Make permanent, any changes to the task's combined dictionary.
PUSH	Save the status of an interrupted user program.
RTRN	Cancel further execution of commands in the source list, flush the stack of active programs, and read the next command from SYSIN.
SPACE	Space the specified number of lines on SYSOUT.
STACK	Display the names in the stack of active user programs.
STRING	Display the commands in the source list that are not yet executed.
SYNONYM	Rename commands, keywords, or command symbols.

Terminal Control Commands

ATTEN	Control ignoring of attention interrupts.
BLIP	Transmit a special nonprinting assurance message to the terminal in order to notify the user that the computing system and the data path to the terminal are working.
BLIP?	Display the current setting of BLIP parameters.
DCMD	Execute a device control command from within a PROCDEF.
HRDCPY	Produce a hard copy of terminal input and output.
HRDCPY?	Display the current HRDCPY status.
KA	Do not fold the EBCDIC character set on input.
KB	Fold the EBCDIC character set on input.
LL	Define the maximum length of any line sent to SYSOUT and specify the action to be taken when the line length is exceeded.
LL?	Display the current setting of the LL parameters.
INTAB	Specify input tab locations.
INTAB?	Display the values of input tabbing controls.
OUTTAB	Specify output tab locations.

OUTTAB?	Display the values of output tabbing controls.
RESTART	Restart continuous reads from the terminal (input buffering mode).
TRANSLAT	Perform immediate alteration of the translate tables used for SYSIN and SYSOUT.

Program Execution Commands

AT	Replace an instruction in a user program with a call to the program control system (PCS) and save the replaced instruction. If control reaches the original location of the replaced instruction, perform the action specified in the remainder of the PCS statement. If execution resumes at this point, execute the replaced instruction.
BRANCH	Change the control path of a program or resume execution at a different location.
CALL	LOAD a program and cause it to be executed, passing any specified parameters.
DISPLAY	Present values of variables, contents of machine registers, and specified areas of the address space.
DUMP	As for DISPLAY, but write data in a VISAM data set, defined with a DDNAME of PCSOUT.
GO	Resume execution of an interrupted program.
IF	Execute the command following the IF command if the condition specified in the IF command is true.
LOAD	Map a program (or data in object module format) that resides on external storage, to virtual storage. Loading reserves space in virtual storage and makes entries in the task dictionary. The first time a page of the virtual storage is referenced, a translation exception occurs and the map is used to find the location in external storage which contains the information corresponding to the page. Thereafter, the page is a part of the address space.
QUALIFY	Indicate which internal symbol dictionary is to be used by PCS for reference to program variables and locations, in the absence of explicit specification.
REMOVE	Reverse the replacement performed by a specified AT statement, that is, remove the call to PCS and restore the overlaid instruction.
SET	Change the contents of machine registers, values of program variables, locations in virtual storage, or establish and initialize command symbols.
STOP	If executed due to an AT statement, cause execution of a user program to be interrupted and control passed to SYSIN; otherwise, display the current user program instruction location counter.

TRAP	Equivalent to an AT statement except that the condition causing transfer of control to PCS is determined by the program event recording (PER) hardware of System/370. Conditions monitored are fetches and/or stores into virtual storage locations, alteration of the contents of machine general registers, and successful branches. In the case of successful branches, PCS examines all successful branches and selectively detects branches into specific ranges of instruction locations.
UNLOAD	Reverse the effect of a previous LOAD command.

Data Management Commands

CATALOG	Add a data set name to the catalog, change a data set name entry in the catalog, or create a generation data group.
CDD	Retrieve and execute specified DDEF commands from a line data set.
CDS	Copy a data set or a member of a data set.
CLOSE	Close one or all open, nonsystem, data sets.
DDEF	Specify a symbolic data definition name and establish the connection between a program and a data set.
DDNAME?	Present the data definition names of a task's currently defined data sets.
DELETE	Remove data set names from the catalog.
DMPRST	Dump or restore a private VAM volume to tape or disk.
DSS? *	Present the characteristics and status of a data set or partially qualified data set name.
ERASE	Uncatalog and free the space occupied by direct-access data sets.
EVV *	Enter (catalog) data sets on a private VAM volume.
FILEDEF	Perform the equivalent of DDEF for OS/VS programs.
FILEREL	RELEASE the definition established by FILEDEF.
JOBLIBS	Rearrange the job library list.
LTDS	List the names of data sets stored on VT-format tape volumes.
OSDD?	Perform the equivalent of DDNAME? for data sets defined with the FILEDEF command.
PC?	Present the catalog information for a data set or partially qualified data set name.
PERMIT	Allow and disallow other users to access one's data sets.
POD?	Present information from the partitioned organization directory of a VPAM data set.

RELEASE	Release the data definition established with a DDEF command.
RET	Change the retention attribute of a data set.
SECURE	Specify and wait for the set of private devices required for further execution of a nonconversational task.
SHARE	Gain access to data sets owned by another user, if the owner has allowed access with the PERMIT command.
SYSINDEX	Build the index component for a symbolic library, using the symbolic component.
TV	Import a VAM data set which is on a tape volume.
VT	Export a VAM data set by means of a tape volume.
VV	Copy a VAM data set.

Editor Commands

CONTEXT	Replace a string of characters or hexadecimal digits within a line or range of lines with another character or hexadecimal string, replacing every occurrence of the first string by the second string.
CORRECT	Change or insert characters or hexadecimal digits for a line or range of lines, using correction control specifications read from SYSIN.
DISABLE	Cancel the effect of a previous ENABLE command and record all subsequent changes in the transaction table for use by the STET command.
EDIT	Invoke the editor.
ENABLE	Clear the transaction table of entries when the next command that alters data is issued. Until the editor is disabled again, record in the transaction table only those changes made by the most recent command.
END	Close the data set being edited and stop editing.
EXCERPT	Insert a range of lines from a data set (including the one being edited) into the data set being edited.
EXCISE	Remove a line or a range of line.
INSERT	Insert line(s).
LIST	Display a line or a range of lines.
LOCATE	Search a range of lines for a specified character string.
NUMBER	Renumber a line or a range of lines.
POST	Clear the transaction table of entries.
REGION	Change the region currently being processed.

REVISE	Excise a range of lines and insert new lines.
STET	Reverse the effect of changes recorded in the transaction table. If the editor is enabled, cancel the effect of the last editor command that altered data; if the editor is disabled, cancel the effect of all editing that was done since the DISABLE command was issued.
UPDATE	Enter a mode to read from SYSIN and update the data set being edited. When operating in this mode, the editor processes update requests which are free from constraints imposed by TSS command syntax.

Data Editing Commands

CONVMAC	Convert a macro library from line data set format to region data set format.
DATA	Create a line data set or a VSAM data set.
EDIT	Invoke the TSS editor.
END	Close the data set being edited by the TSS editor and stop editing.
LINE? *	Display all or portions of a line data set or a listing data set produced by the TSS language processors.
MODIFY	Edit VISAM or VSAM data sets. This command is a predecessor of the TSS editor and can be used for editing most data sets having a format which the TSS editor cannot edit.
VDMP *	Dump all or portions of a data set.
VDSP *	Display all or portions of a data set.
VPAT *	Patch (perform permanent alterations to) a data set.

Language Processing Commands

ASM	Invoke the TSS Assembler.
COBOL	Invoke the OS/VS COBOL program product.
FTN	Invoke the TSS FORTRAN Compiler.
FTNH	Invoke the OS/VS FORTRAN program product.
HASM	Invoke the OS/VS Assembler H program product.
LNK	Invoke the TSS Linkage Editor.
ODC	Invoke the object data converter to convert OS/VS object modules to TSS object modules.
PLI	Invoke the TSS PL/I F Compiler.
PLIOPT	Invoke the OS/VS PL/I Optimizing Compiler program product.

Bulk Output Commands

PRINT *	Cause a data set to be printed.
PUNCH	Cause a data set to be punched.
WT	Cause a data set to be written on tape for offline printing.

Operator, Manager, and System Programmer Commands

ALTER	Update TSS source, generating standard sequence numbers.
ASNBD	Assign or delete unit-record equipment from BULKIO.
BCST	Broadcast a message to all users.
CC	Check catalog validity.
CPS	Clean public storage of unwanted data sets.
CVV	Catalog uncataloged data sets on public VAM volumes.
DIRECT	Reroute RJE output.
DONEXT	Put a job at the head of a queue.
DROP	Restore a device from HOLD status.
EREP	Process RMS records.
FLOW	Control external scheduling parameters.
FORCE	Cause a conversational task to be abnormally terminated.
HOLD	Prevent a device from being used except for servicing.
JOIN	Define a new user to the system.
JOINRJE	Define a new RJE station ID to the system.
LPDS	List public data sets.
MAPGEN	Generate real and virtual storage maps.
MC	Perform catalog maintenance operations.
MODE	Set operational mode of RMS.
MONIT	Activate the system activity display.
MSG	Send a message to a conversational user.
MTT	Initiate an MTT application.
MTTDCN	Terminate an MTT application.

NEWMSG	Flush the stack of saved messages obtained from SYSLIB(0)(SYSMLF) to guarantee use of updated messages.
PARTS?	Display the number and status of batch partitions.
PATCLEAR	Logically initialize a VAM volume.
PATFIX	Perform diagnostic reporting and repair on VAM volumes.
PPREAD	Convert a licensed program distribution tape for use with TSS.
QUIT	Remove a USERID and associated data sets from the system.
QUITRJE	Remove an RJE station ID from the system.
REJOIN	Alter the JOIN parameters for a USERID.
REPLY	Reply to a WTOR message.
REPLY?	Present outstanding WTOR messages.
RPS	Restore public storage.
RT	Initiate a task to read a tape data set for a USERID.
SARD	Display the system activity and resource utilization.
SETMAX	Set limits for print jobs and number of private devices per batch job.
SETPARTS	Define the set of batch partitions.
SHUTDOWN	Terminate all active tasks including the operator's task.
SNAP	Provide a hard copy of displays produced with the MONIT command.
SYNCCAT	Synchronize user and system catalogs with public storage.
UPDTUSER	Synchronize the count of public storage pages in the user accounting record with actual storage usage and erase the temporary data sets not in use.
VARY	Partition specified hardware components online or offline.
VMEREP	Execute the EREP command with recommended parameters.
VSS	Place one's task in virtual support system command mode.

Time Sharing Support System Commands

AT	Replace an instruction in a program with a call to the time sharing support system (TSSS) and save the replaced instruction. When control reaches the original location of the replaced instruction, perform the action specified in the remainder of the TSSS statement. If execution resumes at this point, execute the replaced instruction.
CALL	Initiate execution of a prestored set of command statements.
COLLECT	Move data from a specified area into a specified collection area.
CONNECT	(RSS command) Connect VSS to the terminal of a specified task.
DEFINE	Define a temporary symbol and allocate any storage needed.
DISCONNECT	Reverse the effect of a VSS command or CONNECT command.
DISPLAY	Present the data requested on the TSSS terminal.
DUMP	As for DISPLAY, but present the data on the specified TSSS output device.
END	Stop reading statements initiated with the CALL command.
IF	Indicate the start of a conditional statement or portion of a statement. Execute the remainder of the statement if the condition specified in the IF command is true.
PATCH	(See SET command below.) Perform the SET function and keep a record of the alteration to be used in case restoration is desired.
QUALIFY	Establish implicit *real memory*, *virtual memory*, or *global* qualification for subsequent commands.
REMOVE	Reverse the replacement performed by a specified AT statement, that is, remove the call to TSSS and restore the overlaid instruction. Also, restore the contents of a data field altered by the PATCH command.
RUN	Cause control to revert to TSS at the point of interruption where TSSS took control, or at a specified location.
SET	Alter the contents of a specified data field.
STOP	In an RSS AT statement, cause TSS to halt. In a VSS AT statement, cause the task to halt.

Appendix B: TSS Macros

This appendix presents TSS macros of general interest, organized by functional area. TSS macros are fully described in publications such as *Assembler User Macro Instructions*, *Multiterminal Task Programming and Operation*, *System Programmer's Guide*, and *System Generation and Maintenance*.

Data Set Specification Macros

DDEF	invokes the DDEF command processor to provide the connection between a program and a data set.
CDD	calls the DDEF command processor with one or more DDEF commands obtained from a line data set.
FINDDS	locates the JFCB corresponding to a given data set name, and optionally creates a JFCB (invokes DDEF) if the data set name is in the catalog.
FINDJFCB	locates the JFCB corresponding to a given DDNAME, and optionally creates a JFCB (invokes DDEF).
CAT	invokes the CATALOG command processor to catalog data sets, rename data sets, or create generation data groups.
REL	invokes the RELEASE command processor to dispose of the specified JFCB, freeing the symbolic name of the corresponding DDEF statement for other use. Devices used for data sets on private volumes are optionally released for general use. RELEASE is used to free data sets from concatenation and to close and remove data sets from the job library chain. A RELEASE of the symbolic name of the DDEF statement associated with an open data set results in that data set being closed. Any programs loaded from a job library are unloaded by releasing a job library.
DEL	invokes the DELETE command processor to remove data set names from the catalog.
DCB	defines storage for a data control block.
DCBD	generates a dummy control section (DSECT) to describe the DCB with names having the appropriate attributes for DCB fields.

Data Control Block Processing Macros

OPEN	collects the attributes of specified data sets from various sources, by priority, and merges the information in the respective DCBs. OPEN prepares a DCB and the data set associated with it for processing.
CLOSE	reverses the action of OPEN. CLOSE waits until I/O requests are complete before proceeding. When appropriate, output data set trailer labels are processed and access to volumes is positioned as specified. Control blocks, such as the DCB and JFCB, are restored to their original condition. CLOSE disconnects a data set from further processing and user access. For BSAM and VAM DCBs there is a CLOSE option that causes the same processing as the standard CLOSE macro except that fields of the DCB are not restored to their status before OPEN; the DCBs are in effect open and additional processing may be performed. With BSAM data sets, temporary close is useful for repositioning a volume for subsequent processing and serves the purpose of completing the data set (if it has just been written or extended). In the case of VAM data sets, temporary close causes the DSCBs to be written which captures the current status of the data set on external storage.

Virtual Sequential Access Method Macros

GET	reads logical records in sequential order.
PUT	writes logical records in sequential order.
PUTX	replaces a logical record, previously read by GET.
SETL	logically positions access to a data set at the beginning or end, at the previous record, or at any logical record within a sequential data set. Subsequent PUT or GET operations will proceed from this new position.

Virtual Index Sequential Access Method Macros

GET	reads logical records in sequential order, by key.
PUT	writes logical records in sequential order, by key.
READ	reads logical records in nonsequential or sequential order.
WRITE	writes logical records in nonsequential or sequential order.
DELREC	deletes a specified logical record from a data set.
SETL	logically positions access to a data set at its beginning or end, at the previous record, or at any logical record. Subsequent PUT or GET operations will proceed from this new position.
ESETL	releases a read-lock set by other operations.
RELEX	releases a write-lock set by other operations.

Virtual Partitioned Access Method Macros

FIND	opens an individual member within a VPAM data set for processing. After FIND, appropriate VISAM or VSAM macros can be used to process the records within the member.
STOW	causes a VISAM or VSAM member of a partitioned data set to be added to or deleted from the data set. It also adds, changes, deletes, or replaces member names or aliases, and provides for storing additional information in the partitioned organization directory (POD), as user data.

Basic Sequential Access Method Macros

READ	reads a physical record from an I/O device and specifies or defines a data event control block (DECB) to be used to indicate completion status for the operation. After READ, control is returned to the user program. The user program is responsible for deblocking logical records from physical records.
WRITE	is the same as READ except that data transfer is in the opposite direction.
CHECK	tests the queue of DECBs associated with READ or WRITE operations to determine if the operations are complete and if so, whether errors or exceptional conditions occurred.
DQDECB	removes all unchecked DECBs associated with READ and WRITE operations for a specified device. DQDECB is used when restarting I/O after user program action on error conditions.
NOTE	makes available to the program, for use with POINT, the relative position within a volume of the last block read or written.
POINT	repositions access to a data set at a specified block within the data set.
BSP	backspaces one physical record or block on the current tape or direct-access volume regardless of the direction in which data is being stored or retrieved on that device.
CNTRL	controls tape positioning and writing of tape marks. CNTRL can be used to obtain sense data from tape or direct-access devices.
FEOV	positions access to the data set at the next volume of a multivolume set.
GETPOOL	requests allocation of virtual storage for use as a buffer pool and assigns that area to a DCB.
GETBUF	obtains a buffer work area from a buffer pool previously assigned to a DCB either by a GETPOOL macro or as provided according to DCB buffer options.
FREEBUF	returns a buffer work area obtained by GETBUF to the related buffer pool.

FREEPOOL releases areas previously assigned to specified DCBs as buffer pools either by a GETPOOL macro or as a result of buffer options specified in the DCB.

Queued Sequential Access Method Macros

GET reads logical records in sequential order. The initial GET causes a physical record from the input device to be transferred to a system-maintained buffer area and makes the first logical record available to the user program. Each subsequent GET delivers logical records until all logical records within the physical record have been processed. Meanwhile, the next physical block is transferred.

RELSE causes the remaining records of the current input buffer to be ignored and positions access to the data set at the first logical record of the next physical record. The next GET macro will retrieve the first logical record from the new input buffer.

PUT is the same as GET except that data transfer is in the opposite direction.

PUTX replaces a logical record, previously read by GET, or writes an updated or identical logical record directly from an input data set to an output data set.

TRUNC causes the current output buffer to be regarded as if it were filled. The output buffer is written to the output device, leaving access to the data set positioned at the next buffer area. The next PUT issued is for the first record of the next block.

SETL logically positions access to a data set at its beginning or end, at the previous record, or at any logical record. Subsequent PUT or GET operations will proceed from this new position.

CNTRL controls tape positioning and writing of tape marks. CNTRL can be used to obtain sense data from tape or direct-access devices.

Multiple Sequential Access Method Macros

GET reads a card image from a card reader. The access method buffers I/O operations, freeing the user program from such concerns. Many cards are read with one chain of channel command words (CCWs). The supervisor can chain successive requests from a task together so that reading proceeds at the maximum rate of the device with a minimum of interrupts.

PUT writes records on a printer or punch. As with GET, the access method buffers I/O operations.

SETUR specifies the configuration for printers and punches. SETUR communicates with the operator regarding forms (paper and cards) to be placed in the devices, and loads buffers associated with printers, if applicable.

FINISH is used to indicate to the access method that processing of a data group (a subsection of an MSAM data set but a complete data set to the user of a device supported with MSAM) is complete. FINISH causes input and output buffers to be flushed.

Input/Output Request Access Method Macros

IOREQ	initiates a request for an I/O operation specified by a user-written channel program and specifies or defines a data event control block (DECB) to be used to indicate completion status for the operation. After IOREQ, control is returned to the user program.
CHECK	tests the queue of DECBs associated with IOREQ operations to determine if the operations are complete and if so, whether errors or exceptional conditions occurred.
DQDECB	removes all unchecked DECBs associated with IOREQ operations to a specified device. DQDECB is used when restarting I/O after user program action on error conditions.
VCCW	defines storage for a virtual channel command word (VCCW). A VCCW serves same function as the a CCW. Chains of one or more VCCWs specify I/O operations to be performed.

Copy Data Set Macro

COPYDS	invokes the CDS command processor to copy data sets or members of partitioned data sets.

Bulk Output Macros

PR	invokes the PRINT command processor to cause a data set to be printed.
PU	invokes the PUNCH command processor to cause a data set to be punched.
WT	invokes the WT command processor to cause a data set to be written on tape for offline printing.

Erase Data Set Macro

ERASE	invokes the ERASE command processor to uncatalog and free the space occupied by direct-access data sets.

SYSIN/SYSOUT Communication Macros

GATRD	reads a record from SYSIN and places it in a user-designated virtual storage area.
TGATRD	extended function form of GATRD macro.
SOLICIT	presents a continuously incremented number as a prompt to TGATRD operations.
GATWR	writes a record on SYSOUT.
TGATWR	extended function form of GATWR macro.
TGATWS	writes a record on the primary SYSOUT.
TWRTLST	writes records from a list of virtual storage areas to SYSOUT.

GTWRC	writes a record on SYSOUT. The first byte of the record is used for carriage control when printing nonconversational SYSOUTs. Carriage control action is approximated for conversational tasks.
GTWAR	writes a record on SYSOUT and reads the next available record from SYSIN and places it in a user-designated virtual storage area.
TGTWAR	extended function form of GTWAR macro. If input buffering is in effect, the write operation is suppressed.
GTWSR	writes a record on SYSOUT and reads the response to that record, placing it in a user-designated virtual storage area. If issued in a nonconversational task, unless the user has indicated otherwise, the task will be terminated.
TGTWSR	extended function form of GTWSR macro. If input buffering is in effect, the write operation is performed immediately and the read operation in response to that write is performed immediately.
SYSIN	optionally writes a record on SYSOUT and reads a record from SYSIN into virtual storage. If the record is recognized as a command, it is placed in the source list for subsequent processing by the command analyzer. User programs can detect the incidence of commands and take action accordingly. Otherwise, the user program is interrupted and the command is processed.
TCNTRL	specifies miscellaneous control operations.
CHCKT	checks the status of a DECB related to SYSIN/SYSOUT operations.
TRCBUF	reads a record from the conversational buffer for the terminal and places it in a user-designated virtual storage area.
TDCMD	issues device control commands from user programs to control the terminal environment.
TCLEAR	purges any pending or active request buffers on SYSIN/SYSOUT.
TFREE	disconnects a secondary SYSIN/SYSOUT from the task.
TRANLCD	makes the address of a specified terminal translation table available to a user program.
MCAST	temporarily substitutes a user-specified character translation table and function control table. The character translation table specifies substitution of character codes for transfer of data between user programs and SYSIN and SYSOUT. The function control table identifies characters which are to have special effects, for example backspace to mean overstrike, not character correction.
ATTNSAV	saves current conditions (buffers and terminal environment) in a pushdown stack.
ATTNRST	restores previously saved conditions and buffers, disposing of the current conditions.

ATTNDST	disposes of saved conditions and buffers no longer needed.
TERMPRO	saves or restores the terminal environment in a specified data set.
EXLIST	specifies the locations in the user program which are to receive control when events occur such as completions or interrupts.
PRMPT	invokes a system facility which prompts the user with messages from the system message file, if not from the user message file. The prompter analyzes responses to messages whose coding indicates that a response is required.

Virtual Storage Management Macros

GETMAIN	is used to acquire additional virtual storage.
FREEMAIN	releases virtual storage acquired with GETMAIN.
CKCLS	determines the most restrictive protection class assigned to a specified number of contiguous halfpages of virtual storage.
CSTORE	saves contiguous virtual storage areas in object module format.
RSVSEG	associates a name with a contiguous set of virtual storage segments.
DISCSEG	disconnects a segment group from a virtual address space and assigns a name to it.
CONSEG	connects a disconnected segment group to an unassigned portion of a virtual address space.
RELSEG	releases a reserved segment group, deleting the name, but leaving addressable the virtual address space of the group.
DELSEG	deletes a disconnected segment group. The name and any space on auxiliary storage are deleted.

Program Linkage Macros

LOAD	explicitly loads a program, if it is not already loaded, into virtual storage. The address at which the program has been loaded can be obtained from address constants previously defined by an ADCON or ARM macro. The program remains in virtual storage until it is unloaded by a DELETE macro or an UNLOAD command.
CALL	explicitly or implicitly loads the called program into virtual storage and establishes conventional linkage between the calling and called program. The address at which the program has been loaded can be obtained from address constants previously defined by an ADCON or ARM macro. CALL causes control to be given to the called program.
ARM	initializes the address constant group defined by an ADCON macro with the name of the program, entry point, or control section that is to be loaded into virtual storage. The initialized address constant group can subsequently be used by a CALL or LOAD macro to explicitly load the program.

ADCON — generates a group of address constants for use by CALL, LOAD, or DELETE macro instructions.

ADCOND — generates a DSECT to describe the address constant group with names having the appropriate attributes. These names make it possible for an assembler language program to reference symbolically the resolved address constants and control flags placed in the group during execution of a LOAD or explicit CALL macro.

DELETE — unloads an explicitly loaded program that is no longer needed, freeing virtual storage. Any associated programs are also deleted.

SAVE — stores the contents of the general registers according to a standard convention. The SAVE macro is normally the first instruction in a called routine.

RETURN — restores the contents of the general registers according to a standard convention and returns control to the calling routine, optionally setting a return code for the calling routine.

BLIST — provides for one-line specification of multiple branch validation and for analysis of the return code from a called program.

Interrupt Handling Macros

SIR — specifies a user interrupt routine (named via a SPEC, SAEC, SIEC, SEEC, STEC, or SSEC macro, according to the type of interrupt) to the task monitor. SIR specifies the processing priority for that routine. The user's routine replaces any system-supplied interruption servicing routines for this type of interruption, unless the user's routine is deactivated with the DIR macro. System-supplied routines are reinstated after the user routines are deleted.

DIR — deletes an interruption servicing routine, reversing the effect of the corresponding SIR macro.

SPEC — names a user-written program interruption servicing routine and defines an interrupt control block (ICB) in which data pertaining to a program interruption can be recorded. The named routine will be used when it is defined to the task monitor as an interruption servicing routine by a SIR macro.

SSEC — names a user-written SVC interruption servicing routine and defines an ICB in which data pertaining to an SVC interruption can be recorded. The named routine will be used when it is defined to the task monitor as an interruption servicing routine by a SIR macro.

SEEC — names a user-written external interruption servicing routine and defines an ICB in which data pertaining to an external interruption can be recorded. The named routine will be used when it is defined to the task monitor as an interruption servicing routine by a SIR macro.

SAEC — names a user-written asynchronous interruption servicing routine and defines an ICB in which data pertaining to an asynchronous interruption can be recorded. The named routine will be used when it is defined to the task monitor as an interruption servicing routine by a SIR macro.

STEC — names a user-written timer interruption servicing routine and defines an ICB in which data pertaining to a timer interruption can be recorded. The named routine will be used when it is defined to the task monitor as an interruption servicing routine by a SIR macro.

SIEC — names a user-written I/O interruption servicing routine and defines an ICB in which data pertaining to an I/O interruption can be recorded. The named routine will be used when it is defined to the task monitor as an interruption servicing routine by a SIR macro.

INTINQ — inquires about the interruption information recorded in a specified ICB. Various options are available. Control can be relinquished until the ICB indicates an interrupt. If the interrupt has been queued, the routine in which the INTINQ is issued may regain control immediately. Also, the task can be made to wait until a corresponding interrupt has occurred. Interruptions queued on the ICB can be cleared by INTINQ. A specified branch can be taken if the interrupt information is present.

SAI — saves the task's current interruption servicing status indicator and inhibits further interrupts until a RAE macro is issued. Interruptions occurring while the inhibit indicator is on are saved and queued for later servicing.

RAE — restores the interruption servicing status previously saved by an SAI macro. Depending on the saved status (enabled or inhibited), processing continues. If interrupts were previously enabled, any interruptions that occurred while interruption servicing was inhibited are processed before processing continues.

PIREC — efficiently tests an address for validity. Program interrupt codes 4, 5, and 6 occurring when PIREC is being executed are not processed in the normal manner. Detection of an invalid address results in a branch to a specified location.

USATT — causes subsequent attention interruptions to be processed by a user-written routine that was previously established as an interruption servicing routine by the SIR and SAEC macros.

CLATT — reverses the effect of a USATT macro. Control of attention interruptions obtained with a USATT macro is relinquished.

AETD — causes attention interruptions to be processed by any one of several user-written routines, depending on the number of times the attention key is pressed. The AETD macro is also used to relinquish control of attention interruptions acquired by the AETD macro.

Timer Maintenance Macros

STIMER — sets a software interval timer, measuring either task execution time or real time, and indicates what action should be taken when that specified time interval has elapsed.

TTIMER — tests an interval timer previously set by the STIMER macro and indicates the time remaining in that interval. It can also be used to cancel a previously specified timer setting.

REDTIM — provides time as a double precision, fixed-point number in microseconds. In TSS the epoch is March 1, 1900.

EBCDTIME — converts system-maintained time into specified EBCDIC formats. The time is expressed in some combination of years, months, days, hours, minutes, seconds, tenths of seconds, and hundredths of seconds.

Command System Interface Macros

BPKDS — generates all necessary linkage information and parameter storage areas required for use during the execution of a command that was defined with the BUILTIN command. Also, information from the BPKDS expansion is used by the KEYWORD command.

GDV — gets the value for a default from the task's combined dictionary.

GETDV — gets the value for a specified name and type from the task's combined dictionary.

SETDV — sets the value for a specified name and type into the task's combined dictionary.

OBEY — temporarily passes control to the command system for execution of a specified command. The command specified by OBEY will be issued just as if the user had interrupted the program and issued the command. When the command or a program invoked as a result of the command returns control to the command system, execution of the program from which the OBEY was issued will be resumed.

PAUSE — (for conversational tasks only) writes a user-specified message on SYSOUT and causes the task to enter command mode. A GO command causes execution of the program to resume. The interruption of a program by PAUSE is very similar to that which results from an attention interrupt. If the user has control of attention interruptions before issuing a PAUSE, the system regains control of them until a GO command is issued. PAUSE is ignored in a nonconversational task.

COMMAND — is the same as PAUSE except that it is not ignored in nonconversational mode. The SYSIN data set is read for the next command. Execution of the interrupted program can be resumed with a GO command.

CLIC — is the same as the PAUSE macro except that no message is issued.

CLIP — is the same as the COMMAND macro except that no message is issued.

EXIT — is a simple way of terminating execution of a program and optionally causing a predefined system message and a user-specified message to be written on SYSOUT. Control is returned to the command system and the next commands are taken from SYSIN.

ABEND — indicates an abnormal end condition to the user and the operator. The ABEND macro provides for various types of system action based on the severity code specified. Codes are: (1) Terminate execution of the program, returning control to SYSIN for conversational tasks; for nonconversational tasks either delete the task from the system or switch SYSIN to a data set defined with a DDNAME of TSKABEND. (2) Terminate the task, creating a new task if the old task was conversational. (3) Terminate the task, do not create a new task. (4) Terminate the task without attempts to write to SYSOUT. (Used by privileged programs only.) A message may be specified with the ABEND macro, either as the actual message or as the identification code of a message in the system or user message file.

Operator and System Log Communication Macros

WTO — writes a user-specified message on the operator's console.

WTOA — writes a user-specified action message on the operator's console. Action messages differ from those sent by WTO in that they are prefixed by characters intended to catch the operator's eye. They should only be used when action is required by the operator, otherwise the operator may disregard the action message format.

WTOR — writes a user-specified message on the operator's console. The user task waits for the operator to respond to the message. The operator is periodically reminded of unanswered messages. The reply from the operator is made available to the program. If the operator fails to reply within a reasonable time, the user can use the attention key to regain control and decide on some other course of action.

WTL — causes a user-specified message to be written in the system log data set. If the operator wishes to have WTL messages appear on the console, a default can be set in the combined dictionary of the operator USERID.

System Oriented Macros

AWAIT — tests for completion of an event and returns control to the task if the event is completed, or places the task in a delay state from which it will be removed when any task interruption occurs.

VSEND — sends a message from one task to another. The message is queued on the recipient task status index (TSI) as an external interrupt.

USAGE — causes resource statistics for a task to be made available for processing by a user program.

XTRTM — extracts and examines the total accumulated CPU time of the issuing task from the extended task status index (XTSI) of the task.

HASH — provides a hash value for a name.

LPCEDIT — invokes the editor, which can be used by a language processor controller for input of source statements.

LPCINIT — identifies the program which issues it as a language processor controller and initializes the editor for later use.

LIBESRCH determines if a specified object module is to be found in any of the job libraries and if so, which library.

CHDERMAC generates messages pertaining to errors encountered during macro expansion.

CHDVAL determines the type code of a parameter during macro expansion.

CHDPSECT changes the name of the current control section of an assembly to the name of the first PSECT for that assembly. If no PSECT exists, a branch to a specified location is generated.

Appendix C: Summary of TSS Publications

TSS publications summarized in this appendix are presented in groups according to the readership for which they are intended:

- Operations Management
- System Programmers
- Application Programmers

TSS publications are listed in the *IBM System/370 Bibliography*, GC20-0001.

Figure C-1 is the TSS publications plan. The publications are presented in a suggested order of reading, within each group.

This publication, *Concepts and Facilities*, introduces the concepts implemented in TSS and describes the facilities of the system. It is intended for managers of data processing installations, system programmers, application programmers and end users, and operators.

IBM Time Sharing System: Terminal User's Guide, GC28-2017, is intended for all users and gives instructions for operating the terminals used with TSS. Procedures are given for setting up the terminal, entering and canceling lines, correcting lines, and communicating with the system. Error conditions and termination procedures are also given. Appendixes list the character sets applicable to each terminal, and discuss terminal servicing operations such as changing ribbons, inserting paper, and setting tabs and margins.

The publication *System Messages*, is also intended for all users. It is a collection of all messages issued by the system and provides guidance and further explanation. In the interest of convenience and accuracy, it is published in machine-readable form.

Publications for Operations Management

IBM Time Sharing System: Manager's and Administrators Guide, GC28-2024, describes the facilities that are available for managing and allocating resources of a computer installation.

IBM Time Sharing System: Operator's Guide, GC28-2033, contains an overview of system operations, including functions of the operator. Operating procedures, commands used, messages, and typical operator responses to the messages are included.

IBM Time Sharing System: Independent Utilities, GC28-2038, describes utility programs that run under control of the utility support system (USS). USS supports stand-alone execution of programs outside the TSS environment. Included are descriptions of how to run utility programs that initialize direct-access volumes, dump and restore direct-access volumes, and maintain the data base. It also explains how to write utilities that operate in the USS environment and includes descriptions of USS macros. Also described is how to use a stand-alone storage dump program (not part of USS).

IBM Time Sharing System: Remote Job Entry, GC28-2057, explains how to use RJE terminals and includes information about RJE control statements. Also, commands used by the system manager and system operator to control the RJE facility are described.

Publications for System Programmers

IBM Time Sharing System: System Programmer's Guide, GC28-2008, describes the facilities available to system programmers for modification and extension of TSS. It also includes programming guidelines and examples of how to make TSS modifications, and a summary of the conventions that are to be respected when writing TSS programs.

IBM Time Sharing System: System Generation and Maintenance, GC28-2010, explains how to define and generate a system adapted to the requirements of a specific installation. It also explains how to incorporate IBM-supplied maintenance distributions and user modifications into existing systems.

IBM Time Sharing System: Time Sharing Support System, GC28-2006, describes the time sharing support system (TSSS) and the commands used to operate it. TSSS is an interactive facility for problem determination and maintenance. The support system can be used only by properly authorized system programmers; therefore, this publication contains no information required by users other than such system programmers.

IBM Time Sharing System: System Logic Summary, GY28-2009, describes the logic of the system, emphasizing the interrelationship of system components with respect to performance of system functions. It is the starting point for gaining an understanding of TSS design and enables the reader to relate an area of the system to a specific program logic manual.

IBM Time Sharing System: System Control Blocks, is supplied as a machine-readable source data set from which a listing of all system control blocks can be produced. This data set may also be used as input to programs which scan the entire source data base for the purpose of producing indexes and cross-references.

Listings of TSS programs are available on microfiche. The microfiche is produced using source programs from which the system is built and therefore, matches system code.

The program logic manuals describe the implementation of TSS design. They are organized to allow a reader to locate the coding within a component that applies to an area of interest. They also serve as guides to program listings.

IBM Time Sharing System: Quick Guide for System Programmers, GX28-6401, is a compact reference to TSSS, assembler language, and system control blocks, along with an appendix of machine control information.

Publications for Application Programmers

IBM Time Sharing System: Command System User's Guide, GC28-2001, describes the facilities of the TSS command system. It includes a brief description of conversational and nonconversational modes of operation, a detailed explanation of each command, typical terminal sessions, and command procedures. The rules for forming command statements are given.

IBM Time Sharing System: Data Management Facilities, GC28-2056, is a reference guide to TSS data management facilities. Topics covered include: storage classes, unit-record devices, data set characteristics, label formats, record formats, data set sharing, gaining access to data sets, and use of data management facilities. It contains information of use to assembler, FORTRAN, and PL/I users.

IBM Time Sharing System: Linkage Editor, GC28-2005, explains the use of the TSS linkage editor. Functions, data sources, destinations, and system inputs (commands and operands) required to link-edit programs are described.

IBM Time Sharing System: Multiterminal Task Programming and Operation, GC28-2034, explains how to create and operate a multiterminal-per-task (MTT) program.

IBM Time Sharing System: Quick Guide for Users, GX28-6400, is a compact reference to TSS terminals, commands, and languages.

Publications for Assembler Language Programmers

IBM Time Sharing System: Assembler Programmer's Guide, GC28-2032, provides tutorial and reference material about TSS as viewed by an assembler language programmer. Conversational and nonconversational operations are discussed; the rules and conventions are given that must be observed at source coding time to use TSS efficiently. Assembler examples and typical coding sequences are shown.

IBM Time Sharing System: Assembler Language, GC28-2000, describes the TSS assembler language coding conventions and basic statements. The macro language and procedures for its use are described.

IBM Time Sharing System: Assembler User Macro Instructions, GC28-2004, describes the TSS macros available to the assembler problem programmer. The types and forms of macros supplied with the system are explained; each macro is described in detail.

Publications for FORTRAN Language Programmers

IBM Time Sharing System: Fortran Programmer's Guide, GC28-2025, provides tutorial and reference material about TSS from the standpoint of a FORTRAN programmer. Both conversational and nonconversational operations are discussed; rules and conventions are given that must be observed at source coding time to use TSS efficiently. Compiler examples and typical coding sequences are shown.

IBM Time Sharing System: FORTRAN IV Language, GC28-2048, describes the TSS FORTRAN language. It includes FORTRAN coding conventions, a discussion of the elements of the language, and a detailed explanation of each of the types of FORTRAN statements. Examples are used to clarify programming rules and to illustrate the various ways in which FORTRAN statements can be written.

IBM Time Sharing System: FORTRAN IV Library Subprograms, GC28-2026, describes the subprograms in the IBM-supplied TSS FORTRAN library and their use in either a FORTRAN or an assembler program. The subprograms are divided into three groups: mathematical, I/O, and service.

IBM Time Sharing System: Fortran IV Primer, GC28-2048, introduces FORTRAN programmers to a 13-command subset of the TSS command system enabling users to create and edit data sets and to compile and execute FORTRAN programs.

Publications for PL/I Language Programmers

IBM Time Sharing System: PL/I Programmer's Guide, GC28-2049, provides tutorial and reference material about TSS from the standpoint of a PL/I programmer. Both conversational and nonconversational operations are discussed; rules and conventions are given that must be observed at source coding time to use TSS efficiently. Compiler examples and typical coding sequences are shown.

IBM Time Sharing System: PL/I Language Reference Manual, GC28-2045, describes the TSS PL/I language in two parts. The first part contains discussion and examples that explain the concepts of PL/I, as well as the different features of the language and their interrelationships. The second part provides quick reference to specific information for detailed rules and syntactic descriptions.

PL/I Library Computational Subroutines, GC28-2046, gives details of the computational subroutines available in the TSS PL/I library. It describes the library support for the PL/I built-in functions and the operators used in the evaluation of PL/I expressions in four major categories: bit and character strings, arithmetic, mathematical, and arrays.

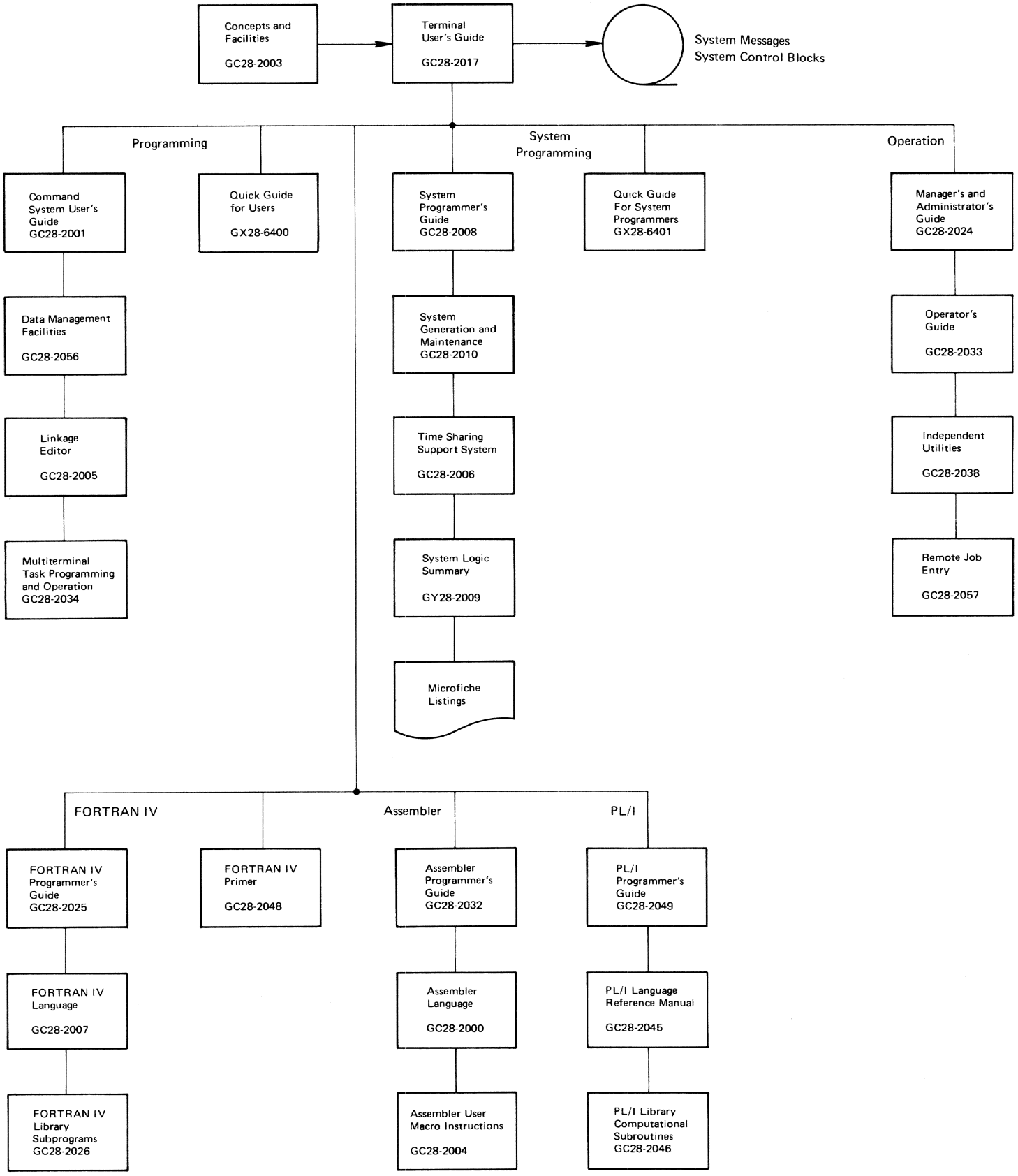

Figure C-1. TSS Publications Guide

Glossary of Terms and Abbreviations

This glossary provides definition of terms in the context of TSS, along with explanation and additional information, which appear in italics. For terms not included in this glossary try the *IBM Data Processing Glossary*, GC20-1699 or the publications it references. This glossary also includes interpretation of all abbreviations used in this publication.

Entries in the glossary are arranged in a collating sequence based on all characters, including blanks (lower case letters raised to upper case before sorting).

***ALL.** A value that may be entered for some parameters which are used for specification of names, when *all* possible names are meant. For example, PERMIT *ALL,*ALL,RO (Permit all one's data sets to all users.)

%COM. The PCS symbol for the FORTRAN blank common block.

%CSECT. The PCS symbol for the unnamed control section. *When the TSS assembler generates code in a control section for which no name has been specified, the control section is unnamed. The dynamic loader is able to load a module with an unnamed control section, but only one such section can be loaded at a time. The symbol* %CSECT *makes it possible to refer to this control section in PCS statements.*

-A-

ABEND. Abnormal end condition. *ABEND is a user command and also a macro instruction. A task is not necessarily ended when an ABEND is issued in a program. Depending on the severity code specified, the program may be terminated, the task may be terminated and a new task created, or the task may be terminated without creation of a replacement task.*

access method. A technique for moving data between main storage and input/output devices. See *virtual access method.*

active list. The list of tasks which are contending for service. The active list is divided into two parts: the dispatchable list, and the eligible list. Contrast with *inactive list.*

address constant. A word of object code that changes as a result of relocating the program in storage.

address space. The complete range of addresses that is available to a program.

addressing capability. The size of storage addresses that a CPU uses to fetch and store data.

addressing mode. A parameter of the LOGON command which specifies the addressing capability to be utilized, when such specification is appropriate.

ANSI. *American National Standards Institute.*

ASCII. American National Standard Code for Information Interchange.

attention interrupt. (1) A signal from a terminal which means that the program controlling the terminal is to respond to an unsolicited request from the user. *In the case of SYSIN terminals, this usually indicates that control is to be given to the command system.* (2) In System/370, one of the types of interrupts that input/output channels can present. *In this case also, the interrupt indicates that a response is to be made to handle the condition which caused the interrupt.*

auxiliary storage. That portion of direct-access storage which is used to support the paging of virtual storage. Contrast with *external storage.*

-B-

batch processing. Execution of commands and programs under the control of statements contained in a data set, without any intervention by a user. See also *noninteractive computing.*

bind. To assign a value to a symbol. *For example, assignment of a value to a variable, association of a storage address with a symbolic address or label in a computer program.* See also *late binding.*

blocked page set. The set of pages which have the greatest probability of being used by a task when it is dispatched. *The blocked page set is brought in before a task is dispatched. The set consists of the PSW page, the ISA page, some of the pages referenced in the last three time slices, and the XTSI and page table pages supporting the referenced pages.*

BSAM. Basic sequential access method.

BULKIO. The name of the task which manages bulk input and output.

-C-

CCW. Channel command word.

combined dictionary. The installation's command system dictionary merged during LOGON with a user-modified copy of the installation's dictionary, used to customize the command system for each user. The combined dictionary has entries for defaults, synonyms, command symbols, PROCDEFs, and BUILTINs.

command procedure. A sequence of commands, defined by use of the PROCDEF command, to be invoked and executed as a single command.

command symbol. A variable that can be defined and given a value within the context of the command system.

command system. The set of programs that receives control when a user logs on. The command system is controlled with a command language. *The command system is used to start all processing of programs for a task. The command system is responsible for establishing the characteristics of the interaction between users and the system.*

connect time. The length of time that a user's terminal is connected to the system.

control block. A storage area which holds information vital to the control of interfaces within and between programs. *In TSS, all system control blocks are described by DSECTs, and all reference to system control blocks is achieved by use of symbols defined in the DSECTs.*

control section. The smallest relocatable unit in a program; that portion of text specified by the programmer to be an entity, all elements of which are to be allocated contiguous storage locations. Abbreviated CSECT.

control section packing. A parameter of the LOGON command which specifies options for the dynamic loader governing loading of control sections into adjoining storage.

control storage. (1) Storage used for basic or elementary machine instructions. (2) In this publication, storage used for level 0 programs.

conversational. Pertaining to the interaction or dialog between a user and a system which takes place through a terminal. Contrast with *nonconversational. This implies the ability to control, interrogate, modify, and observe processing of programs entered through a terminal.*

CPU. Central processing unit.

CPU time. The amount of time devoted by a CPU to the execution of instructions. *The value of CPU time that is accounted for, and made available to, each task (or user) is the time spent by the CPU for execution of programs in and above level 2, to which is added an installation-specified charge per SVC.*

CSECT. Control section.

current PSW. In System/370, the PSW actively controlling the state of a CPU.

current VPSW. In TSS, the VPSW actively controlling the state of a task.

-D-

D. An ASCII record format in which records in the data set are variable length.

data base. A collection of data fundamental to a system or an enterprise.

data communication. (1) The transmission and reception of data, often including operations such as coding, decoding, and validation. (2) Exchange of information between and among computing systems, terminals, and people.

data control block. The control block shared between access methods and user programs to control storage and retrieval of data.

data management. System programs that organize, catalog, locate, store, retrieve, and maintain data.

data set. The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

data set label. A collection of information that describes the attributes of a data set and is normally stored on the same volume as the data set.

DB/DC. Data base/data communication.

DCB. Data control block.

DCL. In TAM II, a device control library.

DCM. In TAM II, a device control module.

DDNAME (Data definition name). The name used to link a data control block in a program to a data definition statement (DDEF command), thus providing the link between programs and data sets.

DECB. Data event control block.

default value. The choice among exclusive alternatives made by the system when no explicit choice is specified by the user.

delta data set. A data set containing modules that are to replace system modules. STARTUP can be given a list of delta data sets to be used when it link-loads the system. *Because delta data sets are separate from the data sets in which the system resides, management of change associated with replacement of system modules is simplified.*

dispatchable list. A division of the active list containing tasks in main storage which are receiving CPU service. The supervisor performs multiprogramming when more than one task is on the dispatchable list. Contrast with *eligible list.*

DSCB (Data set control block). (1) For BSAM, a data set label for a data set in direct-access storage. (2) For VAM, a control block in external storage which describes the attributes of a data set and includes the necessary information to locate all information in the data set.

DSECT (Dummy control section). A description of a control block that the assembler can use to generate code that refers to items in the control block. *By properly defining a DSECT and using its symbols to refer to the area that the DSECT describes, it is possible to write a program that is independent of the layout of the control block. Thus, when it is necessary to expand or rearrange the control block, any code that refers to fields in the control block need not be changed.*

dynamic address translation. A function of a suitably equipped CPU whereby storage addresses, generated during the execution of a program, are converted from virtual to real. If there is no corresponding real (main storage) address, the CPU is interrupted by a translation exception. The control program responds by bringing the addressed virtual storage into main storage and updating the tables which the CPU uses to perform the translation.

dynamic allocation. Assignment of system resources to a program during execution of the program.

dynamic loader. A program which can bind executable programs together and load them into storage, resolving references to other programs as it proceeds. *The dynamic loader is the means by which programs (including data in program format) are mapped to virtual storage. The dynamic loader eliminates the need to link-edit the output of language processors.*

-E-

EBCDIC (Extended binary coded decimal interchange code). A set of 256 characters, each represented by eight bits.

editing. The process of modification of the form or format of data, for example, alteration of source programs, memos, graphic images, and input data for programs. *Typically, users of interactive systems spend the majority of the time they are at terminals performing tasks which can be characterized as editing.*

eligible list. A division of the active list containing tasks which are waiting for main storage. Contrast with *dispatchable list.*

entry name. A name within a control section that defines a location which can be referenced from outside the control section.

entry point. A location in a program, defined so as to make it an entry name, to which control can be passed by another program.

EODAD (End of data set address). An address in the DCB indicating where control is to be transferred upon reaching the end of an input data set being processed.

EREP (Environmental recording, editing, and printing). The program that makes the data contained on the system recorder file available for further analysis.

EXLST (Exit list address). An address in the DCB providing for transfer to user-supplied routines during OPEN processing. (BSAM only.)

explicit loading. Loading of programs in which a call to the dynamic loader is deferred until a specific request is made. Contrast with *implicit loading.*

external name. A name that can be referred to by any control section or separately assembled or compiled module; that is, a control section name or an entry name in another module.

external reference. (1) A reference to a symbol that is defined as an external name in another module. (2) An external symbol that is defined in another module; that which is defined in the assembler language by an EXTRN statement or by a V-type address constant, and is resolved during linkage editing. *External symbols are also resolved by the dynamic loader.* Abbreviated EXTRN.

external storage. The portion of secondary storage that is available to users for permanent data storage (that is, the part of secondary storage not used for auxiliary storage). Contrast with *auxiliary storage.*

-F-

F (Fixed-length). A record format in which all records of a data set or member are the same length.

FCL. In TAM II, a format control library.

FCM. In TAM II, a format control module.

FORTRAN (FORmula TRANslating system). A language primarily used to express computer programs by arithmetic formulas.

-G-

GDG. Generation data group.

generation data group. A collection of data sets that are kept in positional order; each data set is called a generation data set. Abbreviated GDG. *The data sets in a GDG are usually in chronological order. When they are placed in the group, or their position in the group is altered, the order is established.*

GQE (General queue entry). The unit of work for the supervisor.

-H-

hook. An inclusion in code, which normally has no effect on execution of the program other than the time it takes to execute the few instructions of the hook. The purpose of a hook is to provide an optional exit to another program for some purpose. A hook is activated by action external to the program. *Hooks are used for performance analysis (SIPE) and for replacement of level 2 programs with test versions. (Such replacement does not affect the remaining users of the system, who continue to use normal versions of the programs.)*

-I-

I/O. Input/output.

IAM (Independent access method). A supervisor and loader for System/370 that provides services for programs which run outside the TSS environment, usually for maintenance purposes. When IAM is in control, only the programs it loads and executes can use the computing system.

ICB. Interrupt control block.

implicit loading. Program loading that is performed immediately as external references are encountered. Contrast with *explicit loading.*

inactive list. The list of tasks which are not making any demands on the CPU for service. Inactive tasks are waiting for an external event to occur. Contrast with *active list.*

indexed sequential. Pertaining to the organization of a data set characterized by provision of the ability to read, write, insert, delete, and update records, using a key contained in the record. It is possible to have direct access to records in a data set with indexed sequential organization. *A member of a partitioned data set may have indexed sequential organization, but it is not necessary that all members of the data set be indexed sequential.*

interactive computing. Pertaining to an application in which each entry elicits a response, as in an inquiry system or an airline reservation system. Interactive computing may also be conversational, implying a continuous dialog between the user and the system. Contrast with *noninteractive computing.*

IPL (Initial program load). A function, normally activated by pushing a key on the computer console, that causes a special record to be read from an I/O device. This record contains a program which begins a succession of initialization procedures that ultimately establishes an operating environment.

IPL bootstrap. The content of the special record which is read into main storage as the result of pushing the IPL key on a computer console.

ISA (Interrupt storage area). In the terminology of this publication, the functional analog in *level 2* of the prefixed storage area (PSA) in *level 1.* The ISA contains the status switching control information for a task (virtual computer), for example, the old and new VPSWs.

ISD (Internal symbol dictionary). A dictionary of locations and attributes of locations defined in source programs, preserved in the object module by the language processor for use at execution time.

IVM (Initial virtual memory). Level 2 address space as created by STARTUP and saved for use each time a new task logs on.

-J-

JFCB (Job file control block). The control block created as a result of the definition of a data set (by the DDEF command). When a DCB is opened, data management attempts to match the DDNAME in the DCB with a name from the list of JFCBs. This establishes the association between a program and a data set.

job library. A partitioned data set containing object modules, which, by virtue of its definition as such, is a potential source of object modules for the dynamic loader. The output of all language processors is placed in the job library which is at the top of the list of defined job libraries. Abbreviated JOBLIB.

JOBLIB. Job library.

-L-

language processor. An assembler, compiler, or other program that accepts statements in one language and produces functionally equivalent statements in another language, for example, machine language. *In TSS, the linkage editor is classified as a language processor. This is more a result of the way the linkage editor is invoked than any characteristic of its execution and is also due to the fact that the output of the linkage editor is an object module.*

late binding. The deferral until the last possible moment of any association that cannot be dissolved readily. Almost every design feature of TSS has the goal of keeping the use of objects that users process free of constraints that might interfere with the iterative nature of the programming process.

level 0. In System/370, the implementation of the system architecture.

level 1. In TSS, the implementation of an extended machine architecture in which level 2 programs operate.

level 2. In TSS, the environment in which privileged system programs operate, protected from damage due to user program execution. The majority of operating system function, exclusive of management of real resources, is contained in level 2 programs.

level 3. In TSS, the environment in which language processors and user programs operate.

level 4. In TSS, the environment in which users interact with the system. Implementation in level 4 is the responsibility of customers, except for the language processors, editors, and command system.

line data set. A virtual indexed sequential data set that is organized by line number. *In TSS, source programs are in line data set format.*

line number. The key in records of a line data set.

link-loading. A process by which object modules are bound and loaded into storage without producing a new object module.

linkage editor. In TSS, a program that transforms one or more object modules into a single object module. The linkage editor resolves external references within the input object modules, and, optionally, combines separate control sections into a single control section, and replaces, deletes, adds, and renames control sections, as specified.

logical I/O. The process of performing I/O at the record level, free from concern with actual devices. Contrast with *physical I/O.*

logoff. The procedure by which a task is ended normally. *A user "logs off."*

logon. The procedure by which a task is initiated and by which a USERID is associated with a task. *A user " logs on."*

LPC. Language processor controller.

-M-

map. To establish a correspondence between the elements of one set and the elements of another set. *The principal TSS access method, VAM, accomplishes its function by using the dynamic address translation feature of the CPU to* map *data sets in external storage to the address space in which programs operate.*

MC (Monitor Call). In System/370, an instruction which performs the function of a hook.

member. A partition of a partitioned data set.

MSAM. Multiple sequential access method.

MTT. Multiple terminals per task.

-N-

new PSW. In System/370, a location containing the value that will be loaded into the current PSW when the CPU is interrupted. There is a new PSW for each interruption type.

new VPSW. In TSS, a location containing the value that will be used for a task's current VPSW when the task is interrupted. There is a new VPSW for each interruption type.

nonconversational. Pertaining to a program or a system that does not involve a dialog with a terminal user. Contrast with *conversational.*

noninteractive computing. Data processing which does not involve human interaction. Contrast with *interactive computing. In this publication, the term* noninteractive computing *is used to expand upon the opposite of conversational and include processing which does not involve human interaction but does involve rapid response to outside stimulus.*

nonprivileged. Pertaining to the state in which level 3 programs execute. Contrast with *privileged.*

nonprivileged state. One of the three states of execution defined by TSS architecture; the conditions which govern the execution of level 3 programs. Contrast with *privileged state.*

NOP. No operation (no op).

-O-

object data converter. The name of a system service program which converts the output of OS/VS language processors into TSS object module format. Abbreviated ODC.

object module. The output of a single execution of a language processor (including the linkage editor), or the object data converter, which constitutes input to the dynamic loader or the linkage editor; an object module consists of one or more control sections in relocatable (but not executable) form, and an associated program module dictionary.

ODC. Object data converter.

offline storage. Storage of data which has been put into a hierarchical store one or more levels removed from the user, possibly compacted, usually as a result of migration from public storage.

old PSW. In System/370, a location containing the value of the current PSW as saved when the CPU was interrupted. There is an old PSW for each interruption type.

old VPSW. In TSS, a location containing the value of the task's current VPSW as saved when the task was interrupted. There is an old VPSW for each interruption type.

online computing. Unattended processing of data from ongoing processes, for example, logging of data from continuous experiments and use of such data to control the experiments.

online storage. Public storage, and less frequently, private storage volumes that happen to be mounted.

OS/MVT. Operating System/multiprogramming with a variable number of tasks.

OS/VS. Operating System/Virtual Storage

-P-

page. (1) In the two-level virtual storage of System/370, a contiguous portion of virtual storage described by a single page table entry. (2) In TSS, the basic unit of virtual storage, 4096 bytes. (3) To read or write portions of virtual storage into main storage or onto auxiliary storage.

page table. In the two-level virtual storage of System/370, a table containing the real addresses of virtual storage pages, for one segment of virtual storage. The page table also indicates whether a page is in real storage.

page-addressable. The ability to address data by page number, without regard to the characteristics of the facility in which the data is stored. *This is a TSS design point, for example, data on external storage in VAM data sets, and data in virtual storage.*

paging. The process of transferring pages of virtual storage between main storage and auxiliary storage.

paging supervisor. Not actually a separate program or set of programs, but a function of the supervisor responsible for managing virtual storage.

parallel reenterable. Pertaining to a program, an attribute indicating that it is structured so that more than one CPU can execute a single copy of the program simultaneously. This implies that changeable portions of the program (work areas) are separately obtained for each CPU executing the code. *In level 1 programs, this is accomplished by common techniques which all programs use to get unique working storage. For level 2 and level 3 programs, STARTUP and the dynamic loader treat properly defined object modules in a manner that guarantees parallel reenterability.*

parcel. The unit of data in a symbolic library, for example, a macro.

partitioned data set. A VAM data set having partitions, called members. *Each member can have all of the characteristics of VAM data sets, except that the member can not be partitioned. Partitioned data sets are convenient for referring to many data objects as a collection. Members of a partitioned data set may expand and contract freely, unused space being automatically returned to public storage for general use.*

partitioned organization directory. The information by use of which the contents of a partitioned data set are managed. The directory is part of the data set.

PAT (Page availability table). The basic control block used to control space on VAM volumes.

PCI. Program controlled interrupt.

PCS. Program control system.

PER. Program event recording.

physical I/O. The process of performing I/O at the device level, specifically, the use of channel programs. Contrast with *logical I/O.*

PL/I. A high-level programming language, designed for use in a wide range of commercial and scientific computer applications.

PPLI. Program product language interface.

PRELUDE. The common name for the program loaded by the IPL bootstrap which does some configuration analysis and then gives control to the IAM loader, which asks the system operator for the name of a utility to be run, for example, STARTUP, which initializes TSS.

private device. An I/O device which is dedicated to a specific user and allocated under control of a system service function, called device management. Private devices are acquired by means of the DDEF command. The amount of time that a private device is dedicated to a user is recorded by the system and made available to installation-provided accounting routines.

private segment. A segment of virtual storage that is not shared with any other address space (task). Contrast with *shared segment.*

private volume. A volume, such as a reel of tape or a disk pack, owned by, or assigned to, an individual user.

privileged. Pertaining to the state in which level 2 programs execute. Contrast with *nonprivileged.*

privileged state. One of the three states of execution defined by TSS architecture; the conditions which govern the execution of level 2 programs. Contrast with *nonprivileged state.*

problem state. (1) In System/370, an execution state defined for programs that can not be allowed to take over control of the system. (2) In TSS, the real hardware state in which level 2 and level 3 programs execute. Contrast with *supervisor state.*

program library list. The list of currently defined job libraries.

program module dictionary. The collection of control and descriptive information, concerning an object module, required by programs that must process that module.

program status word. In System/370, a double-word maintained by the CPU, defining the execution state of the machine. In the event that the CPU is interrupted, the contents of the current program status word are stored in one of several fixed locations in main storage, according to the type of interrupt. These locations are called the old PSWs. The CPU loads a corresponding PSW from one of several fixed locations in main storage, according to the type of interrupt. These locations are called the new PSWs. See *old PSW* and *new PSW.* Abbreviated PSW.

protection key. (1) In System/370, a portion of the current PSW which must match the storage keys of all locations in real storage that are to be successfully used. (2) In TSS, a portion of the current VPSW which must match the storage keys of all locations in virtual storage that are to be successfully used.

PSA (Prefixed storage area). The area of main storage which contains fixed locations related to machine status, for example, the old and new PSWs.

PSECT (Private control section). A control section with a special attribute having meaning to the dynamic loader as regards dynamic sharing of programs in shared virtual storage. *When the dynamic loader loads a module from a shared job library, any control sections in that module having the public attribute and already loaded in another task will be shared between address spaces. PSECTs will not be shared; a private copy is obtained for each address space (task). PSECTs usually contain information that is modified by program execution.*

PSW. Program status word.

PTF. Program temporary fix.

public attribute. That attribute of a control section which indicates to the dynamic loader that it can be shared by other address spaces (tasks).

public segment. A segment of virtual storage that can be shared with any other address space (task). See *shared segment.*

public volume. A direct-access volume that is part of a set of volumes which comprise the public storage of the system. Public storage contains VAM data sets and all of it is available for allocation to all users, subject to an installation-specified upper limit for each user.

-Q-

QSAM. Queued sequential access method.

-R-

R-value. A value (address) associated with an external name (which may be a module name), giving the location of the PSECT corresponding to the external name. If there is no corresponding PSECT, the R-value is equal to the V-value of the external name.

RC (Real core). Level 1 storage.

real storage. Addressable space in main storage, from which instructions and data are fetched. Contrast with *virtual storage.*

record. (1) A unit of data in a data set, read or written with one request. (2) A unit of data in a data storage device whose contents are, excepting use of VAM, not addressable by the CPU.

region. In a region data set, those records which have keys that are the same except for the rightmost seven characters.

region data set. An indexed sequential data set divided into portions, each of which is in the format of a line data set, except that the keys in the records consist of two parts: the region name and the line number.

RESTBL. Relative external storage correspondence table.

retrieval address. An address, returned to a program using VAM, each time a record is written into a VSAM or VISAM data set (or member of a VPAM data set). The retrieval address may later be used to gain direct access to the record, even with VSAM data sets (or members), which are strictly organized sequentially. Use of retrieval addresses facilitates construction of data bases with multiple keys and various means of access, including computed retrieval addresses in the case of fixed-length sequential data sets.

RJE. Remote job entry.

RM (Real memory). Level 1 storage.

RMS. Recovery management system.

RO. Read-only.

RSS. Resident support system.

RTAM. Real terminal access method.

RW. Read/write.

-S-

save area. An area used, when one program calls another program, to store the status of the CPU, such as register content, program mask, and condition code, so that the called program may use the CPU as it sees fit, and subsequently restore the conditions that were in effect at the time of the call.

script. A series of commands and data in the form of a data set which can be invoked with the EXECUTE command. Scripts execute as nonconversational jobs in a task separate from the task that requested execution.

secondary storage. (1) Storage to which the CPU does not have direct access. (2) In TSS, secondary storage on direct-access devices is called auxiliary storage or external storage, depending on its use. See *auxiliary storage* and *external storage.*

segment. In the two-level virtual storage of System/370, a contiguous portion of virtual storage described by a single page table. *TSS uses 64-kilobyte segments.*

segment table. In the two-level virtual storage of System/370, a table containing the addresses of page tables, one address for each segment of virtual storage. The segment table also indicates whether a page table is in real storage.

sequential. Pertaining to the organization of a data set characterized by access to records one after another or one before another, in the case of reading the data set from back to front. *Records of a VSAM data set can be accessed in any order using retrieval addresses.*

serially reusable. Pertaining to a program, an attribute indicating that it is structured to allow repetitive use without refreshing the code. In other words, the program has only one state; execution does not change that state. *In TSS, use of a program which modifies its code is not recommended, because programs typically remain loaded after use and are easily available for subsequent use. Not infrequently, programs are interrupted and reexecuted and should therefore be coded to be serially reusable.*

service program. A system program that assists in execution of user programs, without directly controlling the system or producing results.

shared segment. In the two-level virtual storage of System/370, a contiguous portion of virtual storage described by a single page table and indicated as being sharable by another address space. Sharing is accomplished when entries in two or more segment tables point to the same page table. See *public segment.*

SIPE. System internal performance evaluator.

starter system. An operating system, supplied with an assumption about the actual configuration of the computing system on which it will be run. The assumption includes numerous addresses for various device types, enough of which can be expected to match the hardware configuration on which the starter system is to run, so that an installation can use the starter system to generate a system tailored to its own configuration.

STARTUP. The name of the utility that performs configuration analysis and initializes TSS.

STCK (Store Clock). In System/370, an instruction that causes the time-of-day clock to be stored.

storage key. In System/370, an indicator associated with one or more storage blocks, that controls storing and fetching of data, requiring a matching protection key in the PSW. In some models, the storage key indicates all references and changes to the storage block associated with the key. *TSS uses the reference and change bits to manage storage.*

subsystem. (1) A program which provides service to a set of users and utilizes the services of the system but is not considered part of the system. (2) A secondary or subordinate system, usually capable of operating independently of, or asynchronously with, a controlling system.

supervisor. (1) The part of a control program that coordinates the use of resources and maintains the flow of CPU operations. (2) In TSS, the programs that operate in level 1.

supervisor state. (1) In System/370, an execution state defined for programs that have control of the system. (2) In TSS, one of the three states of execution defined by TSS architecture; the conditions which govern the execution of level 1 programs. Contrast with *problem state.*

supporting program. A system program that performs a maintenance function, but one that does not directly control the system or produce results.

SVC (Supervisor Call). In System/370, an instruction that causes the SVC new PSW to be loaded. By this means, requests for service can be processed by programs operating in a state different from that of the program that issued the SVC.

symbolic library. A library containing portions of source programs in the form of symbolic statements in the language of the assembler, for example, macros, and DSECTs.

SYNAD (Synchronous error exit address). An address in the DCB indicating where control is to be transferred upon occurrence of an unrecoverable error condition.

SYSGEN (System generation). The process of adjusting the content of tables in the system to a specific hardware configuration.

SYSIN. A system input stream; also, the DDNAME used for the task's system input data set. *The input stream is the keyboard of the terminal being used or a data set.*

SYSMANGR. One of three prejoined USERIDs supplied with TSS. SYSMANGR joins other users to the system and performs special functions related to managing the installation.

SYSOPER0. One of three prejoined USERIDs supplied with TSS. SYSOPER0 is the system operator.

SYSOUT. A system output stream; also, the DDNAME used for the task's system output data set. *The output stream is either the printer or display screen of the terminal being used or a data set.*

-T-

TAM II. TSS telecommunications access method.

task. (1) All work performed by TSS under the direction of a stream of commands from a system input unit that is associated with a user, between the LOGON and LOGOFF commands. Some nonconversational tasks are initiated for a user by the system, and are terminated at the completion of the operation, for example, the task created to carry out WT command function. (2) The address space and the associated control blocks established to support a TSS user.

task management. Those functions of the control program that regulate the use of system resources other than I/O devices.

TASKID. Task identification number.

TDT (Task definition table). The list of JFCBs related to a task.

text processing. Editing and formatting of information intended for publication, for example, the computer-assisted processing which produced the images from which this publication was printed.

time slice. The amount of time that a task is allowed on the dispatchable list, measured in CPU time. *Once the interval has expired, CPU time is allocated to another task; thus a task cannot monopolize CPU time beyond a specified limit. Some installations may wish to allow a task to monopolize the CPU; this is possible, using the facilities of the schedule table and scheduler.*

TSE. Time slice end.

TSI (Task status index). The basic control block for each task to which all other control blocks related to the task are connected.

TSS. (1) IBM Time Sharing System. (2) One of three prejoined USERIDs supplied with TSS. USERID TSS owns the system data sets with the exception of a few owned by SYSOPER0.

TSS application. In the context of TSS subsystems, time sharing. The TSS application involves supplying computer service to a TSS USERID. Coexisting with the TSS application can be user-developed subsystems which execute as a TSS USERID but supply service to users who connect to the system in order to gain access to the subsystem.

TSS/370. IBM System/370 Time Sharing System, Program Number 370G-CL-627.

TSSRES. The volume identification of the TSS starter system volume.

TSSS. Time sharing support system.

-U-

U (Undefined-format). A record format in which the lengths of records in the data set or member are not known before reading or writing.

user catalog. A catalog unique to each TSS user. The user catalog contains the names of all data sets owned by the user and the names used by the user to refer to data sets owned by other users. The user catalog contains specifications of permission for other users to access the user's data sets. It also contains many characteristics of the data sets and points directly to VAM data sets.

user program. Programs written by users that execute in level 3.

USERID (User identification). A three- to eight-character name, used to identify users and their data sets.

USS. Utility support system.

-V-

V (Variable-length). A record format in which records in the data set or member can have different lengths.

V-value. A value (address) associated with an external name (which may be a module name), where the text corresponding to the external name is loaded.

VAM. Virtual access method.

VCCW. Virtual channel command word.

virtual access method. The virtual access method is the principal means of data transfer between virtual storage and external storage. VAM is characterized by the use of the paging hardware and software to perform data set I/O.

virtual program status word. In TSS, a doubleword maintained by the supervisor in level 1 storage, defining the execution state of the level 2 virtual computer (task). In the event that the task is interrupted, the contents of the current virtual program status word are stored in one of several fixed locations in virtual storage, according to the type of interrupt. These locations are called the old VPSWs. The supervisor causes a corresponding VPSW to be loaded from one of several fixed locations in virtual storage, according to the type of interrupt. These locations are called the new VPSWs. See *old VPSW* and *new VPSW*. Abbreviated VPSW.

virtual storage. Addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing capability of the computing system rather than by the actual number of real storage locations. The total size of all virtual storage created by a computing system is limited by the amount of auxiliary storage available.

VISAM. Virtual indexed sequential access method.

VM (Virtual memory). Level 2 storage.

volume. (1) That portion of a single unit of storage which is accessible to a single read/write mechanism, for example, a drum, a disk pack, or part of a disk storage module. (2) A recording medium that is mounted and demounted as a unit, for example, a reel of magnetic tape, a disk pack, a data cell.

volume label. A machine-readable record on a volume, which the system can use to control access to the volume.

VPAM. Virtual partitioned access method.

VPSW. Virtual program status word.

VSAM. Virtual sequential access method.

VSS. Virtual support system.

VTSS. Virtual terminal support system.

-W-

working set. The set of virtual storage pages that are used within some period of time. *A program with high locality of reference will have a small working set and thus does not present as great a demand upon system resources. Working set has nothing to do with how much virtual storage is potentially addressable.*

-X-

XTSI. Extended task status index.