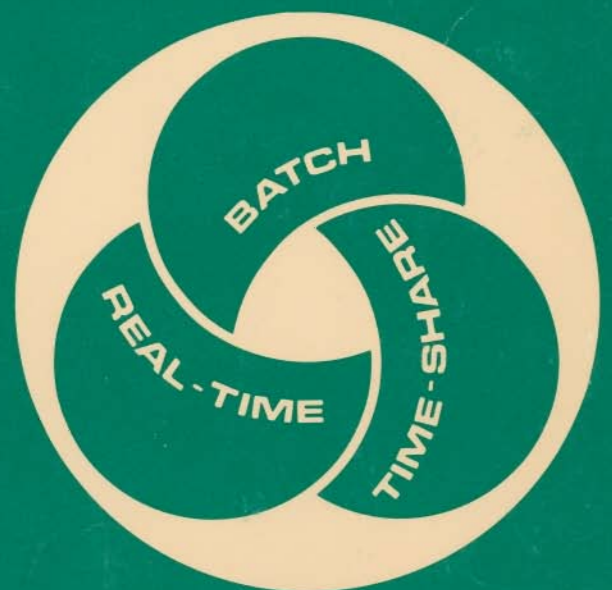


HP 3000

**MULTIPROGRAMMING
EXECUTIVE SYSTEM
MANAGER/
SUPERVISOR
CAPABILITIES**



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP 3000

MULTIPROGRAMMING EXECUTIVE SYSTEM MANAGER/ SUPERVISOR CAPABILITIES



List of Effective Pages

Pages	Effective Date
Title	Nov. 1972
Copyright.	Nov. 1972
iii to ix	Nov. 1972
1-1 to 1-10	Nov. 1972
2-1 to 2-28	Nov. 1972
3-1 to 3-24	Nov. 1972
4-1 to 4-28	Nov. 1972
5-1 to 5-15	Nov. 1972
6-1 to 6-18	Nov. 1972
7-1 to 7-24	Nov. 1972
A-1 to A-5	Nov. 1972
B-1 to B-4	Nov. 1972
C-1 to C-3	Nov. 1972
D-1 to D-3	Nov. 1972
E-1 to E-2	Nov. 1972
F-1 to F-2	Nov. 1972
G-1 to G-2	Nov. 1972
H-1 to H-11	Nov. 1972

Printing History

Part No.	Date	Update Package	Date
03000-90038A	Nov. 1972		

PREFACE

This manual is intended for HP 3000 Computer users assigned the System Manager and/or System Supervisor Capabilities under the Multiprogramming Executive Operating System (MPE/3000). The manual introduces the main features of the system and provides a general overview of MPE/3000, explaining all significant points that a user with System Manager or Supervisor Capability might need to know. It presents information on how to start-up and shut-down the system, and how to log-on and enter commands interactively from a terminal (in time-sharing mode) or from cards or other media (in batch-job mode).

For the user with the *System Manager Capability*, the manual tells how to create, change and delete the accounts under which all users access the system. The manual also shows how to assign the Account Manager Capability that allows a user organizational control over an individual account.

For the user with the *System Supervisor Capability*, the manual tells how to generate and modify MPE/3000, display job status information, manage the MPE/3000 scheduling queue, run programs, permanently allocate code in virtual memory, and transmit messages to other users.

This manual is a reference book rather than a tutorial text for new programmers. The reader should understand the basic operating principles of the HP 3000 Computers. He should also examine the following manual for an overview of the interrelationships between the main hardware and software features offered:

Title	Part No.
<i>HP 3000 Software General Information</i>	<i>03000-90001</i>

The user may also want to read the following manuals for further information on MPE/3000, the HP 3000 Systems Programming Language (SPL/3000), and the functions available through the MPE/3000 Operator's Console.

Title	Part No.
<i>HP 3000 Systems Programming Language</i>	<i>03000-90002</i>
<i>HP 3000 Systems Programming Language Textbook</i>	<i>03000-90003</i>
<i>HP 3000 Multiprogramming Executive Operating System</i>	<i>03000-90005</i>
<i>HP 3000 Multiprogramming Executive Console Operator's Guide</i>	<i>03000-90006</i>

CONTENTS

PREFACE	iii
SECTION I INTRODUCTION TO MPE/3000	1-1
FEATURES	1-1
Multiprogramming	1-1
General-Purpose Versatility	1-1
Choice of Programming Languages	1-2
Ease of Use	1-2
Input/Output Conveniences	1-3
Processing Efficiency	1-3
System-to-System Compatibility	1-5
Program Security	1-5
SYSTEM MANAGER AND SYSTEM SUPERVISOR CAPABILITIES	1-6
SOFTWARE	1-7
HARDWARE	1-7



SECTION II	HOW MPE/3000 OPERATES	2-1
	MPE/3000 RESIDENCE	2-2
	PROCESSES	2-2
	PROCEDURES	2-5
	SYSTEM LIBRARY	2-6
	USER INTERFACE	2-6
	Commands	2-6
	Intrinsic Calls	2-8
	INPUT/OUTPUT	2-8
	File Management System	2-9
	Scheduling	2-9
	COMPILATION	2-11
	PROGRAM PREPARATION	2-13
	ALLOCATION/EXECUTION	2-13
	PCB/CODE SEGMENT/STACK INTERACTION	2-14
	MEMORY MANAGEMENT	2-19
	Main-Memory Linkages	2-19
	Main-Memory Use	2-20
	USER JOB PROCESSING	2-20
	Batch Programs (Jobs)	2-20
	Time-Sharing Programs (Sessions)	2-23
	ACCOUNT/GROUP/USER ORGANIZATION	2-23
	CAPABILITY SETS	2-25
	User Attributes	2-25
	File-Access Attributes	2-26
	Capability-Class Attributes	2-27
	Local Attributes	2-28
	Program Capability Sets	2-28
SECTION III	COMMUNICATING WITH MPE/3000	3-1
	COMMANDS	3-2
	Positional Parameters	3-3
	Keyword Parameters	3-3
	Continuation Characters	3-4
	Command Example Format	3-4
	Command Errors	3-5

INTRINSIC CALLS	3-6
Intrinsic Example Format	3-8
Intrinsic Call Errors	3-9
BATCH JOBS	3-10
Initiating Batch Jobs	3-10
Terminating Batch Jobs	3-13
Typical Job Structures	3-13
TIME-SHARING SESSIONS	3-16
Initiating Sessions	3-16
Interrupting Program Execution Within Sessions	3-19
Terminating Sessions	3-20
Typical Session Structure	3-20
READING DATA FROM OUTSIDE STANDARD INPUT STREAM	3-21
PREMATURE JOB OR SESSION TERMINATION	3-23
 SECTION IV SYSTEM GENERATION AND BACK-UP	 4-1
MODIFYING MPE/3000	4-3
Changing Main Memory Size	4-3
Changing Input/Output Device Configuration	4-3
Changing System Tables and Queue Sizes	4-6
Changing Miscellaneous Values Relating to Log-On Time, RIN's, and Jobs Allowed in System and Execution	4-7
Changing Disc Allocation	4-8
Changing Scheduling Queue	4-8
Changing Segment Limits	4-8
Changing Code Segments and Program Files in the System Library	4-9
USING THE :SYSDUMP COMMAND	4-10
CONFIGURATOR/USER DIALOGUE	4-12
 SECTION V SYSTEM START-UP, MODIFICATION, AND SHUT-DOWN	 5-1
START-UP AND MODIFICATION	5-1
SHUT-DOWN	5-15

SECTION VI ORGANIZATIONAL MANAGEMENT	6-1
TYPICAL ACCOUNTS	6-1
Large Multi-Division Company	6-1
In-House Service Bureau	6-3
Commercial Service Bureau	6-3
Scientific Research Center	6-3
DEFINING ACCOUNTS	6-3
Creating Accounts	6-4
Altering Accounts	6-7
Listing Account Attributes	6-8
Deleting Accounts	6-9
DEFINING GROUPS AND USERS	6-10
Creating Groups	6-10
Altering Groups	6-11
Listing Group Attributes	6-12
Deleting Groups	6-13
Defining Users	6-14
Altering User Attributes	6-15
Listing User Attributes	6-16
Deleting Users	6-17
COMMAND SUMMARY	6-18
 SECTION VII OPERATIONAL MANAGEMENT	 7-1
MANAGING THE MASTER QUEUE	7-1
Closing a Subqueue	7-5
Re-opening a Subqueue	7-5
Changing a Circular Subqueue Quantum	7-6
Rescheduling Jobs/Sessions	7-7
Displaying Subqueue Information	7-8
EXECUTING PROGRAMS	7-9
CREATING PROCESSES	7-12
RESCHEDULING PROCESSES	7-16
PERMANENTLY ALLOCATING PROGRAMS AND PROCEDURES	7-18
Allocation	7-19
De-Allocation	7-20
TRANSMITTING MESSAGES	7-21

APPENDIX A	ASCII CHARACTER SET	A-1
APPENDIX B	SUMMARY OF COMMANDS	B-1
APPENDIX C	SUMMARY OF INTRINSIC CALLS	C-1
APPENDIX D	DRIVER NAMES, TYPES, SUB-TYPES, AND SIZES	D-1
APPENDIX E	DEFAULT PARAMETER SETTINGS FOR CONFIGURATION	E-1
APPENDIX F	MPE/3000 TABLE NAMES AND SIZES	F-1
APPENDIX G	MPE/3000 SEGMENT NAMES AND SIZES	G-1
APPENDIX H	BACK-UP OF DISC FILES	H-1

FIGURES

Figure 1-1.	Small Batch System	1-8
Figure 1-2.	Small Time-Sharing System	1-9
Figure 1-3.	Combined Batch and Time-Sharing System	1-9
Figure 1-4.	Large Processing System	1-10
Figure 2-1.	Code Sharing and Data Privacy	2-3
Figure 2-2.	Process Organization	2-4
Figure 2-3.	User/Software/Hardware Interface	2-7
Figure 2-4.	Scheduling Queues	2-10
Figure 2-5.	Program Management	2-12
Figure 2-6.	Code Segment and Associated Registers	2-14
Figure 2-7.	Data (Stack) Segment and Associated Registers	2-16
Figure 2-8.	Account/User/Group Organization	2-18
Figure 2-9.	Batch Job History	2-21
Figure 2-10.	Account/User/Group Organization	2-24
Figure 5-1.	Switch Register	5-3
Figure 5-2.	Load Map	5-14
Figure 6-1.	Account/Group/User Structure in a Large Corporation	6-2
Figure 7-1.	MPE/3000 Master Queue Structure	7-2

SECTION I

Introduction to MPE/3000

The Multiprogramming Executive Operating System (MPE/3000) is a general-purpose, disc-based software system that supervises the processing of user programs submitted to HP 3000 Computers. MPE/3000 relieves the user from many program control, input/output, and other housekeeping responsibilities by monitoring and controlling the input, compilation, run preparation, loading, execution and output of user programs. MPE/3000 also controls the order in which programs are executed, and allocates the hardware and software resources they require.

FEATURES

MPE/3000 offers the user many important features; some of these are found elsewhere only in medium to large-scale computers.

Multiprogramming

Through multiprogramming (interleaved processing), MPE/3000 allows numerous users to execute many different programs concurrently. The number of programs that can be processed concurrently depends on such factors as the hardware configuration, program operating modes (batch or time-sharing), and applications involved. Each programmer, however, uses the computer as if it were his own private machine — in other words, he need not depend on, nor even be aware of, others using the machine.

General-Purpose Versatility

MPE/3000 allows users to run batch and time-sharing programs concurrently.

BATCH PROCESSING. Batch processing lets programmers submit to the computer, as a single unit, commands that request various MPE/3000 operations such as program compilation and execution, file manipulation, or utility functions. Such a unit is called a *job*. Jobs contain all instructions to MPE/3000 and references to programs and data required for their execution; once a job is running, no further information is needed from the programmer. (Frequently, programs and data are submitted as part of the job.)

Jobs are input through on-site (local) devices such as card readers or tape units, or through local or remote terminals operating non-interactively. In fact, several jobs can be submitted from many devices concurrently. MPE/3000 schedules each job according to its priority. When a job enters execution, the commands within it are sequentially executed on a multiprogramming basis. MPE/3000 generates the job output on a local device such as a line printer, tape unit, or card punch, or on a local or remote terminal. When one job is temporarily suspended, perhaps to await input of data, another immediately enters execution. Thus, when many jobs are active in the system, uninterrupted processing and high throughput can be maintained.

TIME-SHARING PROCESSING. In time-sharing, the programmer interacts conversationally with the computer, receiving immediate responses to his input. Many users at different remote or local terminals can program on-line in this fashion, simultaneously sharing the central processor through time-slicing (where each user receives an equal share of time, in a round-robin fashion). This type of interaction, called a *session*, can be used for program development, information retrieval, computer-assisted education, and many more applications where the user at a remote terminal must access the system directly.

BATCH VS. TIME-SHARING PROCESSING. The basic difference between a time-sharing session and a batch job is that a session is interactive, but a job is not. Thus, during a session, the user maintains a dialogue with the system to control input and monitor output; in a batch job, however, the command stream is entered into the system without a user/system dialogue.

Choice of Programming Languages

Under MPE/3000, the user can submit programs written in the following languages:

- FORTRAN/3000
- BASIC/3000
- SPL/3000 (Systems Programming Language)

In addition, MPE/3000 provides subsystems for editing program files, performing statistical operations, aiding in debugging programs, and running on-line diagnostic programs.

Each language translator and subsystem is accessed by a unique MPE/3000 command. The programmer need learn only one set of conventions for using these programs, because they all use the same command formats, special characters, and error-diagnostic methods. Command coding is based upon the American Standard Code for Information Interchange (ASCII) Character Set, shown in Appendix A.

Ease of Use

MPE/3000 is easy to initialize, operate, monitor, and shut down. Its operation is overseen by a user with the MPE/3000 System Manager Capability, who assigns programming capabilities of various levels to each user. *Standard capabilities* allow the typical “general applications” programmer to interact with the computer through a terminal or console. If this programmer is

planning only to compile and execute a batch job, or to run an on-line interactive program, he may need to know only a few simple MPE/3000 commands. As his needs become more extensive, however, so must his knowledge of MPE/3000. A user planning more complex operations can be assigned various *optional capabilities*. These allow him to access more sophisticated system resources for such tasks as executing privileged instructions. (Standard and optional capabilities are described in detail in *Multiprogramming Executive Operating System (03000-90005)*.)

Input/Output Conveniences

Because MPE/3000 treats all input/output devices as labeled files (or groups of files in the case of mass storage devices), the programmer may access these devices by file names rather than by device types or logical unit numbers. The file names used in programs are independent of the devices used for file input and output, and need only be associated with these devices at the time the programs are run. This means that the user can write programs without immediate concern for the physical source of input or destination of output. This independence also means that programs can be run in either batch or time-sharing mode without changing the names of the files they reference.

Files on disc can be structured for either direct or sequential access, on an exclusive or shared basis. Direct-access files contain fixed-length records; sequential files, however, can contain either fixed- or variable-length records. For files on disc, storage space is automatically allocated as it is needed. MPE/3000 permits simultaneous access of sharable disc files by many programmers.

Processing Efficiency

MPE/3000 offers the user maximum efficiency by taking best advantage of the HP 3000 Computer architecture and the rapid speed of the central processor. This, in turn, permits the rapid changes between user environments that make multiprogramming in batch and time-shared modes possible.

RE-ENTRANT CODE AND PRIVATE DATA. Within MPE/3000, many user and system functions can be active simultaneously without mutual interference. This is because the hardware provides protection of programs and guarantees the privacy of user data areas. MPE/3000 keeps code and data logically separate by organizing them into re-entrant code segments (which can be shared among users but not altered) and data segments (which cannot be shared but which can be altered by the creating user). Code-sharing means that only one copy of each program, accessible to many users concurrently, need exist in memory at any time. Code segmentation allows code to be moved from disc into main memory only when needed and dynamically relocated in main memory to accommodate other programs and routines. The re-entrancy of code means that when a program is interrupted during execution of a code segment, and another program uses that same segment, the segment is completely protected from modification and will be returned, intact, to the previous program.

STACK ARCHITECTURE. Many powerful operating system features are made possible by the computer's use of stacks (linear storage areas for data) where the last item stored in is always the first item taken out. Some of these features are

- Ease of compilation and parameter passing
- Fast execution
- Highly-efficient subroutine linkage
- Minimum memory overhead
- Dynamic allocation of temporary storage
- Rapid interruption and restoration of user environments
- Code compression
- Recursion (where a program calls itself).

MICROPROGRAMMING AND THE CENTRAL PROCESSOR. Additional economy has been provided by microprogramming many system operations normally provided by software. These operations are requested by machine instructions that each, in turn, execute many micro-instructions built into the central processor hardware. Microprogramming eliminates the repetitive coding and main memory requirements otherwise needed for recurring operations (such as moving character strings from one location to another, or scanning strings for a particular character), thereby placing the operating burden on the highly-efficient micro-processor rather than on MPE/3000 software.

VIRTUAL MEMORY. The MPE/3000 virtual memory offers users a total memory space that far exceeds the maximum main-memory size of 65,536 words. Virtual memory consists of both main memory and an extensive, flexible storage area on disc. User programs and information in the disc area are subdivided into units (segments) of code or data that are dynamically moved, through overlays, into main (core) memory for execution.

HIGH DATA THROUGHPUT. High throughput of data is facilitated by high-speed channels, double-buffering, many input/output devices, and input/output program execution (through an input/output processor) in parallel with regular program execution (by the central processor).

AUTOMATIC SCHEDULING. MPE/3000 automatically schedules all jobs and sessions according to their priorities. When execution of a running program is interrupted for any reason (such as input/output, an internal interrupt, or expiration of a time-slice), MPE/3000 passes control to the program of highest priority awaiting execution. In general, time-sharing sessions have highest priority, followed by batch jobs.

System-to-System Compatibility

All HP 3000 Computers operate under one version of MPE/3000. This means that programs prepared on one HP 3000 can be run on any other without modification (provided that all devices required by those programs are connected on-line). It also means that users moving from one installation to another need not undergo additional training or read additional documentation to prepare themselves for the new environment.

Program Security

Each user operates in an environment protected from interference by other users; program protection is provided by the hardware.



SYSTEM MANAGER AND SYSTEM SUPERVISOR CAPABILITIES

MPE/3000 provides tools for the overall management and control of the system to users with the *System Manager* and *System Supervisor Capabilities*.

The System Manager Capability allows a user to have final control of the overall use of the system by defining the accounts under which users access MPE/3000. It also allows a user to assign the *Account-Manager Capability* that permits its possessor to name the users who can access an individual account, and to partition the file domain of that account into groups of files.

The System Supervisor Capability permits a user to supervise and control the general operation of the system by generating and modifying MPE/3000, displaying job status information, scheduling jobs, running batch and time-sharing programs, permanently allocating code in virtual memory, and transmitting messages to other users.

It is important to bear in mind that the above capabilities are granted to users through the assignment of special *attributes*, and in no way imply formal responsibilities or duties. Thus, a regular programmer may have the System Manager Capability or the System Supervisor Capability, or both capabilities—in addition to several others discussed later. For this reason, it is best to speak of “a user with System Manager Capability” rather than a formal “System Manager.”

SOFTWARE

MPE/3000 is furnished to the customer on a reel of magnetic tape. As part of the operating system software, these major components are provided:

- *System Configurator*, for configuring and making a backup copy of MPE/3000.
- *System Initiator*, for installing and starting up MPE/3000.
- *Command Interpreter*, for handling the commands that allow the user to interact with MPE/3000 through a terminal or batch input device.
- *File Management System*, for providing uniform access to disc files and standard input/output devices.
- *Memory Management System*, for dynamically allocating main memory among contending users.
- *Dispatcher*, for allocating central-processor time among programs in execution.
- *Input/Output System and Drivers*, for scheduling, initializing, monitoring, and completing input/output requests for standard devices (accessed through the file system).
- *System Library*, for storage of frequently-used routines sharable among many users.
- *Segmenter Subsystem*, for segmenting and loading programs, and resolving references to external code.

No additional or auxiliary software is required to install, operate, or maintain MPE/3000.

HARDWARE

The minimum hardware configuration required by MPE/3000 is

- HP 3000 Computer, including central processor and 32,768-word main (core) memory.
- Multiplexor Channel
- Moving-Head or Fixed-Head Disc
- System Console (Model 33 Teletype Terminal)
- Magnetic Tape Unit
- System Timer

The disc is used to contain portions of MPE/3000 and user programs. The console is used for those operator commands needed to initialize and maintain MPE/3000, and optionally for input/output of user programs. (The console must be a Model 33 Teletype Terminal when the minimum hardware configuration is used.) The magnetic tape unit is used for cold-loading MPE/3000 and for storage and input/output of user programs. The system timer is used for sequencing various operations.

The following optional hardware can be added to the system:

- Main Memory (for a total of up to 65,536 words)
- Additional System Timer
- Discs (Fixed-Head or Moving-Head)
- Magnetic Tape Units
- Card Readers
- Line Printers
- Terminal Controllers
- On-Line Terminals

In an expanded system, the System Console need not be a Model 33 Teletype; any EAI-compatible device, including a CRT device, can serve this purpose.

With this optional equipment, many hardware configurations are possible. For example, a small system used for batch processing might appear as shown in Figure 1-1. In this system, the card reader is used for input and the line printer for output, while the disc is used for storing programs and data awaiting further processing.

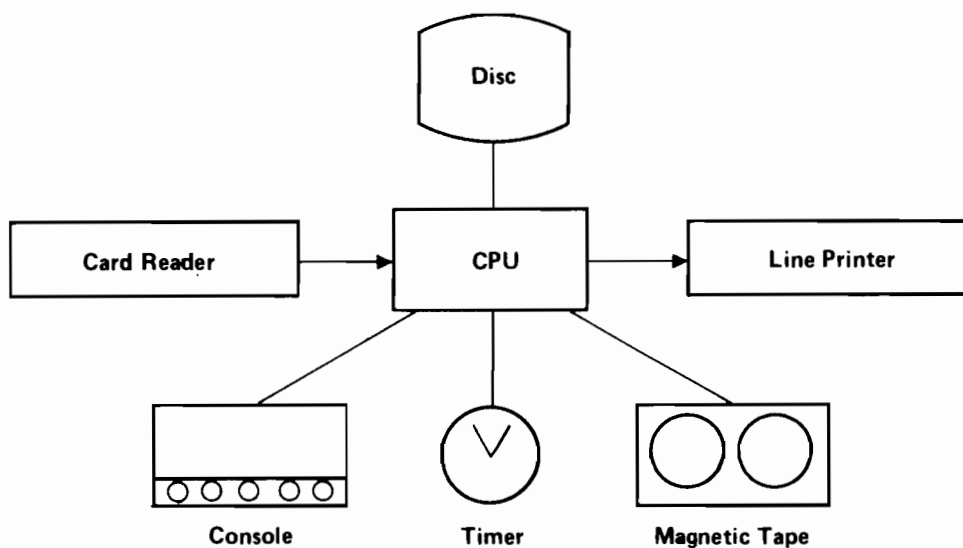


Figure 1-1. Small Batch System

A small time-sharing system might be configured as shown in Figure 1-2. In this system, up to 16 terminals connected to a terminal controller are used for input and output.

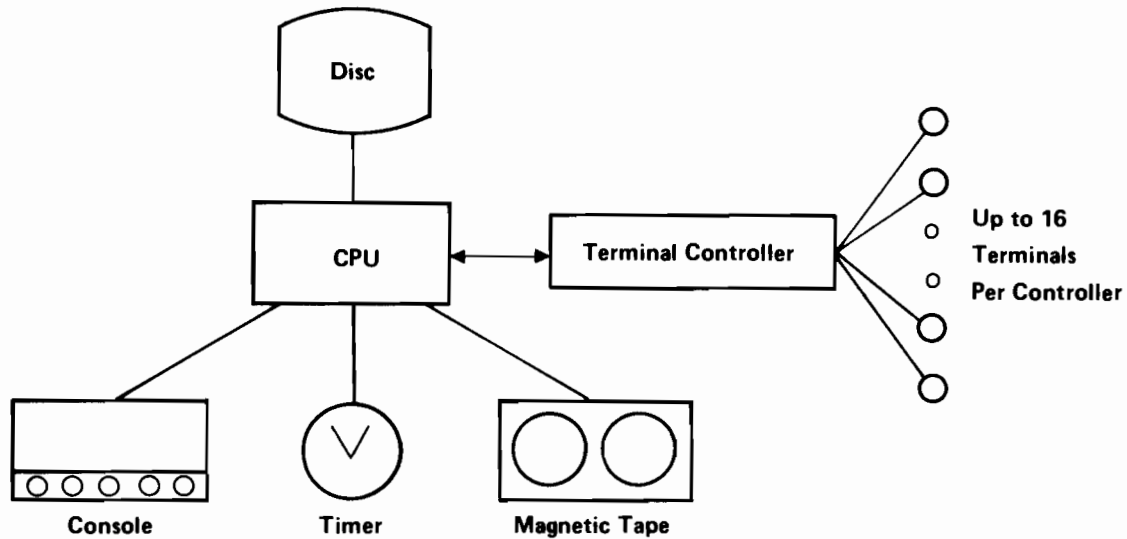


Figure 1-2. Small Time-Sharing System

A combined batch and time-sharing system is illustrated in Figure 1-3. Here, batch and time-sharing operations can occur separately or concurrently.

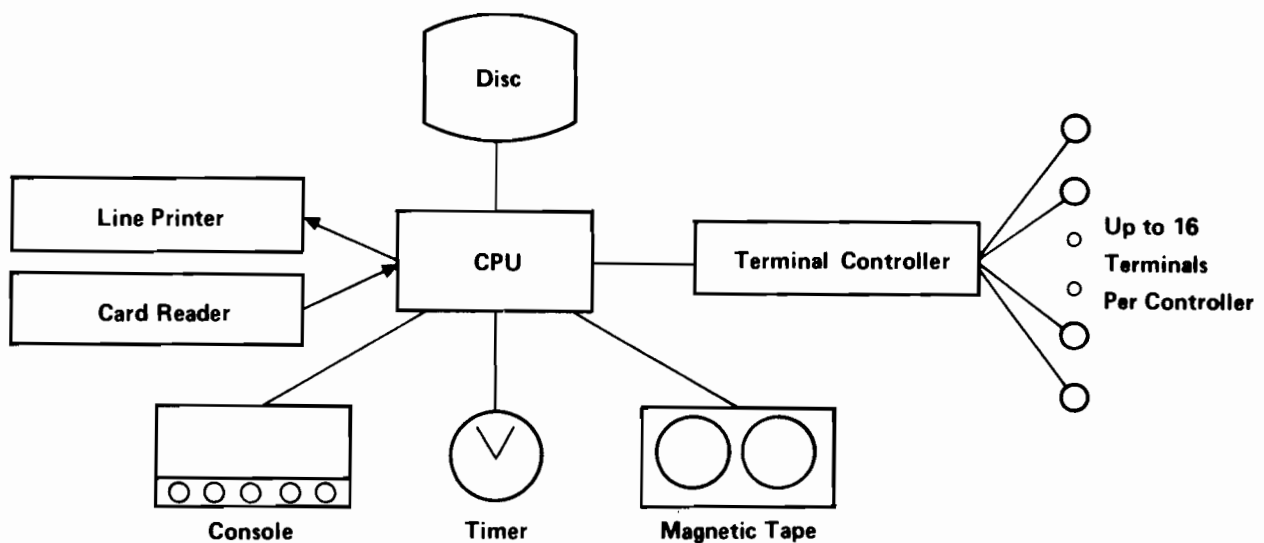


Figure 1-3. Combined Batch and Time-Sharing System

A large processing system is shown in Figure 1-4. This system incorporates a large central memory, fixed-head and moving-head discs, many input/output devices for multiple-job batching, and several controllers, each connected to up to 16 terminals for an extensive time-sharing network.

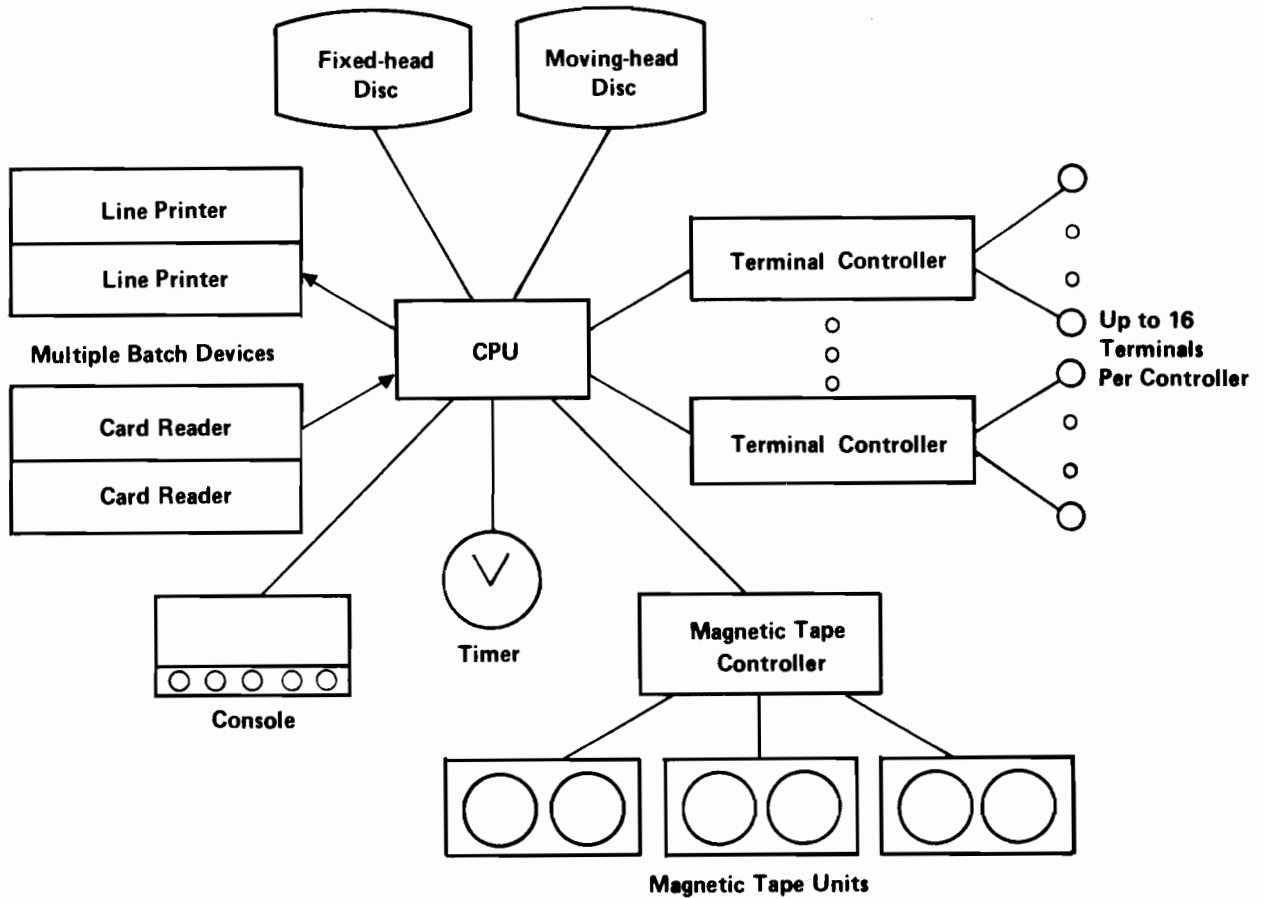


Figure 1-4. Large Processing System

SECTION II

How MPE/3000 Operates

In supervising the compilation and execution of user programs, MPE/3000 performs the following tasks:

- Controls all input/output.
- Allocates memory to user programs and the system routines required for their support.
- Schedules batch programs for access to the central processor.
- Allocates central processor-time to programs.
- Swaps portions of programs to and from main memory as required during execution.

NOTE: *For users employing the MPE/3000 System Manager, System Supervisor, and Standard Capabilities only, MPE/3000 automatically performs these and most other functions at their command. Generally, such users need not be concerned with the internal aspects of MPE/3000 discussed in the first part of this section, and may skip ahead to the heading ACCOUNT/GROUP/USER ORGANIZATION for a discussion of their relationship to the system. However, the information in the beginning of this section will be of value to users with the System Supervisor Capability who plan to create and run processes; they should read it carefully.*

MPE/3000 RESIDENCE

MPE/3000 resides in three general areas:

1. Those routines most frequently used reside in *main memory*, all of which is part of virtual memory. (Routines not residing in main memory are moved there only when required. This ensures that as much main memory as possible remains available for the execution of user programs.)
2. Frequently-used routines reside on *portions of disc within virtual memory*; these disc areas are treated as logical extensions of main memory.
3. All remaining routines (including a permanent copy of each compiler) are stored in *areas of disc outside of virtual memory*; these areas are treated as conventional secondary storage.

PROCESSES

In MPE/3000, programs are run on the basis of *processes* created and handled by the system. A process is not a program itself, but the unique execution of a program by a particular user at a particular time. Therefore, if the same program is run by several users, or more than once by the same user, it is used in several distinct processes.

The process is the basic executable entity in MPE/3000. A process consists of a process control block that defines and monitors the state of the process, a dynamically-changing set of code segments, and a data area (stack) upon which these segments operate. (Thus, while a *program* consists of instructions (not yet executable) and data in a file, a *process* is an executing program with a data stack assigned.) The code segments used by a process can be shared with other processes, but its data stack is private (Figure 2-1). For example, each user working on-line through the BASIC language is running his program under a separate process; all use the same code (the only copy of the BASIC interpreter in the system), but each has his own stack. (The interrelationship of the process control block, code segments, and data stack is discussed in detail under "PCB/Code Segment/Stack Interaction.")

Processes, and the elements that comprise them, are invisible to the programmer accessing MPE/3000 through standard capabilities; in other words, this programmer has no control over processes or their structure. For this user, MPE/3000 automatically creates, handles and deletes all processes. The user with certain optional capabilities, however, can create and manipulate processes directly.

The entire MPE/3000 system is a collection of processes that are either active or dormant. These processes are logically organized in a tree structure (see Figure 2-2); each process has only one immediate ancestor (father) but could have several immediate descendants (sons). The root process that controls all other processes in MPE/3000 (Figure 2-2) is the only process that has no ancestor. It is the first process established, and is created during system initialization. When the system is configured, the root process uses the data specified to create various system processes, including the input/output, and user controller processes; each of these, when activated, creates its own descendent processes. (A *system process* is a process that executes on behalf of MPE/3000.)

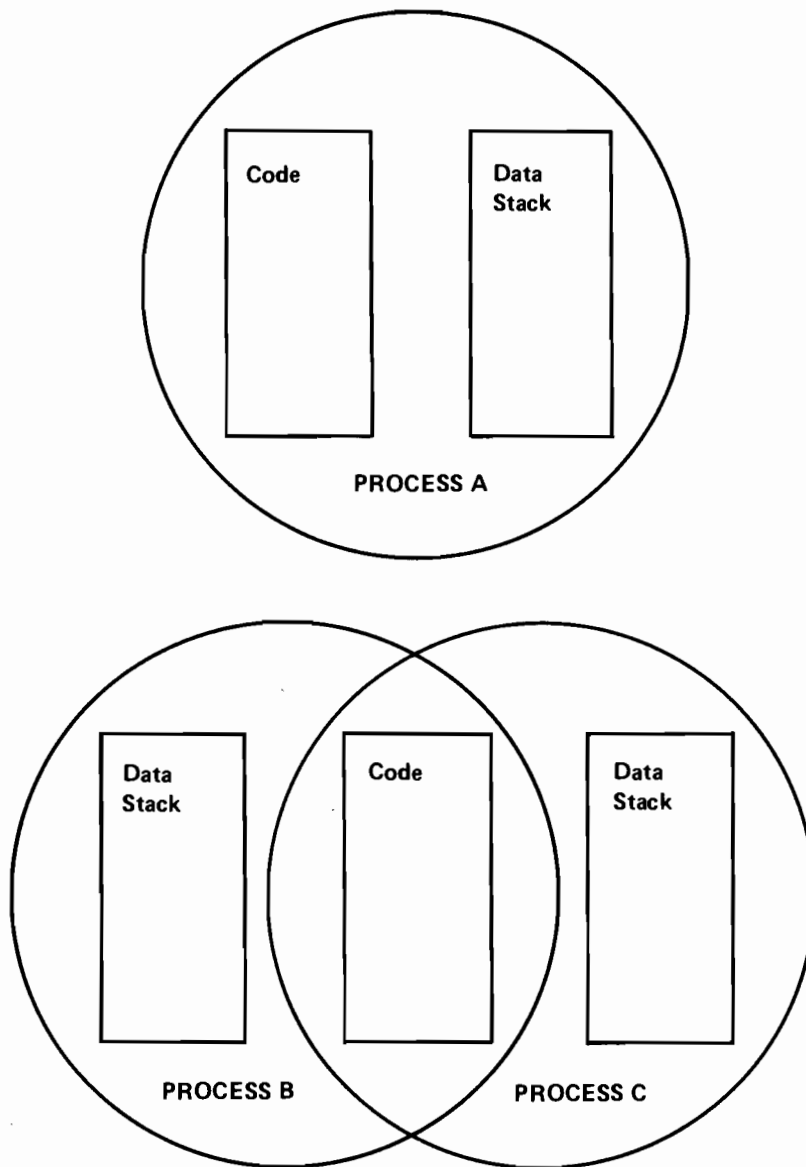


Figure 2-1. Code-Sharing and Data Privacy

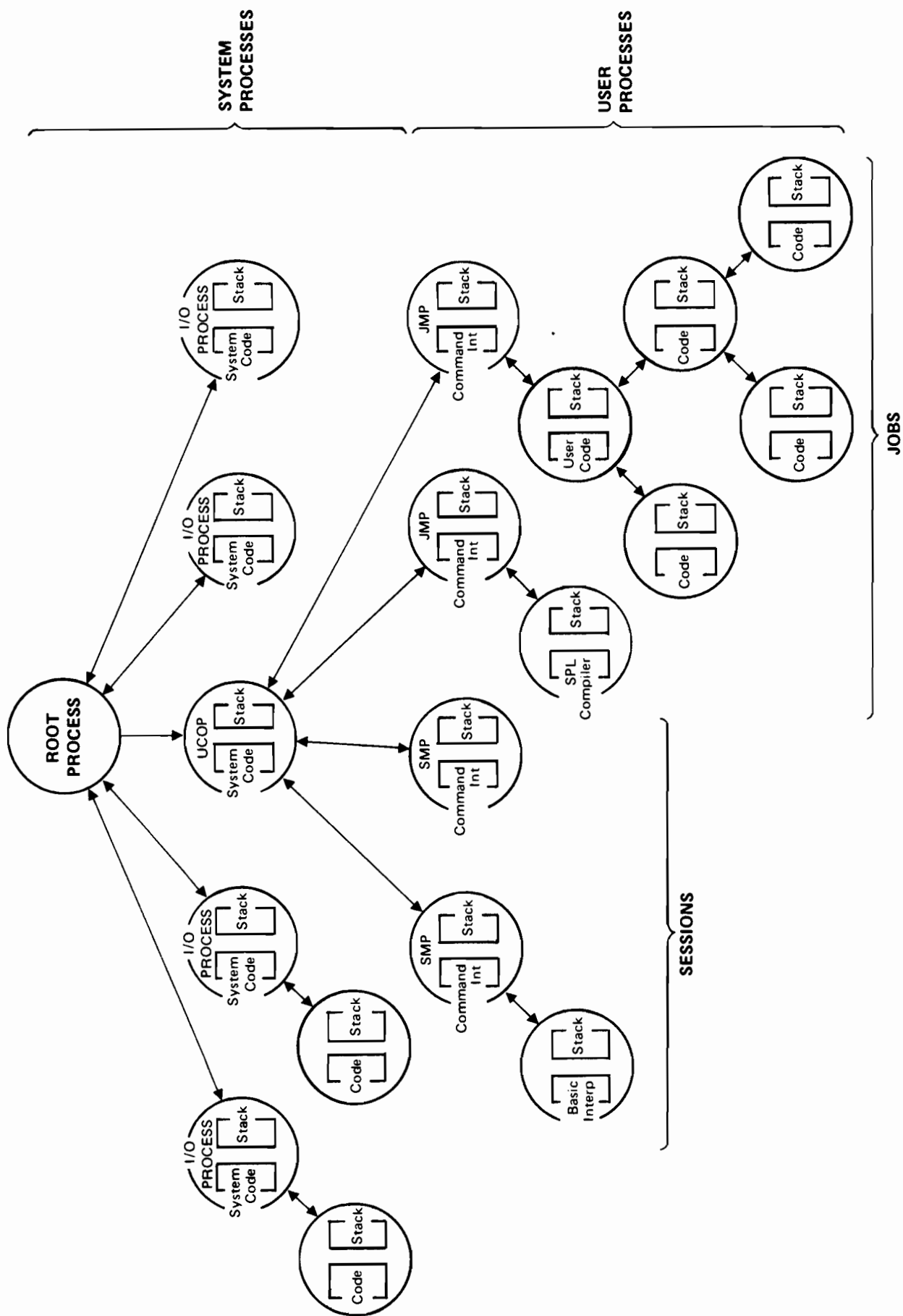


Figure 2-2. Process Organization

The user controller process (UCOP) is the ancestor of all existing user processes. (A *user process* is a process that runs on behalf of the user.) When the user first accesses MPE/3000, the UCOP creates a *main process* that oversees the execution of his program. If the user is running a batch program (called a *job*), a job main process (JMP) is created. If he is executing a time-sharing program (*session*), a session main process (SMP) is created.

All JMP's and SMP's use system code, the MPE/3000 Command Interpreter, interfacing directly with the user. They may create descendants that use system code, such as the BASIC/3000 Interpreter or FORTRAN/3000 or SPL/3000 compilers when translating source programs; when executing user programs, of course, the descendant processes use the user's code.

PROCEDURES

In MPE/3000 most individual programming operations are handled by unique sets of code known as *procedures*. (Others are handled by special programs or interrupt routines.) In SPL/3000, the language in which MPE/3000 is coded, a procedure is defined by a procedure declaration consisting of

- A procedure head, containing the procedure name and type, parameter definitions, and other information about the procedure
- A procedure body, containing executable statements and data declarations local to this procedure

As part of their function, many procedures also return parameter values to the processes that invoke them.

In SPL/3000, each procedure is executed by a corresponding *procedure call*. When a procedure call is encountered, control is transferred to the procedure, which runs until an exit is encountered. Control is then returned to the statement following the call for that procedure.

While a procedure in SPL/3000 (as in ALGOL) is similar to a subroutine in many respects, there are major differences between the two:

- When referencing a logical set of code, a subroutine call is significantly faster and more efficient than a procedure call, while a procedure call has more powerful (and general) applications.
- A subroutine declaration may be used within a procedure declaration, but a procedure declaration may not appear in a subroutine.
- A procedure allows dynamic storage allocation, permitting local variables.
- A subroutine can contain non-local references to variables declared within the scope of the procedure in which it occurs.

Unlike procedures, subroutines are not suitable for solving complex problems that require explicit stack manipulation or many temporary storage locations. They *are* useful, however, for various utility and arithmetic-function routines.

In addition to the procedures provided by MPE/3000, SPL/3000 allows the user to write procedures to suit his own purposes. To distinguish MPE/3000 system procedures (which are always available to the user, directly or indirectly) from any other procedures, the term *intrinsic* is applied to MPE/3000 procedures. Similarly, the term *intrinsic calls* is used to denote the procedure calls that reference MPE/3000 procedures. From this point on, these terms will carry these respective meanings throughout this manual.

Each large functional unit of MPE/3000, such as the command interpreter, file manager, or input/output system, consists of many intrinsics. The creation, handling, and deletion of processes is performed exclusively through intrinsics not directly accessible by users with the standard MPE/3000 capabilities. (However, these intrinsics can be accessed directly by users having certain optional capabilities described in Part 3.)

SYSTEM LIBRARY

The MPE/3000 system library is a collection of frequently-used routines that can be readily shared among many users. A library routine can consist of one procedure from a code segment, an entire segment, or a set of segments. Some of these segments reside permanently in virtual memory, while others are stored elsewhere on disc and called into virtual memory as needed.

USER INTERFACE

In requesting system functions, the programmer never interacts directly with the computer hardware. Instead, he interacts with a dual-level software interface that accepts *commands* (for general functions external to his program) and *intrinsic calls* (for specific functions within his program). In general, the commands and intrinsic calls themselves access uncallable intrinsics that reference machine instructions. These, in turn, execute micro-instructions from the microcode hardwired into read-only memory (Figure 2-3). It is this microcode only that operates directly on the computer hardware.

Commands

Commands are used to initiate and terminate jobs, declare file characteristics, compile and execute programs, call various utility subsystems, and perform other broad functions external to user programs. A user with System Manager Capability uses them to create and alter accounts; a user with System Supervisor Capability uses them to run programs, display information, or manage the scheduling queue. Commands are entered through any standard input device, typically the card reader (for jobs) or the terminal (for sessions). When the input device is connected on-line to MPE/3000, a main process is automatically initiated that uses the MPE/3000 command interpreter program to read and translate the commands. The first command entered must always be one that initiates a job or session.

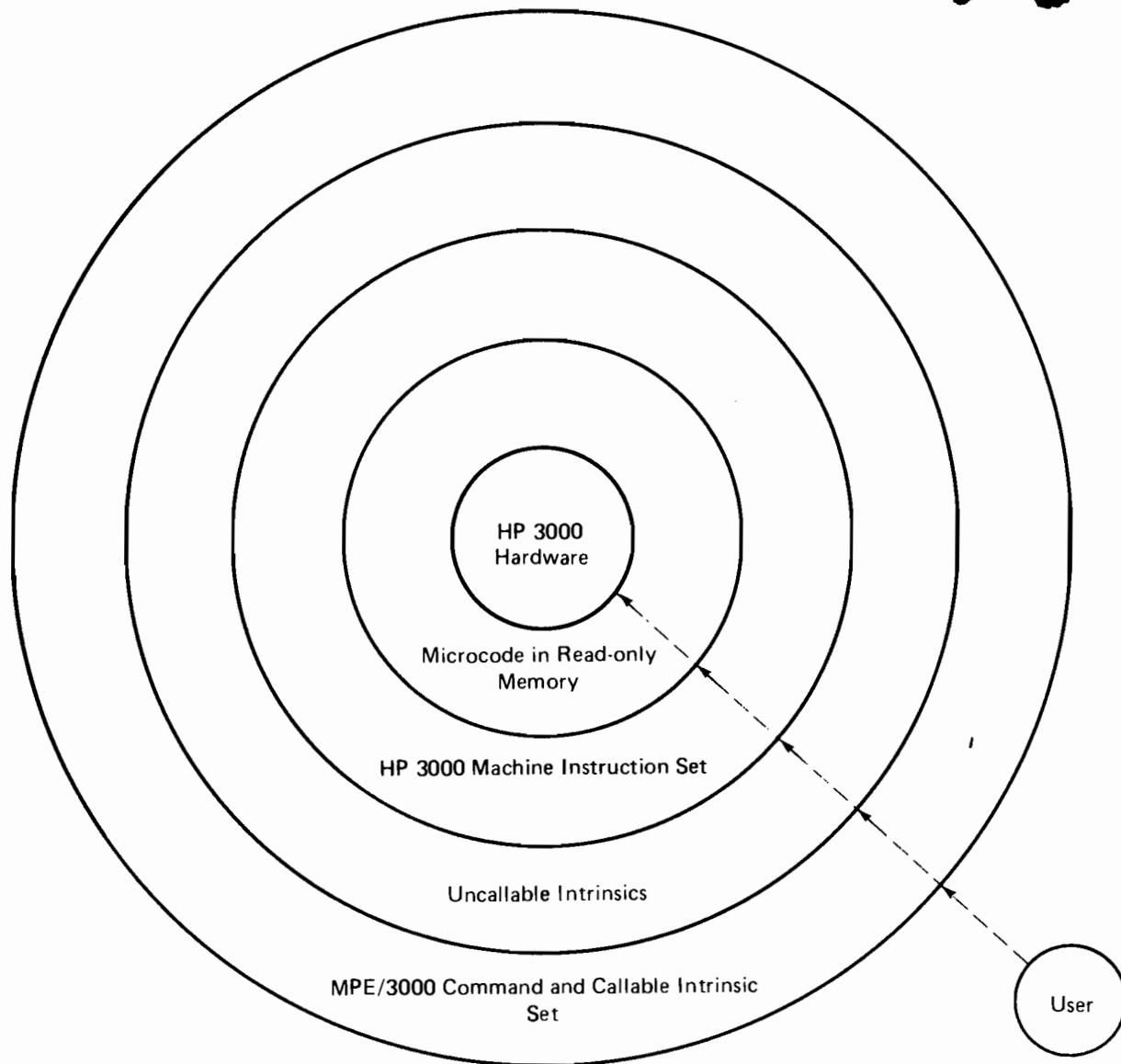


Figure 2-3. User/Software/Hardware Interface

Each command invokes an intrinsic that implements the function desired. (Such intrinsics are called *command executors*.) When a command is entered, the command interpreter checks it against the names in the MPE/3000 command dictionary. If the command is valid, the command interpreter passes the command parameter list image to the appropriate executor for execution. When the action requested by the command is satisfactorily completed, control returns to the command interpreter.

Intrinsic Calls

Intrinsic calls implement MPE/3000 functions requested *programmatically* (within a user's program), such as reading, writing on, and updating files; skipping forward and backward on files, or returning system table information to the user's program. (For some intrinsics, programmers who also have System Supervisor Capability can specify certain parameter values available to no other programmers.) In an SPL/3000 program, the user writes the intrinsic calls explicitly, but in FORTRAN or BASIC programs, in most general applications, the compiler generates any necessary intrinsic calls automatically — they are invisible to the user. (Provisions are also available in FORTRAN and BASIC that allow the user himself to call intrinsics, at his option.) All MPE/3000 intrinsics are treated as external procedures by user programs. External linkages from user programs to intrinsics are satisfied when the user programs are segmented and allocated residence in virtual memory.

The set of intrinsics available for a user depend upon the capabilities assigned to him by the user who is managing the account under which he accesses MPE/3000. This set of intrinsics defines the computer for that user. Generally, the user calls *callable* intrinsics appropriate for a particular problem. These, in turn, invoke *uncallable* intrinsics that actually execute machine instructions, thus maintaining the integrity of the system by protecting it against direct manipulation by the user. (Uncallable intrinsics are those that can be invoked only by code running in privileged mode.)

INPUT/OUTPUT

The MPE/3000 input/output system schedules, initiates, and completes all input/output requests for all HP-supported devices connected to the system. Normally, the input/output system remains invisible to the user, since he accesses it indirectly through the MPE/3000 file management system.

The input/output system includes an independent controller for each type of hardware device connected to the computer. Each controller is associated with an input/output monitor process. This process services the requests for the controller and calls input/output initiator, completion, and error driver routines as required. Requests to the input/output system are issued by the file management and memory management systems.

File Management System

The file management system provides a uniform method of directing input and output of information. It handles various input/output applications, such as the transfer of information to and from user processes, compilers, and data management subsystems.

MPE/3000 treats each set of input or output information as a set of logical records called a *file*. When a file is created, MPE/3000 automatically allocates a device for its storage. Input read from a hardware device such as a card reader is accepted directly from that device. Similarly, output from an executing process is transmitted directly to the required device (such as a line printer).

File information is transferred between storage media in units of one physical record each. On unit-record devices, the size of a physical record is determined by the device itself. (As an example, a physical record transferred from a card reader is always comprised of one card image.) On disc and magnetic tape, however, a physical record is comprised of a group (*block*) of logical records. The block size is specified by the user and determines the size of the unit transmitted between the device and virtual memory. Input/output and blocking are performed automatically by MPE/3000, allowing the user to remain unaware of the actual record-handling details.

The user references a file by the *file name* assigned to the file when it is created. When he does this, MPE/3000 automatically determines the device or disc address where the file is stored and accesses the file for him. Throughout its existence, the file remains device-independent.

The MPE/3000 file management system allocates devices for the storage of files on the basis of specifications from the user. For example, the user can request a device by generic type name (such as any magnetic tape unit or line printer), or by the logical device number that refers to a specific device. Logical device numbers are related to all devices when MPE/3000 is configured.

Scheduling

The MPE/3000 scheduling system ensures that all processes competing for the central processor access it in an orderly manner. All processes are placed in a master scheduling queue in order of their priority. (This queue is actually a linked list of PCB's — process control blocks — for the processes scheduled.) When a process in execution is completed, terminated, or interrupted, the MPE/3000 dispatcher program searches the master queue for the process of highest priority awaiting execution and transfers control to that process.

Within the master queue, five standard subqueues are used for scheduling processes created by user programs (Figure 2-4). Since MPE/3000 schedules each process independently, not all processes for a job are necessarily entered in the same subqueue.

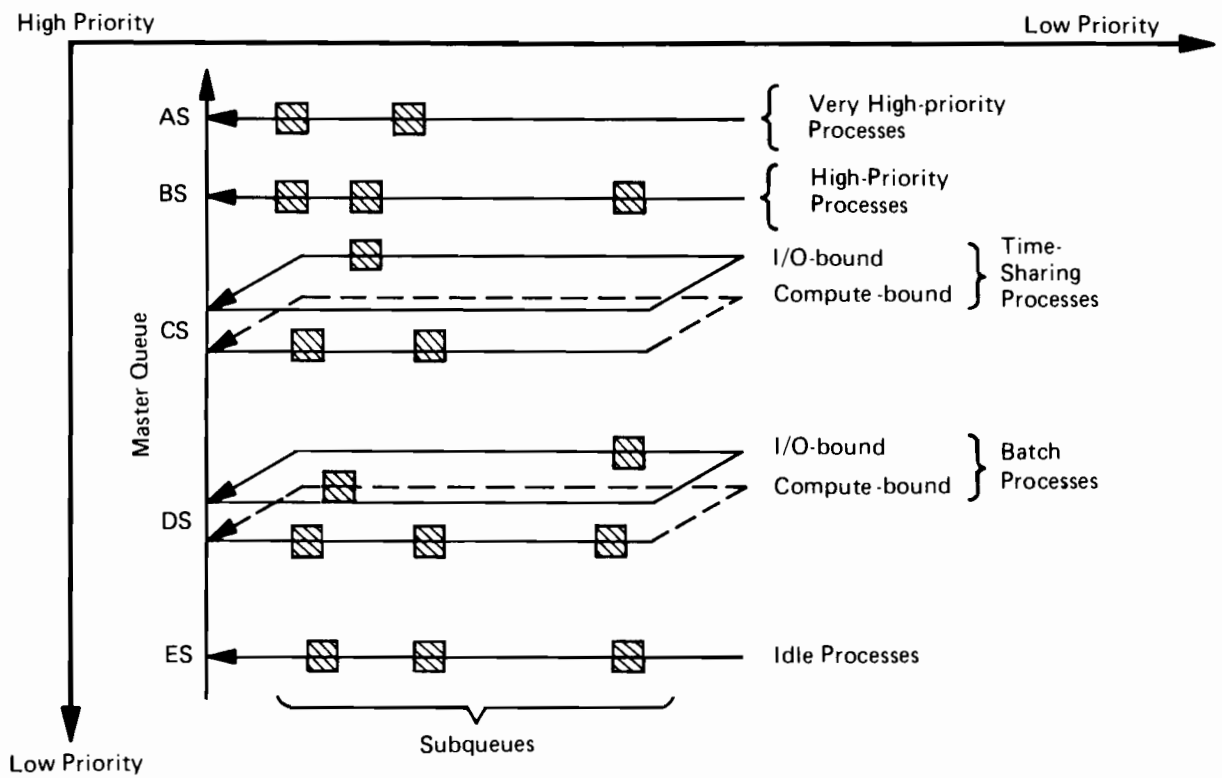


Figure 2-4. Scheduling Queues

Three of the standard subqueues are linear in structure; two of these (AS and BS) are high-priority subqueues that are presently available for general purposes, but in subsequent releases of MPE/3000 will be used for processes with new MPE/3000 optional capabilities; the third linear subqueue (ES) is available for idle processes with low priority. In a linear subqueue, the process with highest priority accesses the central processor first and maintains this access until the process either is completed or suspended to await input/output. (The master queue itself is a linear queue.)

The other two standard subqueues are circular subqueues; one (CS) is used for time-sharing sessions and the other (DS) is used for multiprogramming batch jobs. In these subqueues, each process accesses the central processor for an interval (time-slice) of limited duration. At the end of each time-slice, or when an interrupt occurs, control is transferred to the next process in the queue, continuing in a round-robin fashion. This time-slicing is controlled by a system timer. Each of the two circular subqueues is composed of two sub-subqueues. In both subqueues, the lower-priority sub-subqueue services compute-bound processes while the higher-priority one services input/output-bound processes (those that require excessive time for input/output). When a compute-bound process in one of these subqueues is repeatedly suspended for input/output, rather than exceeding its time-slice, the scheduling system moves it into the slightly-higher input/output-bound sub-subqueue. This ensures that the process obtains adequate central processor time. When the process is no longer input/output-bound, it is returned to the compute-bound sub-subqueue.

COMPILATION

User programs are entered into the computer in a source-language, translated into binary form by a process executing a compiler, and stored on disc (outside virtual memory). (Figure 2-5A.) Because the code making up a compiler is re-entrant, it can be shared by many users. Therefore, only one copy of a compiler is required in memory no matter how many programmers need it at one time to compile their programs.

MPE/3000 regards user source-language programs as consisting of program units. A program unit is the smallest divisible part of any program or subprogram. For example, in SPL/3000, this is an outer block program unit or a procedure; in FORTRAN/3000, this is a main subroutine, subroutine, function, or block data unit. Thus, a *program* consists of one or more program units, one of which must be an outer block (SPL/3000) or main subroutine (FORTRAN/3000) program unit. A *subprogram*, which is the smallest individually-compilable entity, consists of one or more program units.

When a source-language program is compiled, each program unit is transformed into a relocatable binary module (RBM) that contains user code plus header information that labels and describes this code.

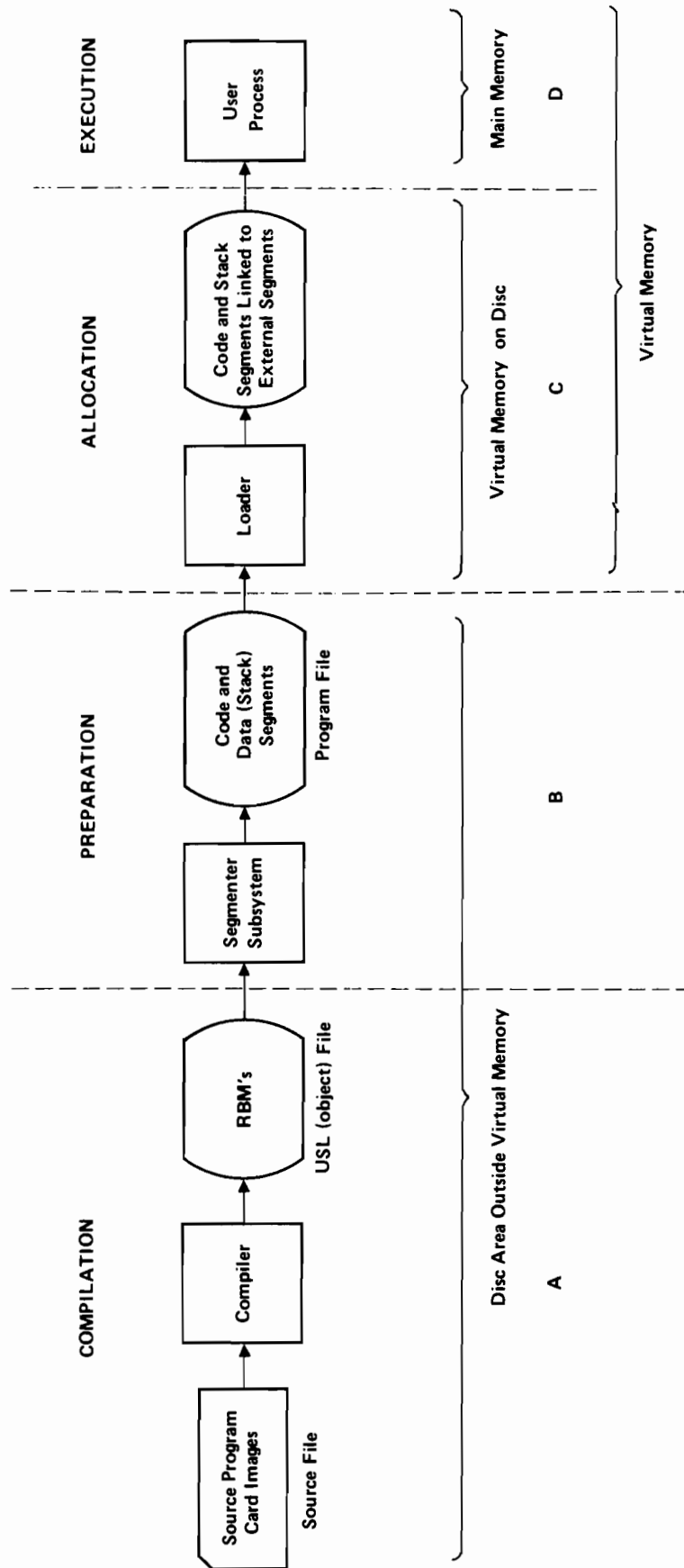


Figure 2-5. Program Management

PROGRAM PREPARATION

The RBM's that result from compilation of a user's program onto disc make up a user sub-program library (USL) file. For each program unit compiled, there is one RBM on the USL.

The USL is not executable, however. Instead, it must be *prepared* for running by MPE/3000. During preparation, a process running the MPE/3000 segmenter binds the RBM's from the USL (object file) into linked code segments arranged in a *program file* (Figure 2-5B). Each segment contains machine instructions produced from the user's program, plus linkages to other segments. The code in a segment cannot be altered because it is treated as read-only information; thus, it remains re-entrant and can be executed repeatedly by many users. When a program is segmented, a special segment for the input of user data is also initially defined; this contains the stack.

ALLOCATION/EXECUTION

Next, the program is allocated and executed. In *allocation*, a process running the MPE/3000 loader moves the segments from the program file into virtual memory on disc, and binds them to referenced external segments from the system library (Figure 2-5C). MPE/3000 then creates a process to run the program, and this process transfers into main memory the code segment containing the entry point of the program to be executed and its associated data stack segment (Figure 2-5D)

Execution now begins. As it progresses, many new processes may be created, run, and deleted. For each process in execution, one or more code segments and one stack, operating under control of a process control block (PCB), typically exist in main memory. Not all code segments belonging to a process in execution need exist in main memory simultaneously. Typically, a process operates on a set of segments that are dynamically swapped between main memory and disc. MPE/3000 maintains records of the frequency each segment is used, so that those used least frequently in main memory become the most eligible for swapping out. The user never needs to determine whether a segment is in main memory or on disc at any given time — this is always done for him automatically by MPE/3000.

During allocation/execution, MPE/3000 keeps track of each code segment by maintaining information about its nature and current location in the code segment table (CST). Similarly, information about the stack is dynamically recorded in the data segment table (DST).

A particular program can be run by many user processes simultaneously, with all processes accessing the same copy of code. But unlike the program code segments, the data segment containing the stack is private to each user's process and cannot be shared among others. As execution progresses, data enters and leaves the stack dynamically. Within the stack, data is arranged as a linear group of items accessed from one end (called the *top-of-stack*). When the last instruction in a process is executed, MPE/3000 releases those segments associated only with that process, including the stack segment, and deletes their related entries in the CST and DST; however, shared code segments and permanently-allocated segments are not released. All descendants of this process are deleted, and all files opened by it are closed. Released segments formerly belonging to the process can now be overwritten by other segments.

PCB/CODE SEGMENT/STACK INTERACTION

Each PCB contains all information needed to control a process. This information includes the priority of the process, and pointers to queues and ancestor and descendent processes. This information is maintained for all PCB's by the job or session.

Each code segment executed by a process can contain one or more program units that include calls to procedures elsewhere in this segment or in another segment. Within a program unit, there are generally many executable machine instructions. Each procedure call also references machine instructions that handle the procedure requested.

When a code segment is executing in main memory, it is defined by pointers in three hardware registers: the Program Base (PB), Program Counter (P), and Program Limit (PL) registers (Figure 2-6). (There is only *one* set of these registers; at any particular instant, their contents refer to the code segment currently in execution.) The PB register contains the absolute address of the starting location of the segment in main memory. The P register holds the absolute address of the instruction currently being executed. The PL register indicates the absolute address of the last location of the code segment. Execution can only be transferred from this segment by a call or exit instruction referencing a procedure in another segment, in which case the PB, P, and PL registers are re-set to reflect the characteristics of the new segment. Whenever an instruction is executed, the PL and PB registers are checked to ensure that referenced addresses fall within the proper segment boundaries. This *bounds check* guarantees that other programs and the system itself are protected against improper access. Since all addresses within a code segment are relative to the contents of the three program registers, (and sometimes other registers as well) a segment can be relocated anywhere in main memory and only the register contents need be changed to reflect this transfer.

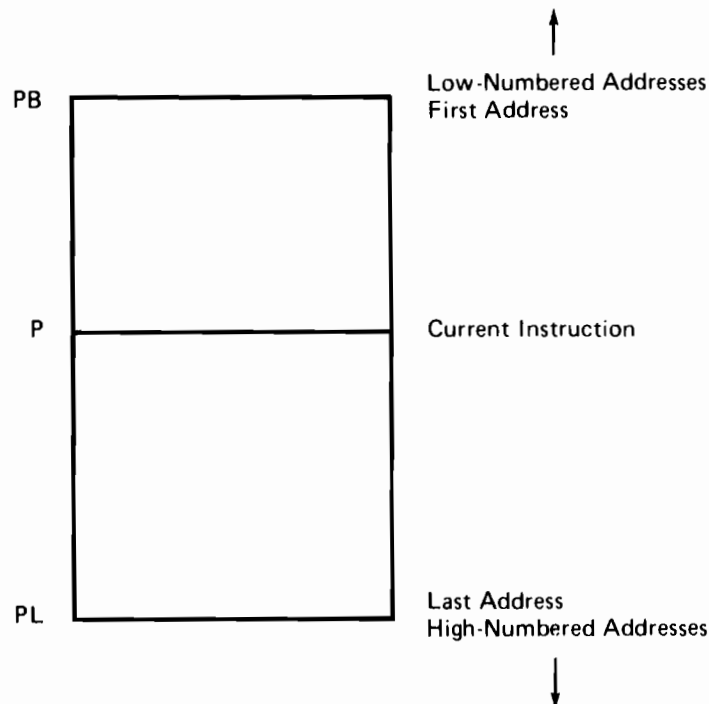


Figure 2-6. Code Segment and Associated Registers

As with code segments, there are normally several stacks resident in memory. (One stack exists for each process.) But since the execution of any process is interleaved with that of others, only one stack is active at any particular instant. Most dynamic computational operations take place on the stack. The last element added to the stack is placed in the word at the *top-of-the-stack*. In this position, it will be the first element removed when the process associated with the stack requests data from the stack. (In other words, the last element in is the first one out.) Each time data is added to the stack, the previous top element becomes the second element from the top; each time the topmost element is removed, the second element returns to the top of the stack.

Programmers operate *directly* on elements in the stack only when using the SPL/3000 language. However, any program in any other language references the stack implicitly when it manipulates data, although the user need not be aware of this.

The stack data segment consists of two general areas: a process control block extension (PCBX) area and a stack area. The *PCBX* contains certain frequently-needed information from the PCB, that need not reside in main memory, such as register settings and file pointers; it is always contiguous with the stack segment. (The PCB is used when the process is *not* running in main memory; the *PCBX* is used when it is.) The *stack area* is the most significant part of the stack data segment; this is the area where the user's data is stored and manipulated. The boundaries of this area, and its logical subdivisions, are delimited by pointers in registers (Figure 2-7). (At any instant, these registers always apply to the stack belonging to the currently-executing process.)



The Data Limit (DL) register contains the absolute address of the first word of the stack area in main memory. The area between this address and that stored in the Data Base (DB) register is an SPL/3000 *user-managed area* that can be accessed and used only through SPL/3000 programs, such as compilers. These programs sometimes require this space for SPL/3000 own-arrays. (A part of this user-managed area, that between the addresses DB-10 and DB-1, is the subsystem data area reserved for data used during subsystem operation.)

The DB register points to the beginning of the *global area* within the stack area. This area is used for global variables (those declared within the data group of an SPL/3000 main program, and thus usable by any procedure within that program). This area also contains global arrays and pointers to those arrays. The end of the global area is initially indicated by the Stack Marker (Q) register, which also denotes the beginning of the *local area*. (As will be illustrated later, the Q-register pointer changes whenever the running process calls or exits from a procedure.) At any given time, the local area contains data local (relating only) to the procedure currently in execution. The end of the local area is delimited by the Top-of-Stack (S) register, which points to the last item in the stack. All storage between the addresses in the S register and the Stack Limit (Z) register, is unused space that remains available for additional data. The Z register indicates the last main memory location that can be used by user data in the stack area. Beyond the address in the Z register, a special zone is available for stack overflow — a condition that occurs when the S-register pointer must be moved beyond the Z-register pointer and space must be provided for certain end-of-stack information.

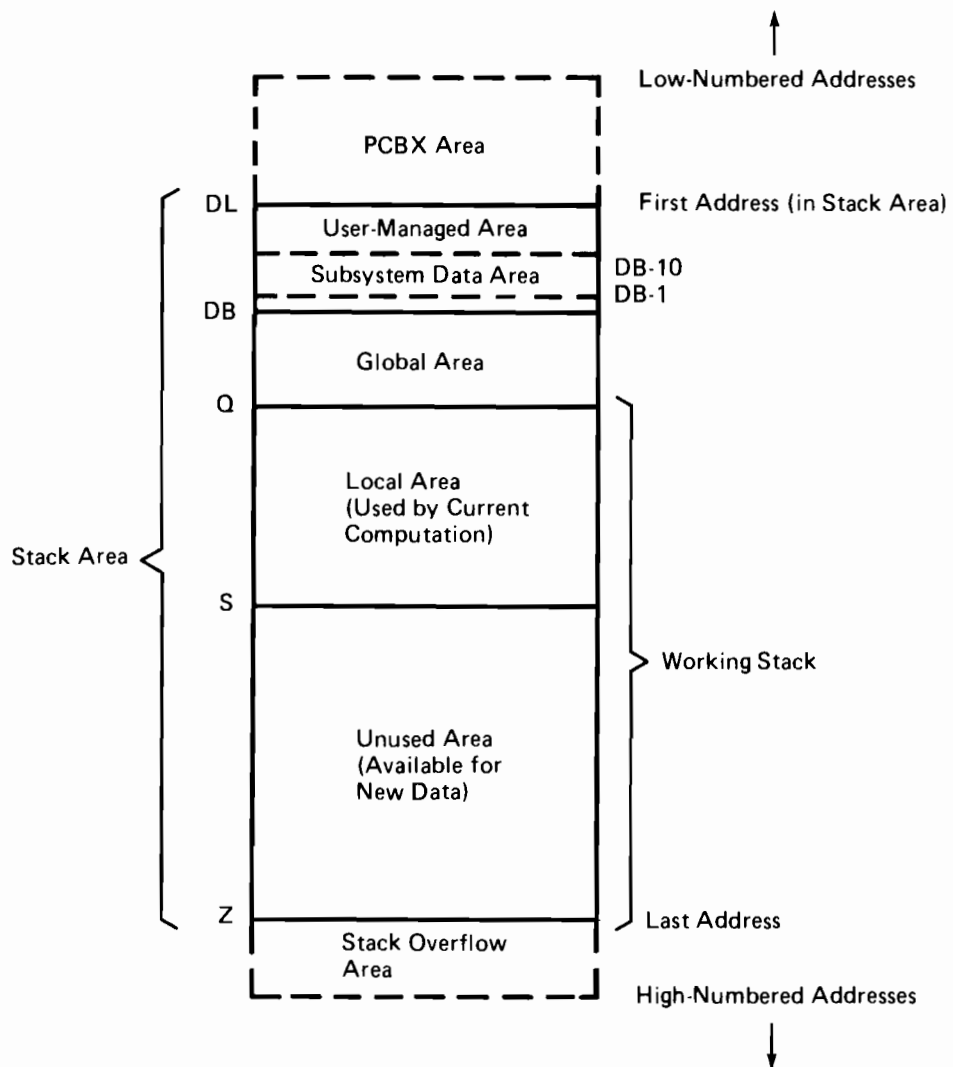


Figure 2-7. Data (Stack) Segment and Associated Registers

The contents of the DL and Z registers delimit the boundaries of the stack area. Through bounds checks that reference these registers, MPE/3000 (and certain hardware provisions) ensure that the user's data remains within these limits, and that no other user accesses this area. Thus, the privacy of the user's stack is guaranteed. All locations in the stack are addressed relative to the DB, Q, or S addresses.

Although the top-of-stack is logically indicated by the S register, up to four of the topmost elements of the stack can actually exist in central processor registers (the TR registers) rather than in main memory — in effect, the stack “spills over” from memory into these registers. These conditions greatly enhance processing speed. The number of TR registers currently in use is indicated by the SR register. The absolute address of the last item of the stack actually residing in main memory is stored in the SM register. The contents of the S register are actually denoted by

$$S = SM + SR$$

Thus, the S register is said to contain the *logical* top-of-stack rather than the actual top-of-stack in memory.

Before a process begins execution, stack space is first reserved for global data, beginning at the DB-address and terminating at the Q-address, which denotes the beginning of the dynamic *working stack* (Figure 2-8A). At this time, no data is stored beyond the Q-address, and so the the S register also points to the same address as the Q register — that is, the bottom and top of the working stack coincide. But, as the process begins execution, data is added to the stack, and the S-pointer (top-of-stack) moves away from the Q-pointer (Figure 2-8B). If, at some point, the process encounters a procedure call, a new area for data local to that procedure must be defined. To do this, the system hardware places a group of four words called *stack marker* on top of the stack to save information necessary to re-create the currently-defined local area later. The Q and S registers are then pointed to the top word of the stack marker, which also delimits the beginning of a new, fresh and unique local area for the procedure just called (Figure 2-8C).

The words in the stack marker preserve the state of the machine at the time of the procedure call. These words contain the following information, (shown in order ascending toward the Q-address):

Word	Contents
Q-3	Current contents of the Index (X) register.
Q-2	The return address for the <i>code</i> segment, denoted by P+1 (relative to the PB register).
Q-1	The current contents of the Status register (which includes the number of the code segment containing the calling procedure).
Q-0	The delta-Q value, which is the number of words between the new and previous Q-locations, enabling the later re-setting of Q to its old value.

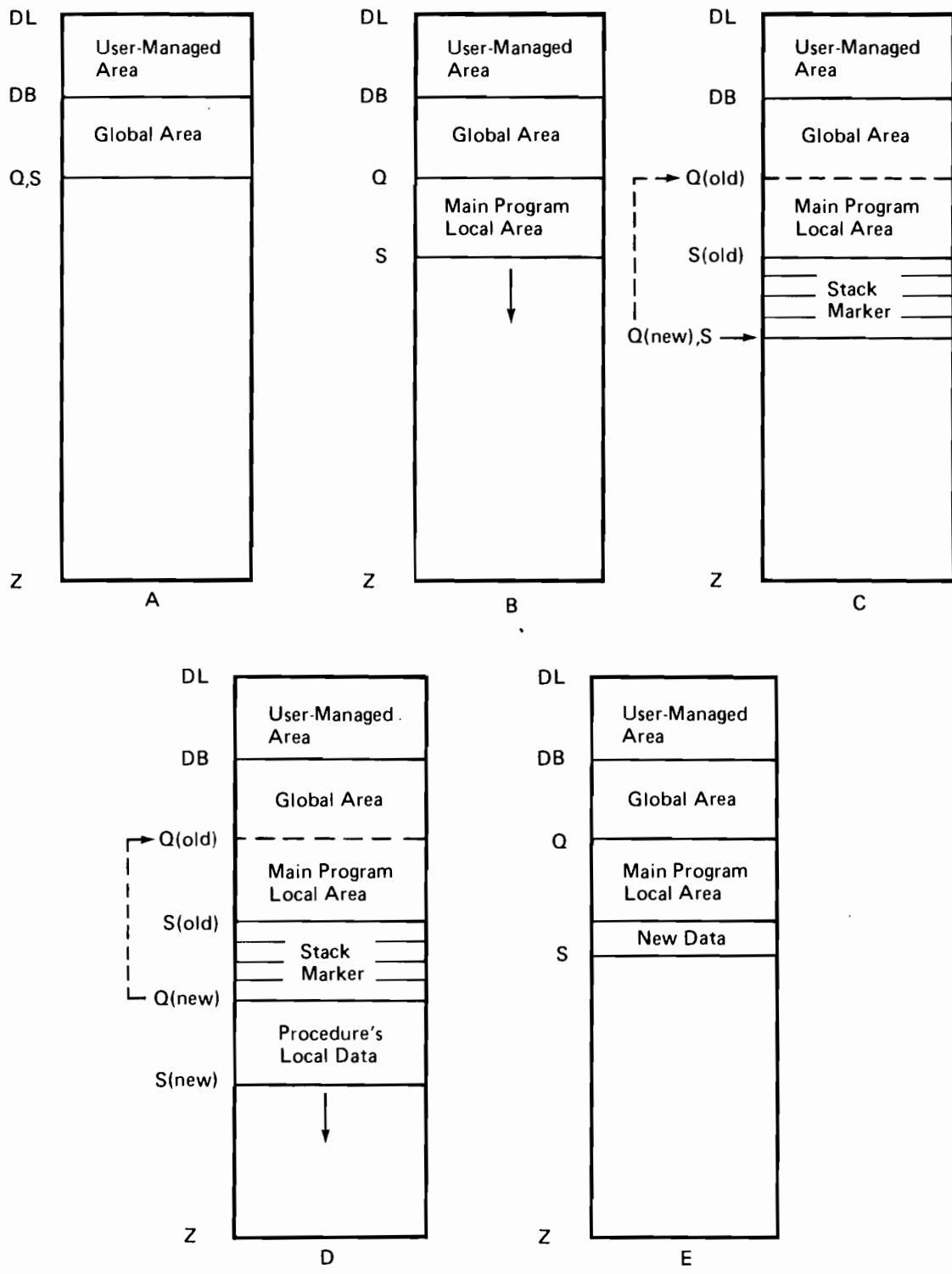


Figure 2-8. Stack Operation

As data is added to the stack during execution of the new procedure, the S-pointer moves away from the new Q-pointer, reflecting the latest data added (Figure 2-8D). When the procedure exits back to the main program, the new local data area is deleted from the stack, the stack marker is used to restore the Q-pointer to its previous setting, any value returned by the procedure is left at the new top of the stack, and the S-pointer is set to indicate the new top-of-stack (Figure 2-8E). This results in a “clean” stack from which temporary data local to the called procedure is eliminated because it is no longer needed.

Whenever a procedure is called, the Q and S registers are manipulated in this manner. The Q register changes with each procedure call and exit; the S register may change when an instruction references data. Thus, when a process executes a main program (outer block) that calls three procedures, there will be a maximum of four local areas (one for the main program and three for the procedures called by it) on the stack. Each procedure’s local area will be delimited at its base by its own stack marker. Within these stack markers, the delta-Q words will form a logical chain that links the present Q register setting back to its initial value.

MEMORY MANAGEMENT

The MPE/3000 virtual memory, which frees the user from the restrictions of a physical memory of constant size, consists of main memory and portions of disc. Both the dynamic swapping of segments from disc to main memory, and their dynamic relocation within main memory, are controlled by the memory management system through segment descriptors in the CST and DST. When additional memory is required during execution of a process, executed code segments are overlayed by incoming segments rather than actually being swapped out; this saves considerable memory overhead. (If needed later, re-entrant copies of these code segments can be brought in again from disc.) Because the content of a data segment changes continually, however, it is sometimes necessary to swap such a segment both to and from disc to preserve and maintain it until the process completes execution.

Since code can be shared between programs, both the need for multiple copies of programs or routines, and time devoted to swapping between disc and main memory, are reduced.

The user’s stack is not the only data area handled as a segment in MPE/3000. In addition, extra data segments (auxiliary to user stacks) and input/output buffers are also managed in this way by the memory management system.

Main-Memory Linkages

Main memory is organized into a sequence of variable-length, linked areas that facilitate the insertion or deletion of segments. The memory linkages contain the following information about each area:

- Availability (in use or available)
- Size
- Type (code or data)
- Table pointer (to the CST or DST entry relating to the area)

Those areas that are available are linked together through an Available-Space List; as areas in use are released, they are added to this list. Adjacent available areas are combined to form one contiguous area. Requests for memory can be implicit (through a CST presence-trap) or explicit (through calls from the system or user). To allocate memory, a space-allocation routine interrogates the Available-Space List to find the first area large enough to satisfy the request. If the search succeeds, all or a portion of that area is allocated. If the search fails, one or more overlays is performed to free an area of adequate length.

Assigned storage is linked according to the priority and frequency of its access. Frequently-accessed segments with high priority are located at or near the end of the Assigned-Storage List; infrequently-accessed segments with low priority are located near the head of this list. When an overlay is required, the list is searched, beginning with the head entry. Segments declared main-memory resident, of course, are not overlayed. If a data segment is selected for overlay, and it has been modified in memory, it is copied to a reserved overlay-storage area on disc. (Code segments are overlayed without being copied, since copies already exist on disc.) Each process is assigned such a reserved area when it is initialized; it may contain either code or data segments.

Main-Memory Use

When MPE/3000 is initialized, part of main-memory (usually in the low-address area) is pre-allocated to MPE/3000 or to the hardware for resident routines and tables; this area is not available for swapping. The remaining portion of main memory is established as the linked areas described above. During MPE/3000 initialization, some of this space is allocated for the CST, DST, and PCB tables, and the remainder is available for overlays (and so described in the Available-Space List).

USER JOB PROCESSING

The following paragraphs discuss how MPE/3000 concepts interrelate, from the standpoint of the user's program.

Batch Programs (Jobs)

To illustrate how a typical batch program (a job) is processed, consider a small FORTRAN source program punched on cards. The user arranges the deck so that the MPE/3000 commands, the FORTRAN program, and any input data are presented in the proper order, and submits this deck to the computer operator to run.

The operator enters the job through a card reader. MPE/3000 assigns the job a unique system job number, and schedules it for access to the central processor. When the central processor is available, a special system input/output process determines whether the attempted access is valid (by checking the user's identity and passwords) and requests the UCOP to create a job main process (JMP) that uses command interpreter code to run the job (Figure 2-9A, B). The JMP completes the job initialization and becomes the process responsible for handling interactions (commands) between the user and MPE/3000 (Figure 2-9B).

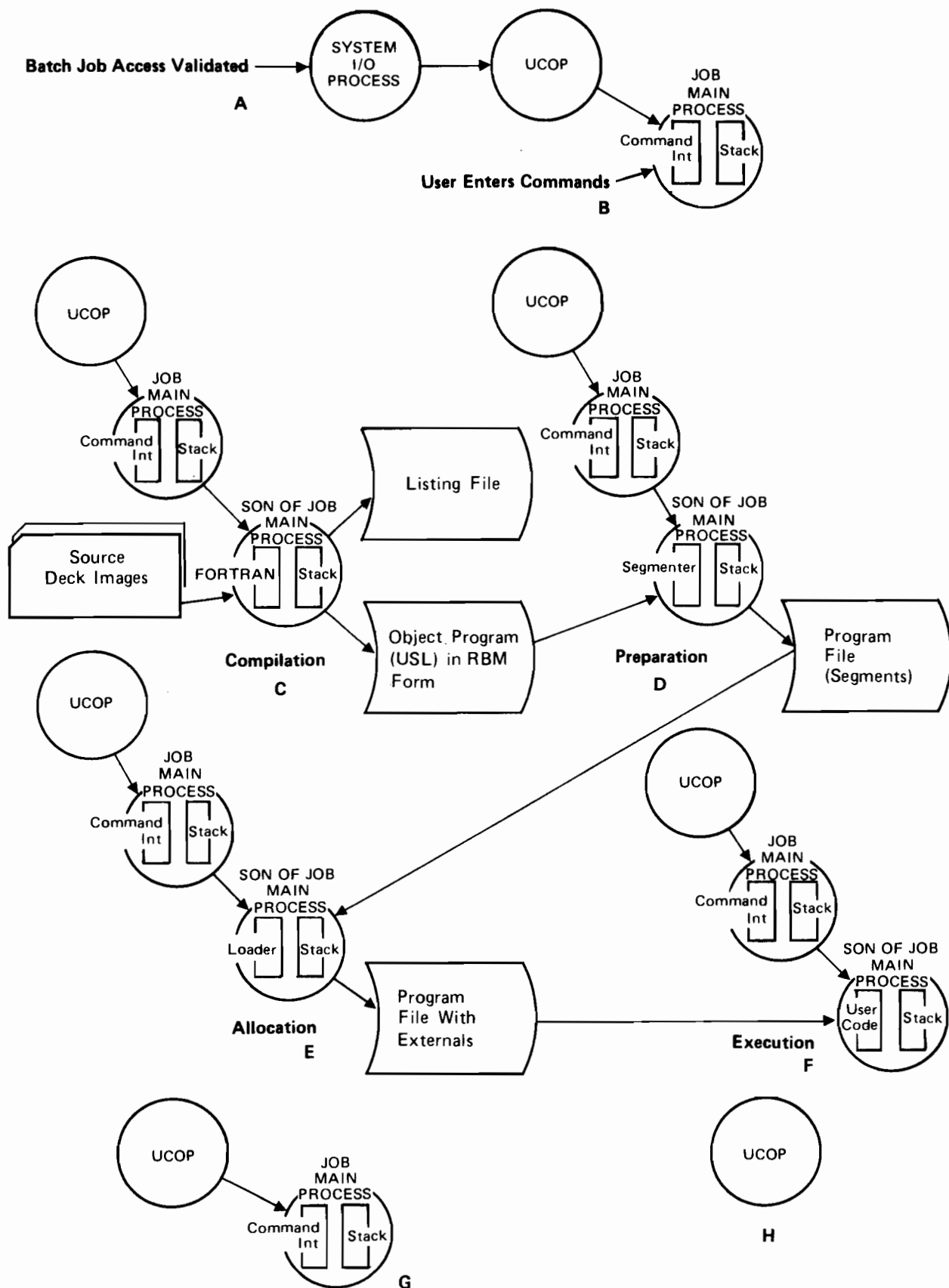


Figure 2-9. Batch Job History

When this process encounters a user request to compile, prepare, and execute the user's program, it creates another process that calls the FORTRAN/3000 compiler into virtual memory and executes the compiler code. (Figure 2-9C). This process translates the FORTRAN program from source to object code and organizes this code into RBM's stored as a USL on disc. In this process, the compiler interacts with the input/output and file management systems to read input and write output. Next, the main process creates a process that runs the segmenter subsystem, which reads the program from the USL and prepares it for execution by arranging it as a set of linked segments on a program file (Figure 2-9D). Then, the loader allocates the segments in virtual memory on disc, and satisfies external references to library routines (Figure 2-9E). The initial stack area is also defined at this time. Then, the code segment containing the entry point of the user's program, and the stack segment, are moved into main memory and execution of the user's process begins. (Figure 2-9F). Files needed by the job, and their associated devices are assigned as requested.

As processes are created by and for the job, they are entered into queues and scheduled for access to the central processor. Each process may create many descendent processes as the job progresses. Various processes share the central processor through interleaving, and in this way are run concurrently. The code segments belonging to each process obtain and operate on data from the stack associated with that process. As each process proceeds, segments are swapped into main memory as they are needed by the file management and memory management systems, which call the input/output system to handle this function. Data enters and leaves the stack area dynamically.

As the job progresses, the operator receives any pertinent error messages. He can suspend or terminate the job at any time. When any process (including the main process) is completed, it and its descendants are deleted from the system; all files opened by this process are closed (Figure 2-9G,H). Through the file management system, output is transmitted to a line printer and printed by an input/output process.

Time-Sharing Programs (Sessions)

As an example of how an on-line time-sharing program is handled, consider a BASIC-language program run by a user at a remote terminal.

The user contacts the system by turning the terminal on (if the terminal is connected directly) or by dialing the system (if the terminal is connected by switched telephone lines). In response, a special system input/output process determines the validity of the access and requests the UCOP to create a session main process (SMP) that uses the command interpreter code. The SMP completes the session initialization and becomes the process responsible for MPE/3000 interactions (commands) between the user and the system. The user next enters a command to access the BASIC/3000 Interpreter. A process is now created that handles input in the BASIC language. (Only one copy of the BASIC/3000 Interpreter is needed in main memory no matter how many users are programming simultaneously in BASIC; however, each user accesses the interpreter through a separate process.)

Many independent user processes share the central processor through time-slicing. Some commands create processes that may, in turn, create descendant processes. When required by each process, data segments are swapped to and from main memory. The file management and memory management systems interact with the input/output system to accomplish this. The file management system is also invoked to allocate devices for files whenever the files are opened. The user can direct output to the terminal, or he can have it transmitted to a printer. The user can terminate the session at any time.

ACCOUNT/GROUP/USER ORGANIZATION

In MPE/3000, the basic unit to which resources (such as file-space and central-processor time) are assigned is the *account*. Associated with each account is a unique file domain, a set of *users* who can access MPE/3000 through this account, and a set of *groups* which partitions the account's file domain into sub-domains.

Each account is defined, modified, and deleted by commands issued by a user with the MPE/3000 system manager capability, who has ultimate control over access to the system and allocation of its resources. Each account is identified by a name. Optionally, a password can be associated with the account to validate a user's ability to access MPE/3000 under this account at log-on time. A maximum priority also is associated with the account; this designates the highest priority at which any process run under this account can be scheduled. To maintain an account, the user acting as system manager grants a user the account-manager capability. This account-managing user may in turn, assign the same capability to other users in his account.

The users and groups associated with each account are defined by commands issued by the account managing user. Each user is identified by a name (unique to this account) and optional log-on password. He is assigned a maximum allowable priority for his processes, which cannot exceed the priority allowed to his account. Each group is identified by a name unique within its account, and optionally, by a password used to validate access to the group and its files at log-on time. (In the current MPE/3000 release, once a user logs on, he can access any file in the system if he knows the name of the file, and the group and account to which it belongs. In future releases, however, a security system will be implemented so that groups can serve to organize files into private subdomains subject to various access restrictions.)

Any MPE/3000 installation can contain several accounts; each account can have several users and groups associated with it; each group can possess several files (see Figure 2-10) which constitute a subset of the file domain. When the user logs on, he specifies the account, user, and group names (and, if required, the account, user, and group passwords).

Each user can be associated with a *home group* by the user managing his account. If the user does not specify a group when he logs on, he is given the home group by default.

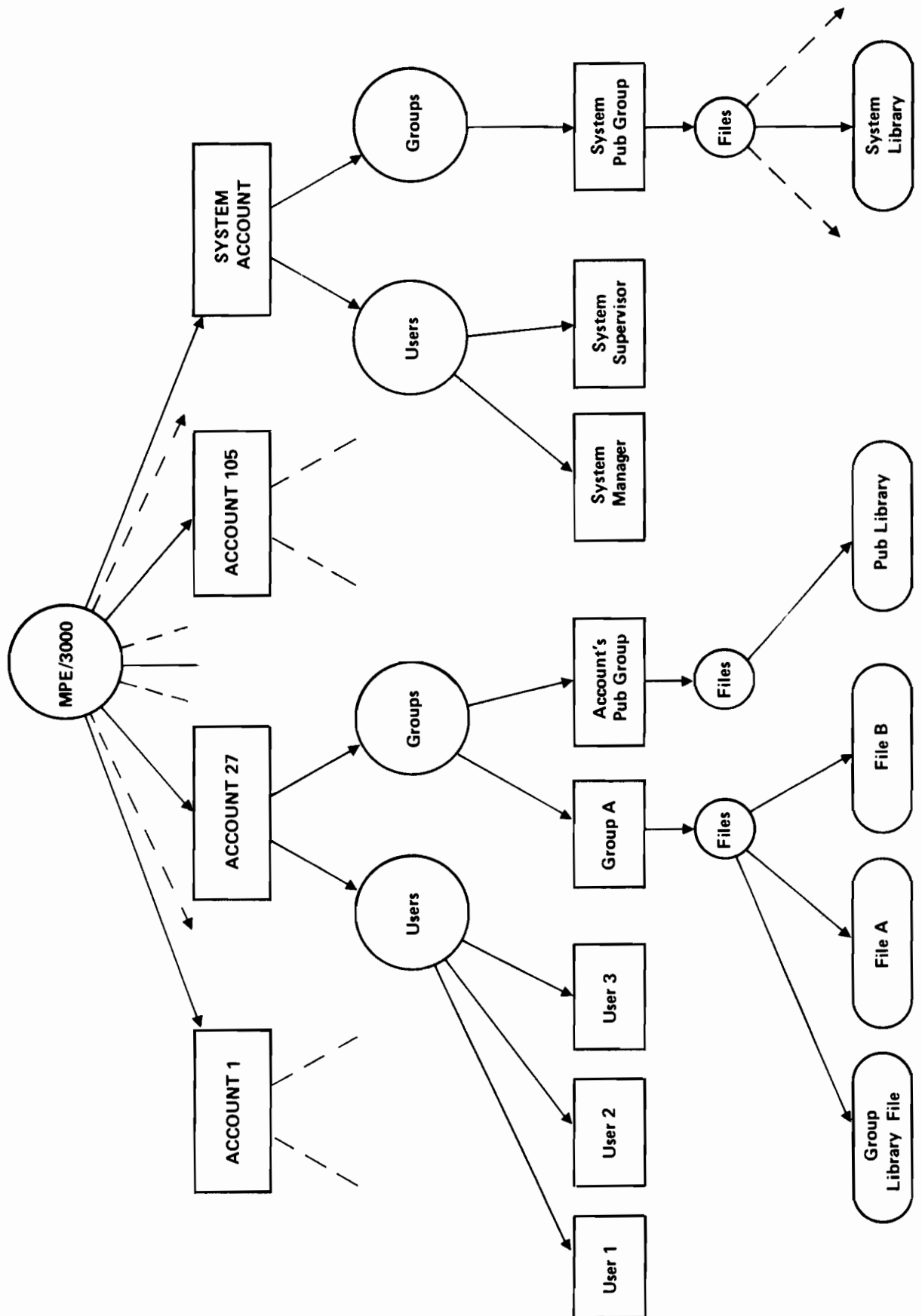


Figure 2-10. Account/User/Group Organization

CAPABILITY SETS

Most HP 3000 computers are used by a large variety of programmers, ranging from those who want to run simple applications programs in BASIC to system programmers who are actually modifying MPE/3000. To protect the system and its users in general, users with system and account manager capabilities can limit access to special system capabilities only to those who fully understand their correct use. This is done through capability sets. Specifically, when a system-manager user creates an account, he defines for it a capability set that determines whether or not users communicating with MPE/3000 through this account can request certain functions. When an account-manager user defines the users of his account, he associates with each user an individual capability set that may allow the user some or all of the general account capabilities. Each capability set contains three types of attributes: user, file-access, and capability-class. A fourth attribute, the local attribute, may also be defined. The combination of these attributes determines the set of commands and intrinsics available to the user. This division of commands and intrinsics greatly simplifies use of the system from the standpoint of each individual user — it defines the extent to which he must understand and inter-relate with MPE/3000, and permits a user to ignore aspects of MPE/3000 that do not apply to him.

Capability sets are also defined for groups by the account-manager user. Group capability sets contain only one type of attribute — the capability-class attributes. The capability set for a group may allow that group some or all of the capability-class attributes defined for the account to which the group belongs. The group capabilities relate to the user's capabilities as noted at the end of this section under *Program Capability Sets*.

As noted at the end of this section, capability-class attributes are also associated with each program on a program file, passed as parameters (in the command that prepares the program) to the MPE/3000 Segmenter.



User Attributes

The user attributes designate the general level at which the user interfaces with MPE/3000. These attributes can be assigned in any combination.

SYSTEM MANAGER ATTRIBUTE. Grants the user the capability to manage the overall system and the accounts within it. It gives him all the capabilities of a standard user (described below) plus any others he assigns himself. The first user with the system manager attribute is designated on the system tape furnished the customer. He, in turn, can designate other users having the same or different capabilities.

ACCOUNT MANAGER ATTRIBUTE. Allows the user to manage all users and groups within an account. The first manager for each account is designated by a user with the system manager attribute when the account is created. A user with the account-manager attribute, in turn, can assign this attribute to other users in his account.

SYSTEM SUPERVISOR ATTRIBUTE. Allows the user to have day-to-day external control of the system. It gives him total control over his programs and allows him to enter commands from any input device to manage scheduling subqueues, alter the system configuration, permanently allocate code in virtual memory, and display various items of system information. This attribute may be assigned by a user with the system or account manager attribute (if that user himself also has the System Supervisor Attribute).

STANDARD USER ATTRIBUTE. Typically identifies a programmer running jobs or sessions. This attribute is assigned by users with the account-manager attribute.

ACCOUNT LIBRARIAN ATTRIBUTE. Can be assigned to grant a user special file-access capabilities for maintenance of files within his account. (These file-access capabilities will be defined in a future MPE/3000 release.) This attribute is assigned by users with the account manager attribute.

GROUP LIBRARIAN ATTRIBUTE. Similar to the account librarian attribute, but limits the special file-access capabilities of the user to his home group. This attribute is assigned by users with the account manager attribute.

DIAGNOSTICIAN ATTRIBUTE. Permits the user to run diagnostic programs for on-line check-out of hardware under the System Diagnostic Monitor (SDM/3000). This attribute is assigned by users with the account manager attribute.

File Access Attributes

The file-access attributes determine whether the user has the capability to

- Save user files permanently
- Access card-readers, line printers, and other non-sharable input/output devices (other than the standard job/session input and listing devices which are available to all users)

These attributes can be assigned in any combination.

Capability-Class Attributes

These attributes define the general MPE/3000 resources available to a user. They allow the user to

- Access MPE/3000 in an interactive on-line mode
- Access MPE/3000 in local, batch processing mode
- Create, handle, and delete processes directly
- Manage data segments (by creating extra data segments)
- Have exclusive use of more than one system resource simultaneously
- Operate in privileged mode (which permits a user to access *all* MPE/3000 resources, including uncallable intrinsics)
- Run a process in the AS subqueue
- Run a process in the BS subqueue
- Access non-standard subqueues

Most users have *only* these MPE/3000 Standard Capabilities; all users must have one of these capabilities *at least*. (Console operators, who are not classified as “users” do not require any of these capabilities.)

MPE/3000 Optional Capabilities

Either the interactive or batch-access attribute is required to communicate with MPE/3000; most users (including users with System Manager and System Supervisor Capabilities) are assigned both as *standard capabilities*. The remaining capability-class attributes are *optional capabilities*. They are independent and can be assigned in any combination. The optional capabilities are discussed in *HP 3000 Multiprogramming Executive Operating System (03000-90005)*. Programmers should be aware that the more powerful the optional capability, the more hazardous its misuse is to the system. Thus, they should use optional capabilities with caution.

Local Attributes

The local attribute is a two-word quantity used only for special applications that require further unique classification of users. MPE/3000 does not reference or make use of this attribute in any way; rather, it is defined arbitrarily by system or account managers, and used by accounts or groups for whatever purpose they require.

Program Capability Sets

Capability sets are always associated with prepared programs as well as users.

Each time someone runs a program, MPE/3000 automatically assigns that program the user and file-access attributes of that user. But the capability-class attributes assigned to the program are those of the user who originally prepared the program (or a subset of those attributes designated by him), passed to the MPE/3000 Segmenter as parameters of the command that prepares the program. (When programs prepared from passed files or job temporary files (explained in Section V) are run, they are assigned the capability-class attributes of the user who runs them.)

The program's capability-class attributes must not exceed those defined for the group to which the program file belongs. If they do, the user will not be able to run the program when he attempts to do so.

Because the capability set is associated with the entire set of code segments being run (and hence with the process running them), all procedures, subprograms, and subroutines on those code segments have the same capability. For the same reason, a user need not have the same capabilities as the programs he runs.

SECTION III

Communicating with MPE/3000

To communicate with MPE/3000, the user issues commands and intrinsic calls.

Commands are requests issued to MPE/3000 to perform various broad functions external to the user's program. For example, they are used to initiate and terminate jobs and sessions, create and maintain files, compile and execute programs, call various utility subsystems, and obtain job status information. A user with System Manager Capability can use them to create or change accounts; a user with System Supervisor Capability uses them to run programs, display information, or manage the scheduling queue. Commands can be entered through any standard input device, typically the card reader (for jobs) or the terminal (for sessions). Each command is accepted by the MPE/3000 command interpreter, which passes it to the appropriate procedure for execution. Following this execution, control returns to the command interpreter.

Intrinsic calls are used to invoke MPE/3000 functions requested within a user's program, such as reading, writing on, and updating files, skipping forward and backward on files, or returning system table information to the user's program. Two intrinsics, CREATE (for creating a process) and GETPRIORITY (for rescheduling a process) allow a programmer who has System Supervisor Capability to specify certain parameter values that cannot be specified by other users calling these intrinsics. For this reason, CREATE and GETPRIORITY are discussed in this manual. All other intrinsics are described in *HP 3000 Multiprogramming Executive Operating System (03000-90005)*. In an SPL/3000 program, the user writes the intrinsic calls explicitly. In FORTRAN/3000 or BASIC/3000 programs, for most applications, the compiler generates any necessary intrinsic calls automatically—they are invisible to the user. At their option, however, FORTRAN/3000 and BASIC/3000 users can also directly call intrinsics as their needs require.

The commands and intrinsic calls discussed in this manual relate to initiating and terminating communication with MPE/3000 and requesting those functions available only to the users with the System Manager or System Supervisor Capabilities. These users also have access to all MPE/3000 Standard Capabilities, which are discussed in *HP 3000 Multiprogramming Executive Operating System (03000-90005)*. Those capabilities include compiling and preparing programs, accessing files, managing libraries, and various utility functions.

COMMANDS

Each command entered by the user, whether in a batch job or time-sharing session, consists of

- A colon (used as a command identifier)
- A command name
- A parameter list (in most cases)

The end of each command is delimited by the end of the record on which it appears—for example, a *carriage return* for terminal input or the end of the card on which it is punched for card input. But, if the last non-blank character of the record is a *continuation character*, as defined later in this section, the end-of-record does not terminate the command. (Users running programs in batch mode should bear in mind that all 80 columns on each card image are scanned by MPE/3000, and thus no characters are ignored.)

The *colon* identifies a statement as an MPE/3000 command. In a batch job, the user begins each MPE/3000 command with this colon in column 1 of the source card (or card image). In a time-sharing session, however, MPE/3000 prints the colon on the terminal whenever it is ready to accept a command; the user responds by entering the command after the colon. (Interactive subsystems of MPE/3000 also use unique prompt characters; in a session, the prompt character output tells the user that a subsystem is ready, and which subsystem it is.)

The *command name* requests a specific operation, and appears immediately after the colon. Imbedded blanks are not permitted within the command name. The end of the command name is delimited by any non-alphabetic character, normally a blank.

The *parameter list* contains one or more parameters that denote operands for the command. It is required in some commands but optional or prohibited in others. Parameter lists can include positional and/or keyword parameters. Within the parameter list, delimiters (commas, semi-colons, equal signs, or other punctuation marks) are used to separate parameters or parameter groups from each other, as described below.

Normally, the parameter list itself is separated from the command name by one or more blanks. However, when the first optional parameter in a positional list (as defined below) is omitted, the command name can be followed immediately by any other delimiter. (At the user's option, he can include one or more blanks between the command name and this delimiter.) Any delimiter can be optionally surrounded by any number of blanks, permitting a free and flexible command format.

EXAMPLE:

The following command (RUN) is preceded by a colon and includes a parameter list with two parameters:

:RUN PROG, ENTRYX

Both decimal and octal numbers are permitted as command parameters. Octal numbers, however, are always preceded by a percent sign (%).

Positional Parameters

With positional parameters, the meaning of the parameter is designated by its position in the list. For example, in the MPE/3000 command to compile a FORTRAN/3000 program, the parameter specifying an input file always precedes the one that specifies an output file. Positional parameters are mutually-separated by commas. The omission of an optional positional parameter from within a list is indicated by two adjacent commas; the omission of a positional parameter that would otherwise immediately follow a command name is indicated by a comma as the first character in the parameter list. When parameters are omitted from the end of a list, however, no adjacent delimiters need be included to signify this — the terminating *return* or end-of-card is sufficient.

EXAMPLE:

The first command below has its first parameter omitted; the second command has its second (embedded) parameter omitted; the third command has its last two (trailing) parameters omitted; the fourth command has all parameters omitted.

```
:FORTRAN ,USFL,LISTFL,MFL,NFL  
:FORTRAN SOURCEFL,,LISTFL,MFL,NFL  
:FORTRAN SOURCEFL,USFL,LISTFL  
:FORTRAN
```

A further example illustrates the relationship between the positions of parameters in a list and their meanings:

EXAMPLE:

In the following command (:FORTRAN), three positional parameters appear: INP refers to an input source file, OUT indicates an object (USL) output file, and LST indicates the listing output file. For the :FORTRAN command, these three fields always have the same meanings. (Note that the second delimiting comma in the parameter list is followed by an optional blank. In future examples, for clarity, delimiters will always be followed by blanks.)

```
:FORTRAN INP,OUT, LST
```

Keyword Parameters

When a parameter list is so long that use of positional parameters becomes difficult, keyword parameter groups are often used. The meaning of such a group is independent of its position in the list. A keyword parameter group is designated by a keyword that denotes its meaning, and optionally an equal sign and one or more parameters. (Each keyword group is preceded by a semi-colon. When more than one parameter appears in a group, they are usually separated from each other by commas. Delimiting semi-colons and commas can be optionally preceded or followed by blanks.) With respect to each other, keyword groups can appear in any order. When keyword and positional parameters both appear in a list, however, the positional parameters always precede the keyword groups; when this occurs, and *trailing* parameters are omitted from

the positional group, their omission need *not* be noted by adjacent delimiters; instead, the occurrence of the first keyword indicates this omission.

EXAMPLE:

In this example, DL and LIB designate keyword parameter groups. The value 500 and the letter G are sub-parameters of these groups:

```
:RUN PNAME, EPNT; DL=500; LIB=G
```

Continuation Characters

When the length of a command exceeds one record (source card or entry-line), the user enters an ampersand (&) as the last non-blank character of this record and continues the command on the next record. In this case, the next record must begin with a colon (entered by the user in batch processing, but prompted by the terminal in time-sharing). Optionally, blanks can be embedded between the colon that begins a continuation record, and the rest of the information on that record. Commands can be continued up to 256 characters, including prompting colons and continuation ampersands.

EXAMPLE:

The following command image contains a continuation character at the end of the first line:

```
:RUN PROGB; NOPRIV; LMAP; STACK=500; PARM=5; &  
: DL=600; LIB=G
```

In continuing a command onto another line, the user cannot divide a primary command element (a command name, keyword, or positional or keyword parameter) — no primary element is allowed to span more than one line.

MPE/3000 does not begin interpretation of a command until the last record of the command is read. During interpretation, all records within the command are concatenated, and all prompt characters and continuation ampersands are replaced by blanks.

Command Example Format

To help clarify the examples that appear throughout this manual, command information output by a printer during a batch job, or by a terminal during a time-sharing session, is *underlined*. Input information is not underlined. (In cases where differences in output occur, *terminal* output is assumed unless otherwise noted.) For both input and output, literal information that always appears exactly as shown is designated by CAPITAL letters and special characters; on the other hand, symbols that represent variable information are indicated by *lower case italics*.

Optional information is indicated by surrounding [brackets]. (However, the user does not enter these brackets as part of the command.) { Braces } indicate that one of the items included is required and must be entered by the user.

EXAMPLE:

The colon is output by the terminal during interactive sessions (as shown by the underline); :SHOWQ is a literal command name entered by the user, (as indicated by capital letters); and subqueueuname is a variable parameter input by the user (as shown by lower-case letters). The subqueueuname parameter is optional, (as indicated by brackets).

:SHOWQ [subqueueuname]

When more than one item is enclosed vertically in brackets, this indicates that exactly one of these items may be specified.

EXAMPLE:

:LISTGROUP [@groupname] [;listfile]

When more than one item is enclosed vertically in braces, this means that exactly one item must be specified.

If items are shown within vertical and disjoint brackets, this signifies that they are all optional and that those specified by the user can appear in any order.

EXAMPLE:

:RUN progfile [,entrypoint]
[;NOPRIV]
[;LMAP]
[;MAXDATA = segsize]
.
.
.

Command Errors

All MPE/3000 commands issued, and any system messages for the user, are copied to the job/session listing device.

When MPE/3000 detects an error in a command, it suppresses execution of that command and prints an error message. If this occurs during a batch job, (and no :CONTINUE command precedes the erroneous command, as discussed later) all information between the erroneous command and the end of the job is ignored, and the job is aborted. If an error occurs during a time-sharing session, control returns to the user; he may then re-enter this command correctly, or enter any other command he desires.

INTRINSIC CALLS

Intrinsic calls are issued primarily by users programming in SPL/3000, although facilities are also available in FORTRAN/3000 and BASIC/3000 for this purpose. The rules governing intrinsic calls from SPL/3000 programs are summarized below. The rules regarding the invoking of intrinsics from programs written in other languages are presented in the reference manuals on those languages.

Before an intrinsic can be called from an SPL/3000 program, it must be declared within the declaration portion of the program, following all data declarations, like any other SPL/3000 procedure. This can be done by writing the standard PROCEDURE declaration. Normally, such a declaration would contain both the procedure head and the procedure body (code). But, since an intrinsic is an *external* procedure, the user follows the normal SPL/3000 conventions for such procedures by writing only the intrinsic *head* and including within it the OPTION EXTERNAL notation. (EXTERNAL signifies that the body code will be supplied from a library and linked to the user's program during program allocation.) In this manual, the complete head format for each intrinsic is presented with the discussion of that intrinsic. (Note that in SPL/3000, each statement is delimited by semi-colons.)

EXAMPLE:

The format for the head of the GETPRIORITY intrinsic is presented in Section VII. Using this format, the user could write a declaration that will enable him to call the GETPRIORITY intrinsic later in his program. The declaration could appear as follows. Note that the user must include the OPTION EXTERNAL notation.



```
PROCEDURE GETPRIORITY (FN,PC,RNK);  
VALUE FN,PC,RNK;  
LOGICAL PC;  
INTEGER FN,RNK;  
OPTION EXTERNAL;
```

Because some intrinsic heads are rather long, the SPL/3000 programmer can save time and effort by using an alternative method of declaring intrinsics; the INTRINSIC declaration. This is written in the following format

INTRINSIC intrinsicname, intrinsicname, . . . , intrinsicname;

In the *intrinsicname* list, the user names all intrinsics he intends to call within his program. (When more than one is named, the names are separated by commas.) He then need not write the heads for these intrinsics.

EXAMPLE:

To use the INTRINSIC declaration to declare the FOPEN, FREAD, FWRITE, and FCLOSE intrinsics, the user could write:

INTRINSIC FOPEN,FREAD,FWRITE,FCLOSE;

Regardless of whether the programmer declares an intrinsic with a PROCEDURE declaration or an INTRINSIC declaration, he must know the formal head format for each intrinsic, since this tells him the number and type of parameters that are used in calling that intrinsic.

The user calls an intrinsic in his program exactly as he calls any SPL/3000 procedure: he writes the intrinsic name and a parameter list, enclosed in parentheses. These elements follow the positional format shown in the intrinsic head; parameters are separated from each other by commas.

EXAMPLE:

A call to the GETPRIORITY intrinsic, shown earlier, could be written as

GETPRIORITY (FWORD,PRWORD,RVAL);

If the OPTION VARIABLE notation appears in the intrinsic head, some of the intrinsic parameters are optional. In this manual, these optional parameters are indicated as bold face in the intrinsic head formats.

Since all intrinsic parameters are positional, the user indicates a missing parameter within a list by following the previous delimiting comma with another comma by itself.

EXAMPLE:

The second parameter is missing:

CREATE (PNAME,,PROG);

If the first parameter is omitted from a list, this is indicated by following the left parenthesis with a comma. If one or more parameters are omitted from the end of a list, this is indicated by simply writing the terminating right parenthesis after the last parameter included.

NOTE: In some intrinsic calls, input parameters are passed to the intrinsic as words whose individual bits or fields of bits signify certain functions or options. In cases where some of the bits within a word are described in this manual as "not used by MPE/3000," the user is advised to set such bits to zero. This will help ensure the compatibility of his current programs with future releases of MPE/3000.

In cases where output parameters are passed by an intrinsic to words referenced by a users' program, bits within such words that are described as "not used by MPE/3000" are set as noted in the discussion of the particular parameter.

Intrinsic Example Format

In this manual, the complete intrinsic declaration head is shown with the discussion of each intrinsic; included within this is the intrinsic call format, distinguished from the remainder of the head by a box. This format appears as follows:

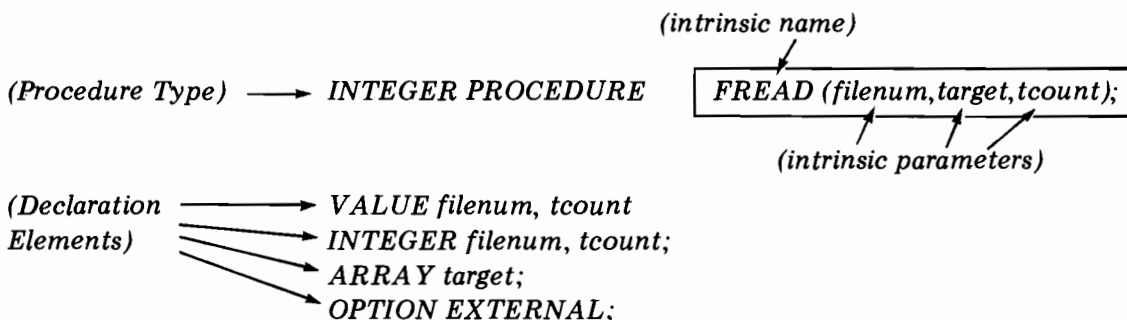
procedure type **PROCEDURE** *intrinsic name (parameter list)*
declaration elements

The *procedure type* applies only to *type intrinsics*—those that return a value to the user's program in response to the intrinsic call. This describes the type of data returned, such as integer, real, or byte values. The data is returned through the intrinsic name. Additionally, type intrinsics can be directly related to functions called in FORTRAN/3000 programs, as noted in the manual *HP 3000 FORTRAN (03000-90007)*.

The *declaration elements* describe each parameter (variable, pointer, label, and array) in the parameter list, and its characteristics. If the VALUE notation is present in the declaration, either a numeric value (expression) or a symbolic identifier may be specified for the parameters indicated by this notation. If VALUE is not present, only symbolic identifiers may be used for parameters.

EXAMPLE:

This is the intrinsic description format for the *FREAD* intrinsic available to all users with MPE/3000 Standard Capabilities.



Intrinsic Call Errors

Some intrinsics alter the *condition code*, always made available to the user's program through the Status register. Since the contents of this register change continually, the user should check this register for condition codes immediately upon return from an intrinsic. A condition code is always one of the following, and has the general meaning indicated. The specific meaning, of course, depends upon the intrinsic called.

Condition Code	General Meaning
CCE	Condition code is zero. This generally indicates that the user's request was granted.
CCG	Condition code is greater than zero. A special condition occurred but may not have affected the execution of the user's request. (For example, the request was executed, but default values were assumed as intrinsic call parameters.)
CCL	Condition code is less than zero. The user's request was <i>not</i> granted, but the error condition may be recoverable. Beyond this condition code, some intrinsics return further error information to the user's program through their return values.

Two types of errors may be encountered when an intrinsic is executed. The first, denoted by the CCG or CCL condition codes, is generally recoverable and is known as a *condition-code error*. The second type is an *abort error*, which occurs when a calling program passes illegal parameters to an intrinsic, or does not have the capability class demanded by the intrinsic. The user's program may recover from this error or take some other appropriate action if an appropriate error-trap procedure has been defined (as discussed in *Multiprogramming Executive Operating System*). Otherwise, his program terminates, and an abort-error message appears on his output device. If the program was entered in a batch job, MPE/3000 removes the job from the system (unless a :CONTINUE command, defined later in this section, precedes the error); if it was entered in a time-sharing session, MPE/3000 returns control to the user at the terminal.

NOTE: *Whenever a callable intrinsic is invoked by a process running in either the non-privileged mode, or the privileged mode with the DB register pointing to the DB area in the user's stack, a bounds check takes place to ensure that all parameters in the intrinsic call reference addresses that lie between the DL and S addresses in the stack (prior to the intrinsic call); if an address outside of these boundaries is referenced, an abort-error occurs.*

When a callable intrinsic is invoked by a process running in the privileged mode, and the DB register points to a data segment other than the user's stack segment, the results depend on the particular intrinsic. Most intrinsics (including those discussed in this manual) immediately abort in this case. Others are allowed to execute

following a bounds-check that ensures that all parameters in the intrinsic call reference addresses that lie within the data segment; any boundary-violation results in an abort-error.

BATCH JOBS

The user can initiate and terminate batch jobs through any device that accepts serial input (such as a card reader, tape unit, or terminal used non-interactively) located at the computer site, or through a terminal used non-interactively and located remotely, as discussed below.

Initiating Batch Jobs

A batch job consists of a set of cards (or card images) that begins with a :JOB command, contains additional MPE/3000 commands, optional user programs and data, and terminates with an :EOJ command. (Typically, a user performing System Manager functions inputs a job consisting only of MPE/3000 commands.) The :JOB command initiates the job by establishing contact with MPE/3000. If it accepts this command, MPE/3000 begins job processing. When processing begins, commands are sequentially accepted from the job until an :EOJ command is encountered or until the job is halted by one of the abnormal events described under "Premature Job or Session Termination."

The user writes the :JOB command in the following format. (The keyword parameters, of course, can be specified in any order.)

```
:JOB [jobname,] username[/upass].acctname[/apass] [,groupname[/gpass]]  
    [;TERM = termtype]  
    [;PRI = executionpriority]  
    [;OUTCLASS = outputclass]
```

The parameters have the meanings noted below.

NOTE: In MPE/3000, names (such as the jobname, username, upass, acctname, apass, groupname, and gpass parameters noted below) consist of from one to eight alphanumeric characters, beginning with a letter, unless otherwise specifically indicated.

<i>jobname</i>	An arbitrary name used in conjunction with the <i>username</i> and <i>acctname</i> parameters to reference this job in other commands. If omitted, a <i>null jobname</i> is assigned. (Optional parameter.)
<i>username</i>	The user's name, as established in MPE/3000 by the user with account manager capability. This name is unique within the account. (Required parameter.)

<i>upass</i>	The user's password. (Required if the user has been assigned a password by the user with account manager capability.)
<i>acctname</i>	The name of the user's account, as established by the user with system manager capability. Note that this parameter is preceded by a <i>period</i> as a delimiter. (Required parameter.)
<i>apass</i>	The account password. (Required if the account has been assigned a password by the user with system manager capability.)
<i>groupname</i>	The name of the group (established by the user with account manager capability) that the user wishes to use, during his job, for his local file domain. If omitted, MPE/3000 assigns the user's home group. (Optional parameter for users with a home group; Required parameter for users without a home group.)
<i>gpass</i>	The group password. When a user logs on under his home group, no password is needed. (Required parameter for some groups.)
<i>termtype</i>	The type of terminal used for input.

(MPE/3000 uses the *termtype* parameter to determine device-dependent characteristics such as delay factors for carriage returns.) This parameter is indicated by one of the numbers 0 through 8, with the meanings shown below:

- 0 = ASR 33 EIA-compatible HP 2749B (10-characters per second (cps)).
- 0 = ASR 35 EIA-compatible (10 cps).
- 1 = ASR 37 EIA-compatible (15 cps).
- 2 = Univac DCT 500 (Type 8541-99, HP 2650A) (10, 15, 30 cps).
- 3 = Execuport
- 4 = HP 2600A or DATAPOINT 3300 (10-240 cps)
- 5 = Memorex 1240 (10, 30, 60 cps)
- 6 = GE Terminet 300 (10, 15, 30 cps)
- 7 = Selectric (IBM 2741) CALL 360 Correspondence Code
- 8 = Selectric (IBM 2741) (PTTC/EBCD Code) (Users at this terminal are directed to the first NOTE under the discussion "Initiating Sessions," later in this section.)

If the *termtype* parameter is omitted, and a terminal is used, a default of 0 is assigned. (Optional parameter for ASR 33 or ASR 35 terminals; Required parameter for all others.)



<i>executionpriority</i>	The priority (subqueue) desired for MPE/3000 command interpretation, and also the default priority for all programs within the job. For users with <i>standard capabilities</i> , this may be CS, DS, or ES, indicating the priorities (subqueues) described in Section II. For users with optional capabilities, this may be any of the priorities described later in this manual (depending on the maximum subqueue priority permitted users of the account). If the priority specified exceeds that permitted for this user, the highest priority permitted him applies. If this parameter is omitted, DS is assigned by default. (Optional parameter.)
<i>outputclass</i>	A particular device (such as a specific line printer) to be used for listing output. This parameter is a device class name or a logical device number (as defined in <i>HP 3000 Multiprogramming Executive Operating System</i>), and may only be entered by users who have the <i>non-sharable input/output device</i> file-access attribute. If this parameter is omitted, the device assigned is the one defined (during system configuration) as the default listing device that always corresponds to the user's current input device. (Optional parameter.)

The *upass*, *apass*, and *gpas* parameters, when included, must be separated from their preceding parameters with a slash. (The slash should not be preceded nor followed by blanks.) When the job output is listed on a device other than the standard listing device (corresponding to the device over which the job is input), passwords are not printed on the job listing.

EXAMPLE:

This job is named ALPH22, and is entered under the user name RJOHNSON, account name ACCT1003, and the group GPA2. The group password MG03 is used. An execution priority of DS is requested. For all other parameters, default values apply.

:JOB ALPH22, RJOHNSON. ACCT1003, GPA2/MG03; PRI=DS

All MPE/3000 commands encountered during job processing are listed on the job listing device (typically, a *printer*). Acceptance of the job is indicated by information output in the following format, immediately after the listing of the :JOB command.

JOB NUMBER = #Jnnnnn
date, time

nnnnn Job number assigned by MPE/3000 to uniquely identify the job to the system. This may range from one to five digits long. The user can reference the job in subsequent commands by this number or by the job identification assigned through the :JOB command:

[jobname,] username . acctname

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

EXAMPLE:

If the job in the previous example had been assigned the job number 123, and had been submitted at 3:23 p.m. on January 31, 1972, the job output listing might appear as follows (with the beginning of the acceptance message shown here by an arrow):

:JOB ALPH22, RJOHNSON. ACCT1003, GPA2/M603; PRI=DS



JOB NUMBER = #J123
MON, JAN 31, 1972, 3:23 PM

Terminating Batch Jobs

The :EOJ command terminates the batch job. This command has no parameters and is written simply as

:EOJ

When this command is encountered, MPE/3000 acknowledges job termination by printing the following information on the job listing

date, time

END OF JOB

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

If no :EOJ command is included to terminate the current job, the next :JOB command will have one of the following effects:

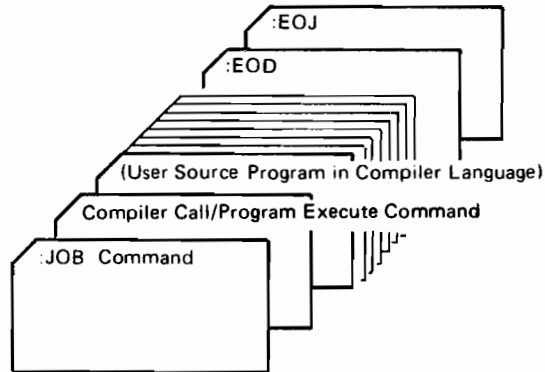
1. If read by the MPE/3000 command interpreter, it will terminate the current job and start a new one.
2. If read by a user's program, it will signal an end-of-file to the program but will be ignored by MPE/3000. (Because this can occur, the user should be sure that he terminates his job with an :EOJ command.)

Typical Job Structures

All batch jobs must begin with a :JOB command and terminate with an :EOJ command. Additionally, the end of each program input within a job should be indicated with an :EOD command, (written simply as :EOD). Beyond this, job structures can vary considerably from application to application.

EXAMPLE:

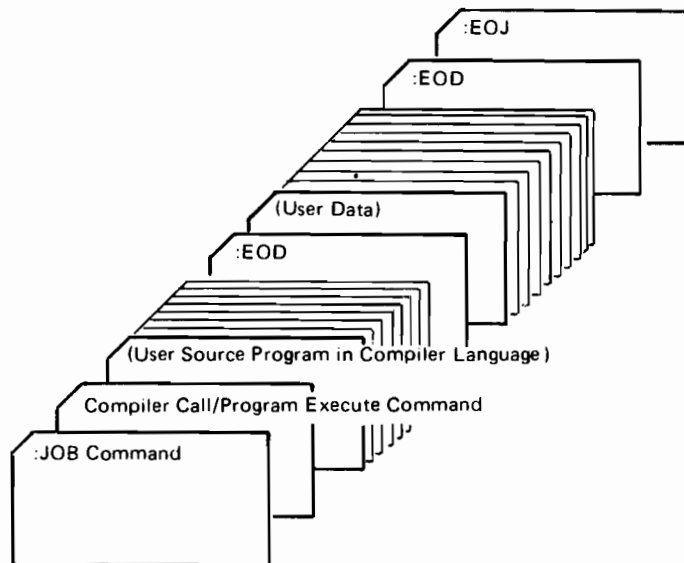
A job submitted through the card reader to compile and execute a program might appear as follows. (Note that an :EOD card denoting the end of the program should precede the :EOJ card.)



When data is included within a job, its end must also be indicated with an :EOD command. (Users writing language processors and other subsystems should be aware that MPE/3000, when reading input from the standard input stream, interprets *any* record beginning with a colon as the end-of-data, whether or not this record contains an :EOD command. The :EOD command is used as a data delimiter in general applications because it executes no *additional* functions.)

EXAMPLE:

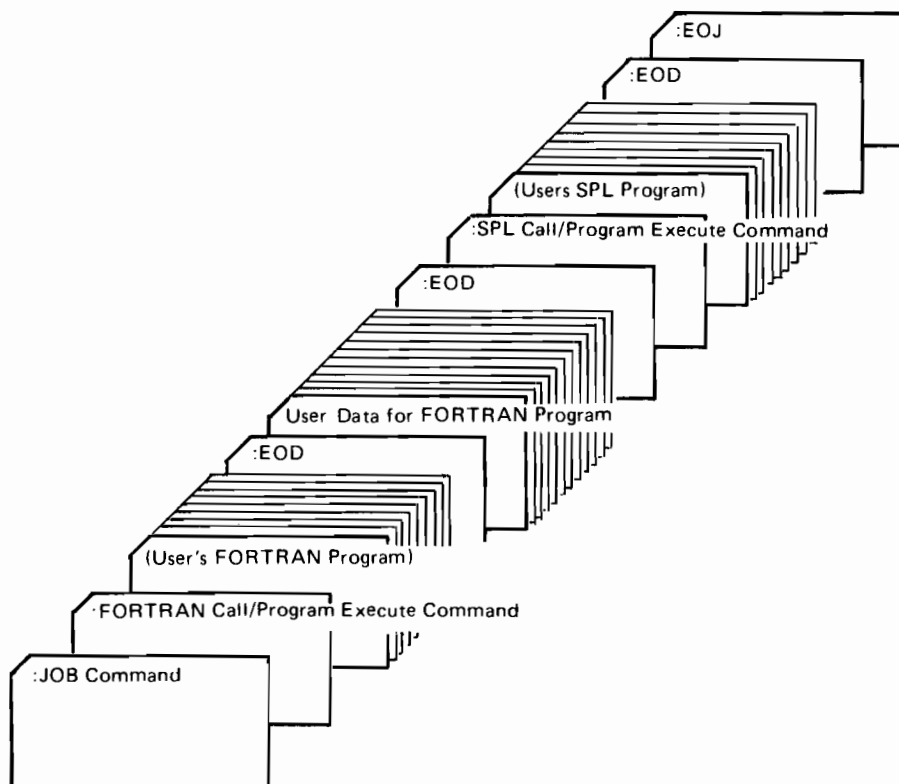
The job shown below includes data to be processed during program execution:



Of course, jobs with much more elaborate structures are possible. Such jobs may contain several user programs, input to different compilers, obtaining input data from various sources, and transmitting output data to several devices:

EXAMPLE:

The following job contains: a FORTRAN/3000 program to be compiled and executed; data for that program; and an SPL/3000 program to be compiled and executed, with data input from a disc file.



TIME-SHARING SESSIONS

The user initiates and terminates time-sharing sessions from an interactive terminal. (Non-interactive devices cannot be used for this purpose.)

Initiating Sessions

The user initiates a session by turning the terminal on, dialing MPE/3000 (if the terminal is connected over switched telephone lines), pressing the *return* key to generate a carriage return and produce the first prompt character (colon), and entering the :HELLO command described below. This logging-on establishes contact with MPE/3000 and allows the user to request whatever functions his capabilities permit. From this point on, the user is automatically prompted for each command (and thus does not type the colon). The session exists until the user issues a command to terminate it, or until one of the conditions described under "Premature Job or Session Termination" occurs.

NOTE: Users at IBM 2741 Selectric Terminals (PTTC/EBCD code) should note that

- 1. Because of the way in which input/output character mapping is handled with this terminal, the prompt character output will be a K rather than a colon.*
- 2. Following the initial prompt character, the user must always type H as the first character of his input; this enables MPE/3000 to determine (from this character's bit pattern) the type of Selectric being used.*

Users at IBM 2741 Selectric Terminals (CALL/360 or PTTC/EBCD code) should note that any CALL/360 or PTTC/EBCD character that does not have an equivalent ASCII character is ignored on input.

The user enters the :HELLO command in the following format

```
_:HELLO [sessionname,] username [/upass] .acctname [/apass] [,groupname [,gpass] ]  
      [;TERM = termtype]  
      [;PRI = executionpriority]
```

The parameters have the meanings noted below.

NOTE: The sessionname, username, upass, acctname, apass, groupname and gpass parameters are all names that can contain up to eight alphanumeric characters, beginning with a letter.

sessionname An arbitrary name used in conjunction with the *username* and *acctname* parameters to reference this session in other commands. If omitted, a null *sessionname* is assigned. (Optional parameter.)

<i>username</i> <i>upass</i> <i>acctname</i> <i>apass</i> <i>groupname</i> <i>gpass</i> <i>termtype</i> <i>executionpriority</i>	}	The same definitions noted under the corresponding parameters in the :JOB command, applying to <i>sessions</i> rather than jobs, and with this exception: the default value for the <i>executionpriority</i> parameter in the :HELLO command is CS.
---	---	---

The *upass*, *apass*, and *gpass* parameters, when included, must always be separated from their preceding parameters by a slash. (This slash should not be bounded by blanks.)

EXAMPLE:

In the following command, the user establishes a session named *INTER3*, under the user name *JONES*, account name *ACT20*, and *GROUP3*. The account password of *PASS* is also entered.

:HELLO INTER3, JONES. ACT20/PASS, GROUP3

When a user is entering a job through a terminal connected over switched telephone lines (a dial-up terminal), he must specifically log on within a period defined during system configuration. Otherwise, his terminal is disconnected and he must dial MPE/3000 again.

When entering a session through certain hard-copy full duplex terminals, the user can suppress the printing (echoing) of passwords at his terminal by temporarily placing the computer in half-duplex mode (by pressing the *ESC* and *;* keys). Where the terminal permits only half-duplex operation, the user can conceal the passwords simply by omitting them from the command; in this case, after the :HELLO command is echoed, MPE/3000 outputs a corresponding request for each required password and directs the carriage to skip two lines and position itself at the beginning of the second line, as shown:

$$\left\{ \begin{array}{l} \underline{USER} \\ \underline{ACCOUNT} \\ \underline{GROUP} \end{array} \right\} \quad \underline{PASSWORD?}$$

← (carriage return)

The user then enters the password. For sessions in which input and listing output are generated on the same device, MPE/3000 prints an eight-character mask before the user enters the password. When he enters the password, it is echoed on the input/listing device, but is printed on top of the mask to ensure privacy.

EXAMPLE:

Suppose that a group password of ORG was required, but not entered, in the command shown in the previous example. The user would be prompted for the password, and would enter it (over the mask) as follows:

```
:HELLO INTER3, JONES. ACT20/PASS, GROUP3  
GROUP PASSWORD?  
▶ XXXXXXXXXX
```

For sessions in which input and listing output are generated on separate devices, MPE/3000 omits the password entered.

Upon initiation of the session, the following information is transmitted to the terminal:

SESSION NUMBER = #Snnnnn
date, time

nnnnn Session number assigned by MPE/3000 to uniquely identify the session.
 This may range from one to five digits long.

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

EXAMPLE:

Suppose that the session in the previous example is accepted by MPE/3000 under the session number 512 at 12:37 p.m. on May 5, 1972. The following information would be output. (Remember, the mask is printed before the password is echoed.)

```
:HELLO INTER3, JONES. ACT20/PASS, GROUP3  
GROUP PASSWORD?  
▶ XXXXXXXXXX  
SESSION NUMBER = #S512  
  
FRI, MAY 5, 1972, 12:37 PM
```

If the user enters the `:HELLO` command incorrectly, the full sequence of log-on information through the *date* and *time* is still output. At this point, a request to re-enter the `:HELLO` command appears. (The erroneous element in the previous `:HELLO` command is not identified.)

On terminals connected to switched telephone lines (dial-up terminals), the user must access successfully MPE/3000 within a set duration, or his terminal will be disconnected. When disconnection occurs, the user must dial MPE/3000 again. (The duration allowed for logging-on is defined during system configuration.)

Interrupting Program Execution Within Sessions

The user can interrupt execution of a program at any point in a session by striking the *BREAK* key. This returns control to the MPE/3000 command interpreter, allowing the user to issue commands to abort the program (without disrupting the session), perform various file or utility operations, or resume the program.

To abort the program, the user enters this command:

`:ABORT`

To request an operation (such as creating or deleting a file), the user enters the command for that operation. File and utility commands are executed immediately. Some commands, however, require aborting of the user's program before they can be executed. When the user enters such a command during a program break, MPE/3000 prints:

`ABORT?`

The user responds by entering *yes* (to abort the program and execute the command) or *no* (to return control to the command interpreter).

To resume a program at the point where it was interrupted, the user enters:

`:RESUME`

A system power or line failure automatically results in permanent termination of sessions; the sessions must be re-initiated from the beginning.

Terminating Sessions

To terminate a session, the user enters the following command:

:BYE

MPE/3000 acknowledges termination by printing the following information:

date, time

END OF SESSION

date Current date (day-of-week, month and day, year)

time Current time (hours:minutes am/pm)

EXAMPLE:

Typical terminal output resulting from logging off:

```
:BYE  
THU, MAY 7, 1972, 12:56 PM  
END OF SESSION
```

If the user hangs-up the receiver at a dial-up terminal, an implicit :BYE command is issued and the session terminates.

Typical Session Structure

All sessions must begin with a :HELLO command. They typically terminate with a :BYE command. The end of data within a session must be indicated with the :EOD command.

If a user issues a second :HELLO command within his current session, this results in an implicit :BYE and :HELLO sequence—that is, it terminates the current session and starts another one.

The structure of sessions varies with the application. One example follows.

EXAMPLE:

:HELLO (Session, user, and account identification.)

:SESSION NUMBER = #Snnn

(date) (time)

·
·
·

: (MPE/3000 Command to call BASIC Interpreter.)

·
·
·

(User Program in BASIC.)

(User Command to exit from BASIC Interpreter.)

:BYE

(date) (time)

END OF SESSION

READING DATA FROM OUTSIDE STANDARD INPUT STREAM

Users can designate, for a job or session, input consisting only of data to be read from a non-sharable, auto-recognizing device that is *not* the standard input device for that job or session. (An auto-recognizing device is one that automatically accepts the :JOB, :HELLO, or :DATA commands.) This designation is typically made to read a deck from a card reader while in session mode. Indeed, it is required for every data deck not imbedded in the standard input stream (\$STDIN).

To designate the data for the job or session, the user must precede the data with the :DATA command and terminate it with the :EOD command. The :DATA command implicitly initiates communication with MPE/3000 and thus is the only command that is not entered within a formally-initiated job or session. (Note that the :DATA command does not apply to labeled data on tape, data from non-auto recognizing devices, or data imbedded in the job or session input stream.)

The :DATA command format is

:DATA [jobname,] username[/upass] .acctname[/apass] [;filename]

(A complete job or session identification.)	↗	jobname	The name of the job or session that is to read the data. (Optional parameter.)
	→	username	The user's name, as established in MPE/3000 by the user with account manager capability. (Required parameter.)
		upass	The user password. (Required if user has a password.)
	↘	acctname	The name of the account, as established by the user with system manager capability. (Required parameter.)
		apass	The account password. (Required if account has a password.)

filename An additional qualifying name for the data that can be used by the job or session to access the data. It may be used, for example, to distinguish two separate data decks from different card readers read by the same program. If *filename* is omitted, no such distinguishing name is assigned. (Optional parameter.)

The data can *only* be read by a job or session that has the same identity:

[jobname,] username.acctname

The data exists in the system until read by the job/session or purged by the Console Operator.

The *jobname*, *username*, *acctname*, and *filename* parameters are all names that can contain up to eight alphanumeric characters, beginning with a letter.

EXAMPLE:

To designate a card data file for a session identified as *JOBSP*, *BLACK.ACTSP*, the user submits the following *:DATA* command on a card preceding the data deck:

```
► :DATA JOBSP, BLACK. ACTSP; FILESP
.
.
.
(USER DATA)
.
.
.
: EOD
```

The session can access the data in the card file by specifying the card reader (either by device class name or logical device number), and the filename *FILESP*.

PREMATURE JOB OR SESSION TERMINATION

A *job* is terminated before an :EOJ command is encountered, if any of the following events occur:

1. The Console Operator issues a command to terminate the job.
2. MPE/3000 aborts the job because an error occurred in the interpretation or execution of an MPE/3000 command.
3. MPE/3000 aborts the job because a subsystem error was encountered.
4. A program within the job is aborted.
5. A second :JOB command or a :DATA command is encountered. It will be executed, resulting in job termination (unless it is read as part of a program's data input from a file/device, in which case it signals an end-of-file to the program but is ignored by MPE/3000).

Events 1 through 4 above place the job in an *error state* (by setting the error state flag). Normally, this results in job termination. But, if the user anticipates an error as a result of a specific command, he can override premature termination by using the :CONTINUE command before that command. The :CONTINUE command format is

:CONTINUE

The :CONTINUE command permits the job to continue even though the following command results in an error (in which case the error state indication is re-set).

EXAMPLE:

Suppose the user submits a job to execute three programs (with the :RUN command), but anticipates a probable terminating error in the first program. To continue the job and execute the second and third programs, he uses the :CONTINUE command as follows:

```
:JOB JNAM, UNAM. ACCTNAM  
:CONTINUE  
:RUN PROG1  
:RUN PROG2  
:RUN PROG3  
:EOJ
```

A session may be terminated before a :BYE command is encountered, if any of the following events occur:

1. The Console Operator issues a command to terminate the session.
2. A second :HELLO, :JOB, or :DATA command is encountered. This will implicitly result in a :BYE command, terminating the current session.

SECTION IV

System Generation and Back-up

Hewlett-Packard furnishes MPE/3000 to the customer as a set of prepared program files on magnetic tape. In this initial configuration (generally performed at the factory by HP), the following are included:

- One user with System Manager and System Supervisor Capabilities (plus *all* other standard and optional capabilities from the MPE/3000 Capability Set). This user is identified by the user name MANAGER.
- One System Account (identified by the account name SYS).
- One Public Group (identified by the group name PUB, belonging to the System Account SYS).
- All files necessary to run MPE/3000. (These include all program files required, plus one USL for external interrupts and one segmented library named SL.PUB.SYS.)

Once the system is started (as described in Section V), the user with System Supervisor Capability can call the MPE/3000 Configurator program to

1. Re-configure MPE/3000 by interactively creating a modified version of the running system on magnetic tape. This permits the user to add new software; add or remove input/output devices; change system table sizes, system disc allocation, or segment size limits; and alter scheduling subqueue quanta. (Software that runs as a subsystem of MPE/3000, such as SPL/3000, FORTRAN/3000, or EDIT/3000 is provided on a separate reel of tape and is installed using the :RESTORE command, described in Appendix H.)
2. Copy the currently-running system to magnetic tape. This is done to provide back-up in case the running system is destroyed by a hardware or software failure, or to transfer the system to another installation.

NOTE: A duplicate of the running system on tape is the only way that a user can provide back-up — if no such tape exists and a system failure occurs, the user must contact an HP field representative for help.

Regardless of whether the Configurator is used to re-configure MPE/3000 or to make an identical copy of it, the *running system* is not altered; any changes made apply only to the *copy* of the system written to tape. (Only magnetic tape, and not disc, can be used for this output.)

MODIFYING MPE/3000

When using the Configurator to modify MPE/3000, the user requires certain background information. The information he needs depends on the type of changes he wishes to make, as noted below.

Changing Main Memory Size

MPE/3000 runs on HP 3000 Computers with main-memory of the following sizes:

32K (K = 1024 words)
40K
48K
56K
64K



The size specified by the user during configuration should be the actual size of main-memory delivered with the machine on which this MPE/3000 configuration is to run. This entry is required so that other configuration parameters (such as table sizes) that depend on main-memory size can be set up correctly.

Changing Input/Output Device Configuration

Every physical input/output device in the system is identified by a unique logical device number, ranging from 1 to 255. Input/output configuration consists of specifying this number and various other characteristics for each such device. Some of these characteristics (such as DRT entry number, device unit number, and select code for SIO Multiplexor, described below) are determined by physical hardware connections made prior to system generation. Other characteristics (such as whether a device is interactive or duplicative, whether it can accept jobs and sessions, and the device class to which it belongs) are user options. When the user is deleting or re-specifying devices already on the system, he can determine the characteristics of these devices by requesting a Device Characteristics Listing during the re-configuration process. When he is adding a new device, he must know the hardware-dependent characteristics of the device and must also carefully determine those characteristics that are user options, as noted below. The characteristics that must be specified for each device are

1. Logical Device Number

The logical device number is the value by which the MPE/3000 File Management System recognizes a particular device. For each device, this is a unique, three-digit number ranging from 001 to 255, determined by the user.

2. Device Reference Table (DRT) Entry Number

Every device on the system is cabled to a device controller. (A particular controller may serve more than one device of the same type.) For every controller, there is an entry in the DRT in main-memory that contains pointers to the driver and interrupt programs that serve the controller (and its devices). Because each DRT entry is four words long, the size of the DRT (in words) is

$$4 \times (\text{Total number of controllers})$$

The DRT is located in fixed-memory locations beginning at octal address 14. The maximum upper limit for the DRT is location 1777_8 , thus limiting the maximum number of four-word DRT entries to 253 (decimal).

Since each DRT entry is always four words long, it is convenient for the hardware to map controllers to DRT addresses simply by multiplying by four. Since the DRT begins at location 14, the lowest controller (DRT entry) number is 3 ($14_8/4_8 = 3_8$). (DRT entry numbers 0, 1, and 2 do not exist.)

When re-configuring the system, the user needs to know the highest DRT entry number that can be assigned to a device. He determines this by adding three to the total number of controllers planned. This value may not exceed 255.

The user also needs to know the DRT entry number of any device that he is adding or deleting. This is a hardware-dependent value, consisting of three digits, ranging from 003 to 255. It is determined by a set of jumpers on the device controller board.

3. Unit Numbers

When a controller services only one device, that device is generally assigned a unit number of 0 (recognized by the associated driver). When the controller serves more than one device, each device is assigned a unique unit number (with respect to that controller) to distinguish it from others cabled to the same controller. The unit number of any device is a hardware-dependent characteristic determined when the device is physically connected to its controller. The value ranges from 0 to a maximum number determined by the type of device controller.

4. Start Input/Output (SIO) Multiplexor DRT Entry Number

When a device is run under SIO, the DRT entry number of the SIO multiplexor connected to the device is specified. This number is determined by the user in the same manner as any other device's DRT entry number.

5. SIO Multiplexor Select Code

This number identifies the port on the SIO multiplexor to which the device is connected, as determined by the physical hardware cabling.

6. Device Type

This number determines the type of device, where

- 0 = Moving-Head Disc
- 1 = Fixed-Head Disc
- 8 = Card Reader
- 9 = Paper Tape Reader
- 16 = Terminal
- 24 = Magnetic Tape
- 32 = Line Printer
- 33 = Card Punch
- 34 = Paper Tape Punch

7. Device Sub-Type

This characteristic is specified for a device whose driver handles devices with different characteristics (such as 7-track versus 9-track tape). It is a number ranging from 0 to 15, depending on the actual device referenced (as specified in Appendix D).

8. Corresponding Output Device

If the device can be used as a job/session input device, the user must specify (by logical device number or device class name) a device that will be recognized as the corresponding job/session list device. That is, all input read from the job/session input device is listed on that particular list device.

9. Option to Accept Job and Session Input Stream

The user can optionally specify that this device can accept an input stream from a job or session — in other words, that it can accept the MPE/3000 :JOB or :HELLO commands, and thus serve as a job/session input device.

10. Option to Accept Data from Outside the Job/Session Input Stream

The user can optionally specify that a device can read data from outside the job/session input stream — in other words, that it can accept the :DATA command. (Typically, a device is designated to accept jobs/sessions, but not data external to a job/session.)

11. Interactive Option

The user can optionally specify that this device is a member of an *interactive pair*. An input device and a list device are said to be interactive if a real-time dialogue can be established between a program and a person using the list device as a channel for programmatic requests, with appropriate responses from a person using the input device. For example, an input file and a list file opened to the same terminal would be an interactive pair.

12. Duplicative Option

The user can also specify that the device is a member of a duplicative pair; in such a pair, input from an input device is automatically duplicated on a corresponding list device. For example, a terminal upon which the information input is also output is duplicative.

13. Driver Name

The name of the driver associated with the device/controller is specified. For standard devices supported by HP, appropriate driver names are found in Appendix D. For non-standard drivers supplied by the user, this is the name of the program file containing the driver; the name must contain from one to eight alphanumeric characters, beginning with a letter. (If the driver name is preceded by an asterisk (*), the driver will permanently reside in main-memory.)

14. The general class to which a device belongs must be specified. (This enables a user to request a device by class name, such as any disc or any tape unit.) These names are arbitrary, user-dependent names that are left to the discretion of the System Supervisor. They consist of up to eight alphanumeric characters, beginning with a letter. A device can belong to more than one class, such as DISC and FHDISC. The only device class specifically referenced by MPE/3000 is DISC; it is the default device class for such operations as building files.

Changing System Tables and Queue Parameters

The sizes of various system tables, queues, and other values can be changed to permit the user to make most efficient use of main-memory. The user can best determine what particular values are best for his installation by comparing the values supplied by HP in the initial configuration with those later suggested by his own operational experience. A table of default values (those normally provided initially by HP) appears in Appendix E.

The elements that can be changed are

1. The sizes of the following tables:

Code Segment Table (CST)

Data Segment Table (DST)

Process Control Block (PCB) Table.

2. The maximum number of entries in

Input/Output Queue (IOQ)

User Controller Process (UCOP) Request Queue

3. The number of

Terminal Buffers in the system. (Each buffer is 36 characters long.)

Input/Output Control Blocks (IOCB's) in the system. (Each buffered input/output requires an IOCB.)

4. The number of words in the interrupt control stack (ICS).

5. The maximum number of concurrent time-out requests allowed for the system clock.

Changing Miscellaneous Values Relating to Log-On Time, RIN's, and Jobs Allowed in System and Execution

The user can change

- The time a user is allowed to successfully complete logging-on to the system when initiating a session
- The number of resource identification numbers (RIN's) available in the RIN pool
- The maximum number of global RIN's available
- The maximum number of jobs allowed on the system
- The maximum number of jobs allowed in execution at one time

Changing Disc Allocation

The user can alter the maximum number of disc sectors available for virtual memory. Note that these sectors are used for data-segment swapping only, since code segments are read directly from files and need not be written back to disc.

The maximum number of disc sectors available for the system directory can also be changed. This directory contains the addresses of all files in the system, plus certain accounting information for accounts and groups.

The most efficient values for virtual memory and system directory sizes depend on the installation and its use. Generally recommended values are shown in Appendix E. A list of all MPE/3000 Table Names and Sizes appears in Appendix F.

Changing Scheduling Queue

All processes competing for access to the central processor access it through the MPE/3000 Master Scheduling Queue. (This queue is described in general in Section II and in greater detail in Section VII.)

The user can change the time-quantum, in milliseconds, allowed any circular subqueue (CS or DS).

The user can also re-specify the number of “look-aheads” allowed — that is, the number of stack segments allowed in main-memory *prior to* execution of the processes to which they belong. (Normally, stack segments are not brought into main-memory until a process attains access to the central processor.)

Changing Segment Limits

The user can change the limits on code and data segments, as follows:

1. Maximum number of words allowed in any code segment.
2. Maximum number of code segments per process.
3. Maximum number of words allowed in any user process' stack.
4. Maximum number of words allowed in any extra data segment.
5. Maximum number of data segments per process.
6. Default number of words initially assigned for a user's stack (Z-Q) area (when the user specifies no value).

7. Number of stack overflows allowed before a process is aborted.

Generally recommended values can be found in Appendix E.

Changing Code Segments and Program Files in the System Library

The user can delete, add, and replace code segments in the System Library. He can also replace program files in the System Library. When any such type of change is performed, the segments or program files are taken from the public group of the system account (PUB.SYS). During configuration, the user can request a list of all code segments and program files currently in the System Library. (A list of all MPE/3000 segment names and sizes is shown in Appendix G.)

USING THE :SYSDUMP COMMAND

The initial configuration of any MPE/3000 system (customarily done by HP) requires only a minimum HP 3000 Computer hardware configuration:

- HP 3000 Computer (with central processor and 32,768-word main-memory)
- Multiplexor Channel
- Moving-Head or Fixed-Head Disc
- System Console (Model-33 Teletype Terminal)
- Magnetic Tape Unit
- System Timer

Any type of disc and tape drive supported by MPE/3000 can be used for configuration. Where a minimum hardware configuration is used, the System Console must be a Model-33 Teletype Terminal. Otherwise, any of several terminals can be used.

The user can either re-configure or copy his running MPE/3000 system onto magnetic tape by entering the :SYSDUMP command through his terminal during a session. This command can be entered at any time.

NOTE: To enter this command, the user requires the System Supervisor Capability.

When the :SYSDUMP command is entered, the Configurator begins an interactive dialogue with the user by asking if he wants to make any configuration changes. If he responds NO, the Configurator copies the running system to tape. If he responds YES, the Configurator continues its dialogue with the user.

The Configurator's output consists of questions (ended by a question mark) and statements (ended by a period). The content of the questions generally indicates the type of answer required. To those questions requiring a simple positive or negative answer, the operator responds with YES (or simply Y) or NO (or N, or simply a carriage return). Other questions contain values followed by a question mark; they normally quote an existing parameter value and ask whether the operator wants to change it. To retain the quoted value, the operator enters a carriage return. To change the value, he enters the new value desired. In any case, the operator must always conclude an entry with a carriage return to transmit the entry to MPE/3000.

The :SYSDUMP command format is

:SYSDUMP *dumpfile* [, *auxlistfile*]

- dumpfile* The actual file designator of the output file (device) on which the modified or duplicate system is to be written. (The format for writing actual file designators is explained in *Multiprogramming Executive Operating System (03000-90005)*. Typically, in this case, the designator is a filename comprised of up to eight alphanumeric characters, beginning with a letter.) This must indicate a magnetic tape file. (Required parameter.)
- auxlistfile* The actual file designator of the output file (device) to which listings requested during the Configurator/User dialogue are written. (These listings can include the Input/Output Device and System Library Listings. They may be requested both before and after configuration changes are made.) If this parameter is omitted, the session list device (typically the user's terminal) is assigned by default. (Optional parameter.)

In addition to the names of the files to be used for the *dumpfile* and *auxlistfile* output, the user must specify what type of devices these files are to reside on. He does this by preceding the :SYSDUMP command with MPE/3000 :FILE commands defining each of the two files. Then, in the :SYSDUMP command, he precedes each filename with an asterisk to indicate a back-reference to the definitions in the :FILE commands. (The rules for writing and back-referencing :FILE commands are discussed in *Multiprogramming Executive Operating System (03000-90005)*. An example is shown below, however, to illustrate the general way in which dialogue with the Configurator is initiated.)

EXAMPLE:

To begin dialogue with the MPE/3000 Configurator, a user enters the :SYSDUMP command shown below. At the end of the dialogue, the MPE/3000 configuration requested is copied to the file named DUMP. Any listings requested during the dialogue are output to the file named LIST. The first :FILE command specifies that DUMP is a magnetic tape file; the second :FILE command defines LIST as a line printer. (In these commands, the TAPE and LP parameters are device class names arbitrarily defined by a user during the previous configuration.)

```
:FILE DUMP; DEV=TAPE  
:FILE LIST; DEV=LP  
:SYSDUMP *DUMP,*LIST
```

CONFIGURATOR/USER DIALOGUE

The Configurator/User Dialogue proceeds as follows, with Configurator output shown verbatim in uppercase letters and user input described in mixed upper- and lowercase letters.

Step No.

Dialogue

1 **ANY CHANGES?**

To prepare for changes, enter YES.

To omit changes, and skip to Step 70, enter NO.

2 **SYSTEM ID = HP32000A.UU.FF?**

In this message, *uu* is the present update-level number and *ff* is the fix-level number.

To prepare for updating software for a new fix level, enter the new fix-level digits (*ff*). (These digits indicate the latest system fix provided by HP.)

Otherwise, enter a carriage return.

3 **CORE SIZE?**

To indicate the size of main-memory for the system for which MPE/3000 is being configured, enter one of the following values: 32, 40, 48, 56, and 64. This denotes memory size in a multiple of 1024 words.

4 **I/O CONFIGURATION CHANGES?**

To prepare for addition or deletion of input/output devices, enter YES.

To maintain the same input/output device configuration, and proceed to Step 22, enter NO.

5 **LIST I/O DEVICES?**

To print a list of input/output devices currently assigned to the system, enter YES.

To suppress this listing, enter NO.

Step No.

Dialogue

If an input/output device listing is requested, it is displayed in tabular form, showing the following items for each device:

- Logical Device Number
- DRT Entry Number
- Unit Number
- SIO Multiplexor DRT Entry Number. (If the device does not run under SIO, zero is output.)
- SIO Multiplexor Select Code.
- Device Type, where
 - 0 = Moving-Head Disc
 - 1 = Fixed-Head Disc
 - 8 = Card Reader
 - 9 = Paper Tape Reader
 - 16 = Terminal
 - 24 = Magnetic Tape
 - 32 = Line Printer
 - 33 = Card Punch
 - 34 = Paper Tape Punch
- Device Sub-Type.
- Record Width. (The length of the physical records read or written by the device, in words.)
- Output Device. (The logical device number or device class name of the job/session list device corresponding to jobs/sessions entered on *this* input device. If this is not an *input* device, zero is output.)
- Mode, where
 - J = The device can accept jobs or sessions (:JOB or :HELLO commands).
 - A = The device can accept data external to job/session input stream (:DATA command).
 - I = The device is interactive.
 - D = The device is duplicative.

Step No.

Dialogue

- **Driver Name.** The name of the driver for this input/output device. (The driver resides permanently in main-memory if its name is preceded by an asterisk.)
- **Device Classes.** The classes to which this device belongs defined by a System Supervisor user.

6 **HIGHEST DRT=XX.?**

In the output, *xx* is a value denoting the present highest DRT entry number that can be assigned to a device.

To change *xx*, enter the new value desired. If the highest-numbered device in the configuration is a device that uses two DRT entries (such as a terminal controller with two data sets), be sure to enter the *highest* of the two entry numbers.

To maintain the current *xx*, enter a carriage return.

7 **LOGICAL DEVICE #?**

To specify a device to be added or removed, enter the logical device number of that device.

To skip to Step 21, enter zero.

8 **DRT #?**

To add a device, enter its DRT entry number.

To remove a device and return to Step 7, enter zero.

9 **UNIT #?**

Enter the physical hardware unit number of the device, if the device shares its controller with other devices.

Otherwise, enter zero to continue.

Step No.	Dialogue
----------	----------

10	SIO MPX DRT #, SELECT CODE?
----	------------------------------------

If the device runs under SIO, enter the DRT entry number of the SIO multiplexor to which this device is connected, a delimiting comma, and the port number of the multiplexor to which this device is connected.

If the device does not run under SIO, enter zero.

11	TYPE?
----	--------------

Enter the device type (Appendix D).

12	SUB-TYPE ?
----	-------------------



Enter the device sub-type (Appendix D).

13	RECORD WIDTH?
----	----------------------

Enter the record width for the device (Appendix D).

14	OUTPUT DEVICE?
----	-----------------------

If the device can be used as a job or session input device, enter the device class name or logical device number of the device to be used for the corresponding job/session listing device. (If a device class name is entered, any device of this class can be used as the listing device.)

If this is not a job/session input device, enter zero.

Step No.	Dialogue
15	ACCEPT JOBS/SESSIONS? To specify that this device can accept a job or session input stream, enter YES. Otherwise, enter NO.
16	ACCEPT DATA? To specify that this device can accept data external to a job or session input stream, enter YES. Otherwise, enter NO.
17	INTERACTIVE? To specify that this is an interactive device, enter YES. Otherwise, enter NO or a carriage return.
18	DUPLICATIVE? To specify that this is a duplicative device, enter YES. Otherwise, enter NO.
19	DRIVER NAME? Enter the name of the program file containing the driver for this device (Appendix D). For drivers written and supplied by the user, this name must contain from one to eight alphanumeric characters, beginning with a letter. (If the driver name is preceded by an asterisk, the driver will reside permanently in main-memory.)

Step No.

Dialogue

20

DEVICE CLASSES?

Enter a list containing at least one device class name (up to eight alphanumeric characters, beginning with a letter). Class names are separated from each other by commas. These names are left to the discretion of the System Supervisor. They will be used in certain file commands when any member of a group of devices (such as any disc drive) can be referenced.

A device can belong to more than one class, such as DISC and FHDISC. When the device class list is complete, enter a carriage return to return to Step 7.

21

LIST I/O DEVICES?

To print a listing of the new input/output device configuration, enter YES. This list appears in the format described in Step 5.

To suppress the list, enter NO.

22

SYSTEM TABLE CHANGES?

To prepare for changing the CST, DST, or PCB tables, or other parameters relating to memory usage, enter YES.

To bypass these changes, and proceed to Step 32, enter NO.

23

CST=XXX.?

To change the size of the CST from xxx entries to another value, enter the new value.

To retain the current value, enter a carriage return.

Step No.	Dialogue
24	DST=XXX.? To change the size of the DST from <i>xxx</i> entries to another value, enter the new value. To retain the current value, enter a carriage return.
25	PCB=XXX.? To change the size of the PCB table from <i>xxx</i> entries to another value, enter the new value. To retain the current value, enter a carriage return.
26	I/O QUEUE=XXX.? To change the number of the input/output queue entries permitted from <i>xxx</i> entries to another value, enter the new value. To retain the current value, enter a carriage return.
27	TERMINAL BUFFERS=XXX? To change the number of terminal buffers in the system from <i>xxx</i>, enter the new value. To retain the current value, enter a carriage return.
28	IOCB=XXX? To change the number of input/output control blocks in the system, enter the new value. To retain the current value, enter a carriage return.

Step No.	Dialogue
29	ICS=XX.? To change the number of words in the interrupt control stack (ICS), enter the new value. To retain the current value, enter a carriage return.
30	UCOP REQUEST QUEUE=XX.? To change the number of entries allowed in the User Controller Process Request Queue to another value, enter the new value. To retain the current value, enter a carriage return.
31	TIMER REQUEST LIST=XX.? To change the maximum number of concurrent time-out requests for the system clock allowed, enter the new value. To retain the current value, enter a carriage return.
32	MISC CONFIGURATION CHANGES? To prepare for the following miscellaneous configuration changes, enter YES: • Number of seconds allowed for logging-on. • Number of resource identification numbers (RIN's) available in the RIN pool. • Maximum number of global RINs available. • Maximum number of jobs allowed on the system. • Maximum number of concurrent jobs allowed in execution. To bypass these changes and proceed to Step 38, enter NO.

Step No.	Dialogue
33	# OF SECONDS TO LOGON=XXX.? To change the number of seconds allowed for logging-on, enter the new value. To retain the current value, enter a carriage return.
34	# OF RINS=XXX.? To change the number of RIN's available in the RIN pool, enter the new value. To retain the current value, enter a carriage return.
35	MAX # OF GLOBAL RINS=XXX.? To change the maximum number of global RIN's, enter the new value. To retain the current value, enter a carriage return.
36	MAX # OF JOBS=XXX.? To change the maximum number of jobs allowed on the system, enter the new value. To retain the current value, enter a carriage return.
37	MAXIMUM # OF CONCURRENT RUNNING JOBS=XXX? To change the maximum number of jobs allowed in execution at one time, enter the new value. To retain the current value, enter a carriage return.

Step No.

Dialogue

38 **DISK ALLOCATION CHANGES?**

To prepare for disc allocation changes, enter YES.

To bypass such changes and proceed to Step 40, enter NO.

39 **VIRTUAL MEMORY=XXXX.?**

To change the size of the area on disc used for virtual memory from *xxxx* sectors to another value, enter the new value.

To retain the current value, enter a carriage return.

40 **DIRECTORY=XXXX?**

To change the maximum size of the directory from *xxxx* sectors, enter the new values for this parameter.

To retain the present maximum size, enter a carriage return.

41 **LIST VOLUME TABLE?**

To list the disc volumes and their currently-assigned logical device numbers, enter YES. The listing is printed in the following format:

<i>VOLUME</i>	<i>LOG DEV #</i>
<i>volname</i>	<i>ldn</i>
:	
:	

In this listing, *volname* is a name of up to eight alphanumeric characters, beginning with a letter, identifying the volume; *ldn* is the logical device number assigned to that volume.

To suppress this listing, enter NO.

Step No.	Dialogue
42	DELETE VOLUME ? To prepare to delete a volume, enter YES. To bypass deletion and skip to Step 44, enter NO.
43	ENTER VOLUME NAME ? To delete a volume, enter the volume name. (When the name is entered, the question is repeated.) Otherwise, enter a carriage return.
44	ADD VOLUME ? To prepare to add a volume, enter YES. To bypass addition and skip to Step 46, enter NO.
45	ENTER VOLUME NAME To add a volume, enter the volume name. (When the name is entered, the question is repeated.) Otherwise, enter a carriage return.
46	LIST VOLUME TABLE ? To list the disc volumes and their currently assigned logical device numbers (as in Step 41), enter YES. In this listing, volumes just added (in Steps 44-45) will have logical device numbers of zero. To suppress this listing, enter NO.

Step No.

Dialogue

47 SCHEDULING CHANGES?

To prepare for changes to the scheduling queue, enter YES.

To bypass these changes and proceed to Step 51, enter NO.

48 TIME QUANTUM-TIME SHARE SUBQUEUE=XXX.?

To change the time-quantum of the CS (time-sharing) subqueue (expressed in milliseconds), enter the new value.

To retain the current value, enter NO.

49 TIME QUANTUM-BATCH SUBQUEUE=XXX.?

To change the time-quantum of the DS (multiprogramming batch) subqueue, enter the new value.

To retain the current value, enter NO.

50 # OF MAKE-PRESENT LOOKAHEADS=X.?

To change the number of stack-segments MPE/3000 can bring into main-memory in anticipation of transferring control to other processes in the scheduling queue, enter the new value. (Normally, a stack segment is not brought into main-memory until the process to which it belongs is granted control of the central processor.)

To retain the current value, enter NO.

51 SEGMENT LIMIT CHANGES?

To prepare for changing the limits on code and data segments, enter YES.

To retain the current limits and skip to Step 59, enter NO.

Step No.	Dialogue
52	MAX CODE SEG SIZE=XXXX.? To change the maximum number of words allowed in any code segment from <i>xxxx</i>, enter the new value. To retain the current value, enter a carriage return.
53	MAX # OF CODE SEGMENTS/PROCESS=XXX.? To change the maximum number of code segments allowed any user process, enter the new value. To retain the current value, enter NO.
54	MAX STACK SIZE=XXXXX.? To change the maximum number of words allowed in any subsystem stack from <i>xxxxx</i>, enter the new value. (A maximum value of 32767 is permitted.) To retain the current value, enter a carriage return.
55	MAX EXTRA DATA SEG SIZE=XXXXX.? To change the maximum number of words allowed in any extra data segment from <i>xxxxx</i>, enter the new value. To retain the current value, enter a carriage return.
56	MAX # OF EXTRA DATA SEGMENTS/PROCESS=X.? To change the maximum number of extra data segments that a process can have, enter the new value. To retain the current value, enter a carriage return.

Step No.	Dialogue
57	STD STACK SIZE=XXXX.? To change the default number of words initially assigned for a user stack (Z-Q area) by default (when the user specifies no value) at preparation time from <i>xxxx</i>, enter the new value. To retain the current value, enter a carriage return.
58	ALLOWED OVERFLOWS=XX? To change the number of stack overflows allowed before a process is aborted, from <i>xx</i>, enter the new value. To retain the current value, enter a carriage return.
59	SYSTEM MODULE CHANGES? To prepare for system module changes, enter YES. To proceed directly to Step 70, enter NO.
60	LIST LIBRARY? To list the names of the code segments and systems programs in the System Library, enter YES. To suppress this listing, enter NO.
61	DELETE SEGMENT? To prepare for deleting a code segment from the System (SL) Library, enter YES. To proceed directly to Step 63, enter NO.

Step No.

Dialogue

62 ENTER SEGMENT NAME ?

To delete a code segment from the System (SL) Library, enter the name of that segment. (When the segment name is entered, the question is repeated.)

Otherwise, enter a carriage return.

63 REPLACE SEGMENT ?

To prepare for replacing a code segment in the System Library, enter YES.

To proceed directly to Step 65, enter NO.

64 ENTER SEGMENT NAME, USLFILE NAME [,S/C/P] ?

To replace a code segment in the System Library, enter the name of the segment; a delimiting comma; and the name of the USL file in the public group of the system account (PUB.SYS) where the replacement segment can be found. Also, optionally, enter a delimiting comma followed by one of these three characters:

- S** To declare the segment to be a permanently-allocated system intrinsic segment (in virtual memory).
- C** To declare the segment to be a main-memory resident system intrinsic segment.
- P** To declare the segment to be a permanently-allocated user segment (in virtual memory). (This option requests the same function as the :ALLOCATE command, described in Section VII.)

The question is then repeated.

Otherwise, enter a carriage return.

65 ADD SEGMENT ?

To prepare for adding a code segment to the System Library, enter YES.

Otherwise, enter NO.

Step No.

Dialogue

66 **ENTER SEGMENT NAME, USLFILE NAME [,S/C/P]?**

To add a code segment to the System Library, enter the name of the segment; a delimiting comma; the name of the USL file in the public group of the system account (PUB.SYS) where the segment can be found. Also, optionally, enter a delimiting comma followed by one of these three characters:

- S To declare the segment to be a permanently-allocated system intrinsic segment (in virtual memory).
- C To declare the segment to be a main-memory resident system intrinsic segment.
- P To declare the segment to be a permanently-allocated user segment (in virtual memory). (This option requests the same function as the :ALLOCATE command, described in Section VII.)

The question is then repeated.

Otherwise, enter a carriage return.

67 **SYSTEM PROGRAM CHANGES?**

To prepare to replace a program belonging to the system, enter YES.

To proceed directly to Step 69, enter NO.

68 **ENTER PROGRAM NAME, REPLACEMENT FILE NAME?**

To replace a program belonging to the system, enter the name of the program, a delimiting comma, and the name of the program file in the public group of the system account (PUB.SYS) which is to replace the program.

Otherwise, enter a carriage return.

Step No.

Dialogue

69

LIST LIBRARY?

To list the updated system library, enter YES.

To suppress this listing, enter NO.

70

The system is now copied to tape (multi-reel files). It may now be loaded and initialized as directed in the next section of this manual.

SECTION V

System Start-up, Modification and Shutdown

Any person with access to the MPE/3000 Console can start up MPE/3000, alter the current input/output device configuration, and shut down the system. (This person need not have System Manager or System Supervisor Capability; in fact, since he need not log-on to the system, he need not even have standard user capability.)



START-UP AND MODIFICATION

Start-up of MPE/3000, and re-configuration of the input/output devices on the system is done through a program called the MPE/3000 Initiator. The Initiator provides an option of four types of start:

- **COOLSTART** cold-loads the MPE/3000 System from the system disc. (This is the standard operating procedure when a system is routinely shut down at night and brought up again the next day.) All resident user files (including programs such as FORTRAN/3000, SPL/3000, and EDIT/3000 that run as MPE/3000 subsystems) are saved, but the operational environment present prior to the last shutdown is not retained. Thus, all jobs or sessions in progress at shutdown are lost.
- **COLDSTART** cold-loads the MPE/3000 System from magnetic tape, using the input/output device and system configuration on that tape plus the user files and directory information currently on disc. This allows modification of the system configuration while retaining the user's information.
- **UPDATE** cold-loads the MPE/3000 System from magnetic tape, using the input/output device and system configuration currently on the system disc and user files on disc. This is the standard operating procedure used when starting the system with an updated MPE/3000 tape from HP, and should only be used in that situation.

- RELOAD cold-loads the entire MPE/3000 System, including all system files, directory and configuration information, from magnetic tape. This is normally done when installing the system from the first MPE/3000 tape received from HP, or when restoring the system following a disc crash (from a tape generated by the user through a :SYSDUMP command (Section IV)).

No start-up option permits resumption of user batch jobs or time-sharing sessions interrupted by a system shut-down; such jobs must be re-initiated from the beginning by the user.

All four start-up options allow the user to alter the input/output device configuration currently in effect. This is done through an Initiator/User Dialogue identical to Steps 3 through 21 of the Configurator/User Dialogue described in Section IV. (For information about the kind of information required in this dialogue, see Section IV.)

When the system is started, the initialization of blank disc packs, the writing of labels for them, and the renaming of volumes, is accomplished by the MPE/3000 Initiator. (Formatting of blank disc packs however, is done by a disc diagnostic program rather than by the MPE/3000 Initiator.)

The Initiator's output consists of questions (ended by a question mark) and statements (ended by a period). The content of the questions generally indicates the type of answer required. To those questions requiring a simple positive or negative answer, the user responds with YES (or simply Y) or NO (or N, or simply a carriage return). Other questions contain values followed by a question mark; they normally quote an existing parameter value and ask whether the user wants to change it. To retain the quoted value, the user enters a carriage return. To change the value, he enters the new value desired. In any case, the user must always conclude an entry with a carriage return to transmit the entry to MPE/3000.

To begin the Initiator/User Dialogue, the user follows these steps:

1. Mount the magnetic tape or disc containing the system and any associated new files and configuration data on the appropriate device. If the system is loaded from the system disc, that disc should be mounted on the disc drive specified by Device Reference Table (DRT) Entry Number 5, Unit 0. (Other discs can be mounted on whatever drives desired.) If the system is loaded from magnetic tape, that tape should be mounted on the tape unit specified by DRT Entry Number 6, Unit 0. The operator's console must be the terminal specified by DRT Entry Number 3.)
2. Set the SWITCH REGISTER on the Computer Control Panel with the control-byte setting and DRT entry number of the device on which the system disc or system tape is mounted. (If the system is loaded from disc, set the switch register to octal 5. If the system is loaded from magnetic tape, set the switch register to octal 3006 (The control byte setting is 0 for disc and 6 for tape.)

The SWITCH REGISTER (Figure 5-1) consists of 16-bits; each bit is set *on* by placing the corresponding switch in the up position, or *off* by placing that switch in the down position. The control byte is represented by the leftmost byte (Bits 0-7); the DRT entry number, by the rightmost byte (Bits 8-15).

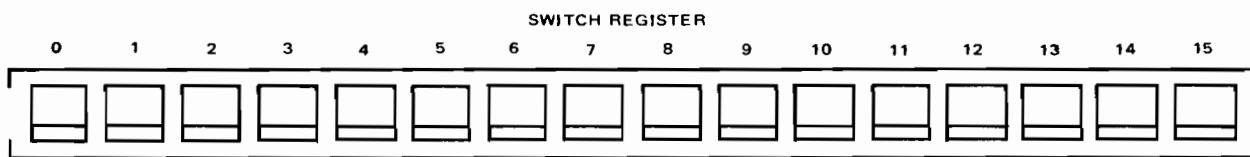


Figure 5-1. Switch Register

3. Press the following switches (on the Computer Control Panel) downward in the order shown:

- a. I/O RESET
- b. CPU RESET
- c. COLD LOAD

This is the cold-load operation which reads into memory the MPE/3000 Initiator.

4. Press the RUN switch. The MPE/3000 Initiator enters execution, printing the following message on console:

HP 32000A.UU.FF

In this message, *uu* is the present update-level number and *ff* is the fix-level number.

The Initiator next begins its dialogue with the user by printing the following information on the console. (In the dialogue shown below, output from the Initiator is shown verbatim in upper-case letters, and input from the user is described in mixed upper and lower-case letters.)

If cold-loading is done from magnetic tape, the MPE/3000 Initiator begins its dialogue at Step 1. If cold-loading is done from disc, the COOLSTART option is assumed, and the dialogue begins with Step 2.

Step No.	Dialogue
1	WHICH OPTION (COLDSTART/RELOAD/UPDATE)? Enter the option desired.
2	LOAD MAP? To request a map showing the correspondences between MPE/3000 segments, programs, and code segment table (CST) entries, enter YES. (The format of the map is shown in Figure 5-2, following this dialogue.) To suppress this map, enter NO.
3	ANY CHANGES? To proceed with changes to the current configuration, enter YES. To maintain the current configuration and skip to Step 28, enter NO.

Step No.**Dialogue**

4

CORE SIZE=XX?

The value *xx* is the current size of main-memory for the system (in multiples of 1024 words). To change this value, enter one of the following values (specifying memory size in multiples of 1024 words): 32, 40, 48, 56, or 64.

5

LIST I/O DEVICES?

To print a list of input/output devices currently assigned to the system, enter YES.

To suppress this listing, enter NO.

If an input/output device listing is requested, it is displayed in tabular form, showing the following items for each device:

- Logical Device Number
- DRT Entry Number
- Unit Number
- SIO Multiplexor DRT Entry Number. (If the device does not run under SIO, zero is output.)
- SIO Multiplexor Select Code.
- Device Type, where
 - 0 = Moving-Head Disc
 - 1 = Fixed-Head Disc
 - 8 = Card Reader
 - 9 = Paper Tape Reader
 - 16 = Terminal
 - 24 = Magnetic Tape
 - 32 = Line Printer
 - 33 = Card Punch
 - 34 = Paper Tape Punch

Step No.

Dialogue

- Device Sub-Type
- Record Length. (The length of the physical records read or written by the device, in words.)
- Output Device. (The logical device number or device class name of the job/session list device corresponding to jobs/sessions entered on *this* input device. (If this is not an *input* device, zero is output.)
- Mode, where
 - J = The device can accept jobs or sessions (:JOB or :HELLO commands).
 - A = The device can accept data external to job/session input stream (:DATA command).
 - I = The device is interactive.
 - D = The device is duplicative.
- Driver Name. The name of the driver for this input/output device. (The driver resides permanently in main-memory if its name is preceded by an asterisk.)
- Device Classes. The classes to which this device belongs, defined by a System Supervisor user.

6

HIGHEST DRT NUMBER=XX.?

In the output, *xx* is a number denoting the present highest DRT entry number that can be assigned to a device.

To change *xx*, enter the new value desired. If the highest-numbered device in the configuration is a device that uses two DRT entries (such as a terminal controller with two data sets), be sure to enter the *highest* of the two entry numbers.

To maintain the current *xx*, enter a carriage return.

Step No.	Dialogue
7	LOGICAL DEVICE #? To specify a device to be added or removed, enter the logical device number of that device. To skip to Step 21, enter zero.
8	DRT #? To add a device, enter its DRT entry number. To remove a device and return to Step 6, enter zero.
9	UNIT #? Enter the physical hardware unit number of the device, if the device shares its controller with other devices. Otherwise, enter zero to continue.
10	SIO MPX DRT #? If the device runs under SIO, enter the DRT entry number of the SIO multiplexor to which this device is connected. If the device does not run under SIO, enter zero to skip to Step 12.
11	SIO MPX SELECT CODE? Enter the port number on the SIO multiplexor to which this device is connected.
12	TYPE? Enter the device type (Appendix D).

Step No.	Dialogue
13	SUB - TYPE ?
	Enter the device sub-type (Appendix D).
14	OUTPUT DEVICE ?
	If the device can be used as a job or session input device, enter the device class name or logical device number of the device to be used for the corresponding job/session listing device. (If a device class name is entered, any device of this class can be used as the listing device.)
	If this is not a job/session input device, enter zero.
15	ACCEPT JOBS OR SESSIONS?
	To specify that this device can accept a job or session input stream, enter YES.
	Otherwise, enter NO.
16	ACCEPT DATA?
	To specify that this device can accept data external to a job or session input stream, enter YES.
	Otherwise, enter NO.
17	INTERACTIVE?
	To specify that this is an interactive device, enter YES.
	Otherwise, enter NO or a carriage return.

Step No.**Dialogue****18 DUPLICATIVE?**

To specify that this is a duplicative device, enter YES.

Otherwise, enter NO.

19 DRIVER NAME?

Enter the name of the program file containing the driver for this device (Appendix D). For drivers written and supplied by the user, this name must contain from one to eight alphanumeric characters, beginning with a letter. (If the driver name is preceded by an asterisk, the driver will permanently reside in main-memory.)

20 DEVICE CLASSES?

Enter a list containing at least one device class name (up to eight alphanumeric characters, beginning with a letter). Class names are separated from each other by commas. These names are left to the discretion of the System Supervisor. They will be used in certain file commands when any member of a group of devices (such as any disc drive) can be referenced.

A device can belong to more than one device class, such as DISC and FHDISC. When the device class list is complete, enter a carriage return to return to Step 7.

21 LIST I/O DEVICES?

To print a listing of the new input/output device configuration, enter YES. This list appears in the format described in Step 5.

To suppress the list, enter NO.

Step No.**Dialogue****22 LIST VOLUME TABLE?**

To list the disc volumes and their currently-assigned logical device numbers, enter YES. The listing is printed in the following format:

<i>VOLUME</i>	<i>LOG DEV #</i>
<i>volname</i>	<i>ldn</i>
:	
:	

In this listing, *volname* is a name of up to eight alphanumeric characters, beginning with a letter, identifying the volume; *ldn* is the logical device number assigned to that volume.

To suppress this listing, enter NO.

Regardless of whether the user requested the listing (by entering YES) or suppressed the listing (by entering NO), at this point in the dialogue, control transfers to Step 23 (if this is a RELOAD) or skips to Step 25 (if this is a COLDSTART, COOLSTART, or UPDATE).

23 DELETE VOLUME?

To prepare to delete a volume, enter YES.

To bypass deletion and skip to Step 25, enter NO.

24 ENTER VOLUME NAME?

To delete a volume, enter the volume name. (When the name is entered, the question is repeated.)

Otherwise, enter a carriage return.

25 ADD VOLUME?

To prepare to add a volume, enter YES.

To bypass addition and skip to Step 27, enter NO.

Step No.**Dialogue****26 ENTER VOLUME NAME?**

To add a volume, enter the volume name. (When the name is entered, the question is repeated.)

Otherwise, enter a carriage return.

If any volume was deleted (Steps 23-24) or added (Steps 25-26), the Initiator proceeds to Step 27. Otherwise, the Initiator skips to Step 28.

27 LIST VOLUME TABLE?

To list the disc volumes and their currently-assigned logical device numbers (as in Step 22), enter YES.

To suppress this listing, enter NO.

28 At this point, the MPE/3000 Initiator ensures that all volumes defined in the volume table are mounted, that all discs have valid labels, and that the volume name of each disc is in the volume table. As a result of this verification, any of the following messages may appear:

- a. *DEVICE 1dn VOLUME volname NOT DEFINED IN TABLE
ENTER VOLUME NAME?*

This means that the volume identified by *volname*, with the logical device number *1dn* does not appear in the volume table. To add this volume, identified by this *volname* and *1dn*, enter a carriage return.

To add this volume under a different volume name, enter the new name. The new name will be entered in the volume table, and the volume will be relabeled with that name.

- b. *INVALID LABEL FOR DEVICE 1dn ENTER VOLUME NAME?*

This indicates that device *1dn* does not contain a valid MPE/3000 volume label. The user must enter a volume name. If the name entered corresponds to that of a volume in the volume table, the logical device number *1dn* is set for that volume. Otherwise, a new entry is made in the volume table. The disc label is then updated.

Step No.

Dialogue

c. ALL VOLUMES MUST BE MOUNTED

This message occurs when an entry appears in the volume table but no corresponding disc is attached to the system.

If any of the above messages appears and the user responds properly, the Initiator proceeds to Step 29. If none of these messages is displayed, the Initiator skips to Step 30.

29 LIST VOLUME TABLE?

To list the disc volumes and their currently-assigned logical device numbers (as in Steps 22 and 27), enter YES.

To suppress this listing and return to Step 5, enter NO.

30 DATE?

Enter the current date in the following format:

mm/dm/yr

where

mm = Two digits representing the month
dm = Two digits representing the day of the month.
yr = The last two digits of the year.

31 TIME?

Enter the current time-of-day in the following format:

hh:mm

where

hh = Two digits indicating the hour (on a 24-hour basis).
mm = Two digits indicating the minute of the hour.

The Initiator program now terminates, transferring control to MPE/3000. The system is ready for use.

Load Map

The load map requested in Step 2 of the Initiator/Operator Dialogue appears as shown in Figure 2-2. This map shows the correspondence between MPE/3000 code segments and programs, and code segment table (CST) entries. In the first column of this listing, the CST number (in octal) is shown. In the second column, the System Segmented Library (SL) segment name or program name is presented. (SL segment names are followed by a parenthesized value in the third column; program file names are not.) In the third column, the parenthesized number indicates the logical segment number of the segment within the system library (identified as SL.PUB.SYS).



20	DIRC	(0)
21	CROUTINE	(1)
22	MMDISK	(2)
23	IOUTILTY	(4)
24	MISC0	(5)
25	CHECKER	(6)
26	PCREATE	(7)
27	MORGUE	(12)
30	MESSAGE	(13)
31	DATASEG	(14)
32	JOBTABLE	(15)
33	PROCMail	(16)
34	ABORTRAP	(17)
35	SYSDEBUG	(20)
36	LOADER1	(21)
37	SYSSTOPS	(22)
40	SYSDSPLY	(23)
41	TTYINT	(24)
42	MMCORER	(26)
43	ALLOCATE	(27)
44	FILESYS7	(30)
45	DISKSPC	(31)
46	FILESYS6	(32)
47	FILESYS1	(51)
50	UTILITY	(107)
51	FILESYS5	(110)
52	FILESYS4	(111)
53	FILESYS3	(112)
54	RINS	(114)
55	PINT	(11)
56	DEBUG	(25)
57	CIINIT	(37)
60	CIMISC	(40)
61	CISUBS	(41)
62	CIFILEB	(42)
63	CIFILEM	(43)
64	CIPRUN	(44)
65	CILISTF	(46)
66	CIUTIL	(47)
67	SEGUTIL	(106)
70	FILESYS2	(113)
71	ININ	
72	DISPATCH	
73	MAPP	
74	DATAIOPM	
75	IOFDISK0	
76	DISKIOPM	
77	IOCLTTY0	
100	TERMIOPM	
101	IOTAPE0	
102	SIODIOPM	
103	IOLPRT0	
104	IOCDRD0	
105	IOTERM0	
106	UCOP	
107	DEVREC	
110	LOAD	
111	PROGEN	

Figure 5-2. Load Map

SHUTDOWN

The user can shut the system down by following these procedures.

1. Press the **CONSOLE INTERRUPT** Button on the Computer Control Panel. In response, MPE/3000 prints an equals sign (=) on the MPE/3000 Console.
2. Enter the **SHUTDOWN** command after the equals sign (and terminate the command by pressing the carriage return key.)

= *SHUTDOWN*

The =SHUTDOWN command shuts the system down in an orderly manner, terminating all jobs and sessions currently in progress. The system can be restarted at any time, as directed in the preceding pages.

SECTION VI

Organizational Management

A user with the MPE/3000 System Manager Capability can create, modify, and delete accounts. With each account, the System Manager User also creates a user with the Account Manager Capability and a Public Group (named PUB). The Account Manager User can then create, modify, and delete groups and users within his account. (The Public Group is a group whose file library will continue to be accessible to all users within the account after the MPE/3000 File Security System is implemented in a future system release.)

TYPICAL ACCOUNTS

To illustrate the various ways in which accounts, groups, and users can be defined and organized at an HP 3000 Computer site, several examples are presented. Notice that these are only examples, and are not meant to imply formal organizational rules.

Large Multi-Division Company

Within a system used to provide time-sharing in a large, multi-division corporation, a System Manager User could assign a set of accounts to each division or corporate staff office according to its functional needs (Figure 6-1). For example, he could assign four accounts to Division A — one for each of its departments: Engineering, Production, Marketing, and Finance. For the Engineering Department account (detailed in Figure 6-1), an Account Manager User has defined three users who can access the system. Each has a group (assigned as his home group) where he stores his own programs and files. (Note that when the MPE/3000 File Security System is implemented, access to the files in these groups can be restricted so that a group defines a private or restricted sub-domain of the entire file domain.) In addition, these engineers all can access two special groups that contain programs and data files for current design projects. The engineers can also access a group defined for work on schedules, budgets, and other administrative tasks, and a public group (with no password required at log-on time) for storage of general-purpose utility programs used by all. (Remember that a password, if required for a group or account, is needed only at log-on time, if the user is logging-on under that group or account. Once a user logs-on, he can access any account or group (permitted by his capability set), whether or not it has a password. If the account or group is not the user's log-on account or group, he need only know the account or group name in order to access it.)

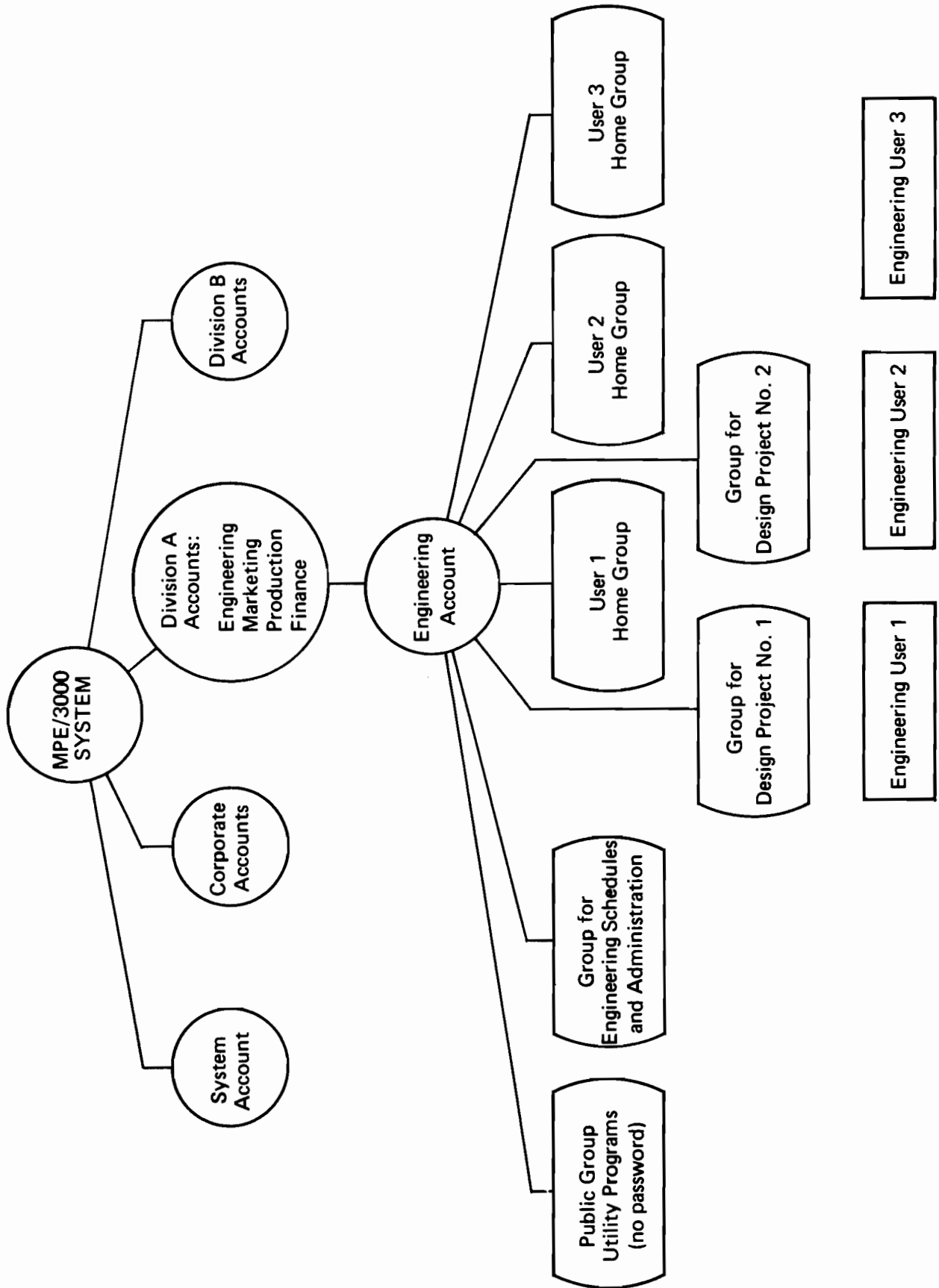


Figure 6-1. Account/Group/User Structure in a Large Corporation

In-House Service Bureau

An in-house service bureau, providing both batch and time-sharing capabilities, serves different users within the same company. The manager of the Data Center, having the System Manager Capability, creates an account for each department (Engineering, Marketing, Accounting, and other departments) in the company. For each account an Account-Manager User assigns as users all programmers in his department, and assigns them all Standard Capabilities plus the following Optional Capabilities: Process-Handling, Data-Segment Handling, and Multiple RINS. The Account Manager User also defines the groups for the account.

Commercial Service Bureau

A commercial service bureau, also providing batch and time-sharing capabilities, offers its services to various outside companies (customers). An account is established for each customer. Users are given Standard Capabilities, but no Optional Capabilities.

Scientific Research Center

In a research center where scientific programmers are using the system for batch and time-sharing applications, the following accounts are available:

- System Account, used by all users.
- Data-Processing account, containing groups allocated as required, used by all research users. For this account, all users have the Standard Capabilities and the following Optional Capabilities: Process-Handling, Data-Segment Handling, and Multiple RIN'S.
- Special Accounts, used by agencies such as the Personnel and Accounting Offices, accessed by members of these agencies.

DEFINING ACCOUNTS

When a user with System Manager Capability accesses MPE/3000 to perform various functions, he logs on to the system in the same way that any other user does. Then, he can enter the commands for managing accounts described below.

The first System Manager User is designated by HP on the system tape supplied the customer. He initially accesses the system under the user name MANAGER, the group name PUB, and the account name SYS. (The initial passwords are null.)

Creating Accounts

To create a new account and an associated Account Manager User and Public Group, the System Manager User enters the following command. (The new Account Manager User can then redefine his own attributes, the Public Group, and new users and groups as he desires.) The Public Group is assigned a null password and has the same attributes as those assigned the account.

NOTE: All names, such as *acctname*, *mgrname*, *password*, are composed of up to eight alphanumeric characters, beginning with a letter.

```
:NEWACCT  acctname,mgrname  
          [;PASS = [password] ]  
          [;CAP = [capabilitylist] ]  
          [;MAXPRI = [subqueueuname] ]  
          [;LOCATTR = [localattribute] ]
```

acctname The name assigned to the new account. (Required parameter.)

mgrname The name of the account manager user; he is always the first user created under the account. He receives the following attributes:

User password:	Null password.
User capability list:	Same as Account's capabilities.
Maximum scheduling priority:	Same as Account's maximum priority.
Local attribute:	Same as Account's local attribute.
Home group:	PUB

(The attributes of an Account Manager User may be changed subsequently, through the :ALTUSER command; in no case, however, can they be greater than those of the account.) (Required parameter.)

password The account password. If omitted, no password is assigned. (Optional parameter.)

capabilitylist

The list of capabilities, mutually separated by commas, permitted this account. Each capability (described in Section II) is denoted by a two-letter mnemonic, as follows:

User Attributes:

System Manager	= SM
Account Manager	= AM
Account Librarian	= AL
Group Librarian	= GL
Diagnostician	= DI
System Supervisor	= OP
Standard User	= SU

File-Access Attributes:

Permanent Files	= SF
Access of non-sharable I/O devices	= ND

Capability-Class Attributes:

Process-Handling	= PH
Extra Data Segments	= DS
Multiple RINS	= MR
Access AS Subqueue	= CR
Access BS Subqueue	= RT
Privileged Mode	= PM
Interactive Access	= IA
Local Batch Access	= BA
Non-Standard Subqueue Access	= NS

If the *capabilitylist* parameter is omitted, the following capabilities are assigned by default:

AM,AL,GL,SU,SF,IA,BA,

(Optional parameter.)

subqueueuname

The name of the subqueue of highest priority that can be requested by any job in the account, specified as "xS," where x is A, B, C, D, or E. If none is specified, the CS subqueue is assigned. (Optional parameter.)

localattribute The local attribute of the user, as defined at the installation site. If omitted, 0 is assigned. This is a double-word of arbitrary meaning which might be used to further classify users. While it will not be involved in standard MPE/3000 security provisions, it is available to processes through the WHO intrinsic for use in the programmer's own security provisions. (Optional parameter.)

In the :NEWACCT command, the default parameters noted are assigned:

1. When a keyword is included but its corresponding keyword parameter is omitted (as in ;PASS=).
2. When an entire keyword parameter grouping (such as ;PASS= password) is omitted.

EXAMPLE:

Suppose that a System Manager wants to create an account with the following characteristics:

Account name = ACI
Account password = PCI
Manager's name = SMYTHE

(All other parameters are assigned by default.)

To accomplish this, the System Manager User could enter:

:NEWACCT ACI,SMYTHE; PASS=PCI

Altering Accounts

A System Manager User can change the attributes of an existing account by entering the :ALTACCT command.

```
:ALTACCT acctname  
    [;PASS = [password]]  
    [;CAP = [capabilitylist]]  
    [;MAXPRI = [subqueueename]]  
    [;LOCATTR = [localattribute]]
```

acctname The name of the account to be altered. (Required parameter.)

<i>password</i>	}	New values for the corresponding parameters described under the :NEWACCT command. The maximum limit for these parameters discussed under :NEWACCT also apply to :ALTACCT. (Optional parameters.)
<i>capabilitylist</i>		
<i>subqueueename</i>		
<i>localattribute</i>		

When an entire keyword parameter group is omitted from the :ALTACCT command, that parameter remains unchanged for the account. When a keyword is included but the corresponding parameter is omitted (as in ;PASS=), the default value is assigned.

Any value changed through the :ALTSEC command becomes effective the next time MPE/3000 is requested to check this value; thus, if an attribute is taken away from an account, users of the account currently running by virtue of that attribute remain on the system until they log-off. After logging-off, however, they cannot log-on again under that account.

EXAMPLE:

To change an account named AC2 so that its password is GLOBALX a System Manager User could enter:

```
:ALTACCT AC2; PASS=GLOBALX
```

Listing Account Attributes

To list the attributes (previously specified in the :NEWACCT or :ALTACCT commands) for one account, or for all accounts in the system, a System Manager User uses this command:

:LISTACCT $\left[\begin{array}{c} @ \\ acctname \end{array} \right]$ [,listfile]

- @** An indication that the attributes for all accounts are to be listed. (Optional parameter.)
- acctname** The name of the account whose attributes are to be listed. (Optional parameter.)
- listfile** The file on which the attribute listing is produced. If this parameter is omitted, the job list device is assigned. (Optional parameter.)

If neither @ or acctname is specified, the log-on account is assumed.

EXAMPLE:

To list the attributes of an account named ACCTNO23, a System Manager User enters:

:LISTACCT ACCTNO23

The listing appears as an octal dump of the account entry or entries. The content of an entry can be decoded as

Words	Content
0:3	Account name.
6:7	Capability (in same format returned by WHO intrinsic, described in <i>Multiprogramming Executive Operating System</i>).
8:9	Local attributes.
10:13	Password
27	Priority (numerical)

A translation of those bytes that contain alphanumeric ASCII characters appears to the right of each line in the octal dump.

Deleting Accounts

To remove an account (and its groups (files) and users) from the system, a System Manager User enters:

:PURGEACCT acctname

acctname The name of the account to be deleted. (Required parameter.)

When the *:PURGEACCT* command is entered in a session, MPE/3000 prints a verification request. This aids the user in making sure he does not accidentally delete the wrong account. The format of the verification request is

ACCT accountname TO BE PURGED?

The user responds with YES or NO.

Actual removal of the account is not done unless all users logged-on under groups in this account have logged off, and any users accessing files belonging to this account have completed such access.

EXAMPLE:

When a System Manager User removes an account named ACTXYZ from the system, the following dialogue takes place:

```
:PURGEACCT ACCTXYZ  
ACCT ACCTXYZ TO BE PURGED? YES
```



DEFINING GROUPS AND USERS

Once an account has been defined and someone has been assigned Account Manager Capability for that account, that person can manage groups and users within the account. The Account Manager User exercises this control through the commands noted below.

Creating Groups

An Account Manager User creates a new group within his account by issuing the :NEWGROUP command. In doing so, he also optionally assigns a password for this group. Thereafter, any user who knows the password can log-on under that group. (The password is not needed to access the files directly.) Additionally, the group can be assigned as a homegroup for any user defined (in the :NEWUSER command, described later). The :NEWGROUP command format is

```
:NEWGROUP groupid  
    [;PASS = [password] ]  
    [;CAP = [capabilitylist] ]
```

- | | |
|-----------------------|---|
| <i>groupid</i> | The name of the new group, composed of up to eight alphanumeric characters, beginning with a letter. (Required parameter.) |
| <i>password</i> | The same meaning and default value as the corresponding parameter in the :NEWACCT command, but applicable to the group. (Optional parameter.) |
| <i>capabilitylist</i> | A list of <i>capability-class attributes</i> , specified as in the :NEWACCT command. The default assigned is IA, BA. (Optional parameter.) |

When a keyword parameter or keyword parameter group is omitted from the :NEWGROUP command, the corresponding default parameter takes effect.

EXAMPLE:

The Account Manager wants to create a new group named GNAME, having the password GPASS. He enters the following command:

```
:NEWGROUP GNAME; PASS=GPASS
```

Altering Groups

To change one or more attributes of a group, an Account Manager User can issue the :ALTGROUP command:

```
:ALTGROUP groupid  
    [;PASS = [password] ]  
    [;CAP = [capabilitylist] ]
```

<i>groupid</i>	The name of the group whose attributes are to be changed. (Required parameter.)
<i>password</i>	New value for the corresponding parameter described in the :NEWGROUP command. (Optional parameter.)
<i>capabilitylist</i>	New value for the corresponding parameter described in the :NEWGROUP command. (Optional parameter.)

When an entire keyword parameter group is omitted from an :ALTGROUP command, the corresponding value for the group remains unchanged. When a keyword is included but the corresponding parameter is omitted (as in ;PASS=), the default value is assigned.

EXAMPLE:

To assign a new password of PASS2 to a group named GROUPXX, the Account Manager enters:

```
:ALTGROUP GROUPXX; PASS=PASS2
```

Listing Group Attributes

An Account Manager User can print a list of attributes (previously specified in the :NEWGROUP or :ALTGROUP commands) for one group, or all groups, by issuing the following command:

```
:LISTGROUP [ @  
            groupname ] [,listfile]
```

@ A specification that the characteristics of all groups are to be listed.
(Optional parameter.)

groupname The name of the group whose characteristics are to be printed.
(Optional parameter.)

listfile The file on which the characteristic listing is to be produced. If
this parameter is omitted, the job list device is assigned. (Optional
parameter.)

If neither the @ or groupname parameter is included, the characteristics of the log-on group are printed.

The listing appears as an octal dump of the group entry or entries. The content of an entry can be decoded as

Words	Content
0:3	Group name
5:8	Password
23	Capability-class attributes (in the same bit-format returned by the WHO intrinsic, described in <i>Multiprogramming Executive Operating System</i>).

A translation of those bytes that contain alphanumeric ASCII characters appears to the right of each line in the octal dump.

Deleting Groups

To remove a group (and all files belonging to it) from the system, an Account Manager User enters:

```
:PURGEGROUP groupname
```

groupname The name of the group to be removed. (Required parameter.)

When this command is entered in a session, MPE/3000 prints this message:

```
GROUP groupname TO BE PURGED?
```

The user verifies his intent to delete the group by responding YES; otherwise, he responds NO.

Actual removal of the group cannot be done unless all users logged-on under this group have logged-off, and any users accessing files belonging to this group have completed such access.

EXAMPLE:

When an Account Manager User removes a group named EXGROUP from the system, this dialogue takes place:

```
:PURGEGROUP EXGROUP  
GROUP EXGROUP TO BE PURGED? YES
```

Defining Users

An Account Manager User can define a new user through the `:NEWUSER` command. This user can log-on to any group to which he has the password, or which is assigned as his homegroup.

```
:NEWUSER userid  
    [;PASS = [password]]  
    [;CAP = [capabilitylist]]  
    [;MAXPRI = [subqueueuname]]  
    [;LOCATTR = [localattribute]]  
    [;HOME = [homegroupname]]
```

<i>userid</i>	The name of the new user, composed of up to eight alphanumeric characters, beginning with a letter. (Required parameter.)
<i>password</i>	The same meaning and default value as the corresponding parameter in the <code>:NEWACCT</code> command, but applicable to the user. (Optional parameter.)
<i>capabilitylist</i>	The same meaning as the <i>capabilitylist</i> parameter for the <code>:NEWACCT</code> command, but applicable only to this user. This parameter cannot specify capabilities not assigned to the Account (in <code>:NEWACCT</code> or <code>:ALTACCT</code> commands.) If omitted, the following values are assigned by default: SU,SF,IA,BA. (Optional parameter.)
<i>subqueueuname</i>	The same meaning as the <i>subqueueuname</i> parameter for the <code>:NEWACCT</code> command, but applicable only to this user. The priority specified in <code>:NEWUSER</code> cannot be greater than that specified in <code>:NEWACCT</code> or <code>:ALTACCT</code> . (Optional parameter.)
<i>localattribute</i>	The local attribute defined at the installation; this must be a subset of the bit configuration for the <i>account's</i> local attribute. (Optional parameter.)
<i>homegroupname</i>	The name of an existing group to be assigned as the homegroup for this user. If omitted, no home group is assigned, and so the user must always specify a group when he logs on. (Optional parameter.)

If the optional parameters or keyword parameter groups are omitted, the corresponding parameters specified in `:NEWACCT` are assigned by default.

EXAMPLE:

To define a new user as *LHSMITH*, and assign him a password of *SMITTY* and a homegroup of *HOMGPX*, the Account Manager User issues the following command. (The user will have the default capabilities and maximum subqueue priority of the account to which he is assigned.)

```
:NEWUSER LHSMITH; PASS=SMITTY; HOME=HOMGPX
```

Altering User Attributes

To alter the attributes currently defined for a user, an Account Manager User enters:

```
:ALTUSER userid  
  [;PASS = [password]]  
  [;CAP = [capabilitylist]]  
  [;MAXPRI = [subqueueuname]]  
  [;LOCATTR = [localattribute]]  
  [;HOME = [homegroupname]]
```

userid The name assigned to the user (in the :NEWUSER command).
 (Required parameter.)

<i>password</i>	}	The new values to be assigned the corresponding parameters originally specified in the :NEWUSER command. (Optional parameters.)
<i>capabilitylist</i>		
<i>subqueueuname</i>		
<i>localattribute</i>		
<i>homegroupname</i>		

When an entire keyword parameter group is omitted from an :ALTUSER command, the corresponding value for the user remains unchanged. When a keyword is included but the corresponding parameter is omitted (as in ;CAP =), the default value is assigned.

Any value changed through the :ALTUSER command becomes effective the next time this value is checked; thus, if an attribute is taken away from a user currently logged-on to the system by virtue of this attribute, that user maintains his access to MPE/3000 until he logs-off.

Listing User Attributes

An Account Manager User can list the attributes currently assigned to a single user, or to all user's in his account, by issuing the :LISTUSER command:

```
:LISTUSER [ @  
           username ] [,listfile]
```

- @ A specification that the characteristics of all user's in the account are to be listed. (Optional parameter.)
- username The name of the user whose attributes are to be listed. (Optional parameter.)
- listfile The file on which the attribute listing is produced. If this parameter is omitted, the job list device is assigned. (Optional parameter.)

If neither the @ or username parameter is included, the attributes of the user issuing the :LISTUSER command are listed.

The listing appears as an octal dump of the user entry or entries. The content of an entry can be decoded as:

Words	Content
0:3	User name
4:5	Capability (in the same bit-format returned by the WHO intrinsic, described in <i>Multiprogramming Executive Operating System</i>).
6:7	Local attributes
8:11	Password
12:15	Home group
17	Maximum job priority (numerical)

A translation of those bytes that contain alphanumeric ASCII characters appears to the right of each line in the octal dump.

Deleting Users

To remove a user from the account, an Account Manager User issues the `:PURGEUSER` command. This command also transfers ownership of all files created by this user to the Account Manager User issuing `:PURGEUSER`, and prints a list of these files. The `:PURGEUSER` command format is

`:PURGEUSER username`

username The name of the user to be deleted. (Required parameter.)

When this command is entered in a session, MPE/3000 issues the following message:

USER username TO BE PURGED?

The Account Manager User must verify his intent by responding YES or NO.

If a `:PURGEUSER` command references a user currently logged onto the system, that user is *not* purged and a message is output indicating this fact.

COMMAND SUMMARY

The MPE/3000 System/Account Management Commands are summarized below according to the user attribute (System Manager or Account Manager) required to enter them.

System Manager (SM) Capability

:NEWACCT

:ALTACCT

:LISTACCT

:PURGEACCT

Account Manager (AM) Capability

:NEWGROUP

:ALTGROUP

:LISTGROUP

:PURGEGROUP

:NEWUSER

:ALTUSER

:LISTUSER

:PURGEUSER

:LISTACCT (user's own account.)

SECTION VII

Operational Management

A user with System Supervisor Capability can enter various commands which enable him to have supervisory operational control of the system. These commands, discussed in this section, can be entered in either session or batch mode. The session mode allows the user the option of requesting functions from a remote terminal.

MANAGING THE MASTER QUEUE

All processes competing for the central processor access it in an orderly manner, under the direction of MPE/3000. The system places all processes in the master scheduling queue in order of their priority. When a process in execution is completed, terminated, or suspended, the MPE/3000 Dispatcher searches the master queue for the next process of highest priority, and transfers control to that process.

The master queue (Figure 7-1) is divided into logical areas, each corresponding to a particular type of dispatching and priority class for the processes within it. A logical area can be a linear subqueue, a circular subqueue, or a portion of the main master queue. In a linear subqueue, the process with highest priority accesses the central processor first and maintains this access until the process either is completed, terminated, or suspended to await the availability of a required resource. In a circular subqueue, all processes have equal priority and each accesses the central processor for an interval (time quantum) of maximum duration (or until completed, terminated, or suspended). At the end of this duration, control is transferred to the next process in the queue, continuing in a round-robin fashion. This time-slicing is controlled by the system timer. Processes that are not scheduled in a subqueue are scheduled in the master queue.

Each subqueue in the master queue is defined by a priority number. While the standard user is aware of the priority class associated with a subqueue, only a user with System Supervisor Capability is concerned with priority numbers. The standard subqueues (priority classes) are as follows; additional subqueues can be added at system generation time. (See section IV.)

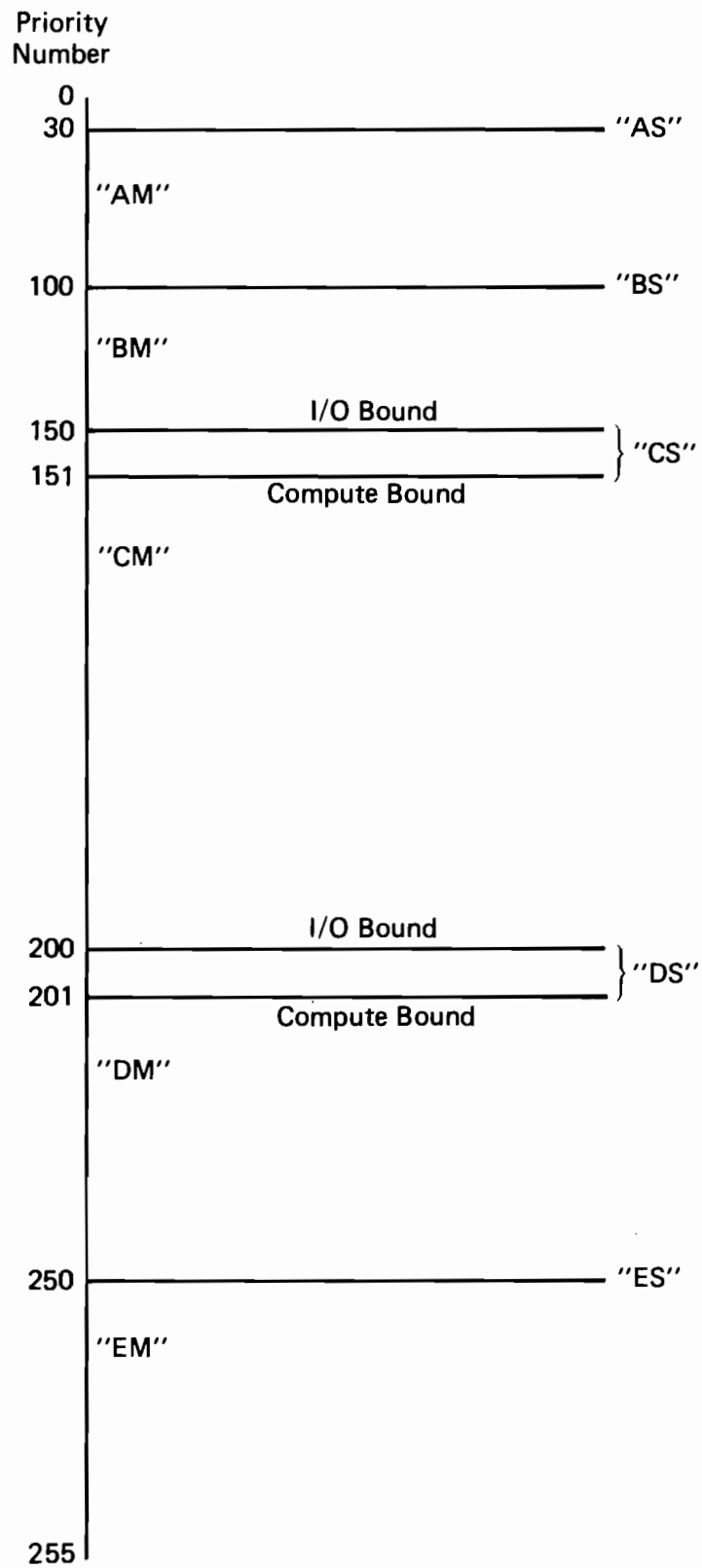


Figure 7-1. MPE/3000 Master Queue Structure

AS	is a linear subqueue containing processes of very high priority. (In a subsequent MPE/3000 release, it will be accessible only to users with a special <i>MPE/3000 Optional Capability</i> to be added to the system then.) Its priority number is 30.
BS	is a linear subqueue containing processes of high priority. (In a subsequent MPE/3000 release, it will be accessible only to users having a special <i>MPE/3000 Optional Capability</i> to be added to the system.) Its priority number is 100.
CS	is composed of two circular sub-subqueues, each having a time-quantum determined at system generation time. This subqueue is used mainly for time-sharing processes. The time quantum specifies the maximum execution time allowed any process in the sub-subqueue before going on to the next process. The priority numbers of the sub-subqueues are 150 and 151; the lower-priority sub-subqueue is used for compute-bound processes, and the higher-priority sub-subqueue is used for input/output bound processes.
DS	is also composed of two circular sub-subqueues, each having a time-quantum determined at configuration time. This subqueue is used primarily for multiprogramming batch processes. The priority numbers of the sub-subqueues are 200 and 201; as in the CS subqueue, one sub-subqueue is used for compute-bound processes and the other is used for input/output processes.
ES	is a linear subqueue, primarily containing idle processes with low priority. Its priority number is 250.
AM,BM,CM,DM,EM	are discontinuous portions of the main master queue. Processes can be scheduled in these areas only if they are system processes or are initiated by a user with System Supervisor Capability. Such processes are dispatched linearly.

The type of process run need not bear relation to the subqueue in which it is scheduled. For example, a batch job could be run in the CS subqueue (which is used primarily for time-sharing processes, but not *restricted* for them).

The priority class of a process can be specified by the normal user with standard or optional MPE/3000 capabilities. In the two-character string that comprises a priority-class reference, the first character refers to the location of a subqueue within the master queue (in alphabetical order) and the second character specifies whether the logical area is the subqueue itself (S) or the portion of the master queue (M) that immediately follows the subqueue. When a priority class is requested for a process, it is assigned the lowest priority within that class (relative to other processes already assigned the same class). In a circular subqueue, the actual priority of the process is modified as other processes use central processor time. In a linear subqueue, an automatic adjustment is made as necessary.

Only a User with System Supervisor Capability can assign a priority number to a process. Priority numbers range from 1 to 255 inclusively, with 1 denoting the highest priority. Any two processes having the same priority number are scheduled in the same subqueue or portion of the master queue. The System Supervisor User also can specify the relative ranks of processes having identical numbers within a linear subqueue, and can assign them specific ranks in the master queue.

With each circular subqueue of priority P, can be associated another subqueue of priority P-1 called the input/output subqueue. (As noted above, these are the *subqueues* within the CS and DS classifications.) If any process is detected as being input/output bound, the system switches it from the normal (compute-bound) subqueue to the input/output subqueue. (An input/output bound process is defined as one which, for a specified number of consecutive times, has been suspended for input/output instead of being timed out at the end of its time-quantum.) This gives the process a higher priority until it is no longer input/output bound. A process is considered to be compute-bound when, for a specified number of consecutive times, it reaches the end of its time-quantum.

A System Supervisor User can control the master queue in the following ways:

- Open and close any subqueue (except the ES subqueue), making it available or unavailable to users. This technique, described below, enables the user to regulate different operations on-line, and to select them according to system load or time of day.
- Modify the time-quantum for any circular subqueue, on-line, also discussed below.
- Modify the master queue at configuration time by adding or deleting non-standard subqueues. The number of standard subqueues, however, cannot be changed. (This option is described in section IV.)
- Examine the master queue, as described on the following page.

Closing a Subqueue

A System Supervisor User can close a subqueue, so that no further processes can be scheduled on that subqueue. (For example, the CS subqueue is typically closed to bar the use of interactive terminals requesting this subqueue during a period when batch job load is particularly heavy.) The processes already in the subqueue at the time it is closed remain there. But, if an attempt is made to schedule other processes in the subqueue from this point on, they are instead placed in the ES subqueue (which cannot be closed). To close a subqueue, this command is used:

:BAR subqueueuname

subqueueuname The first letter in the subqueue priority-class reference. (Required parameter.)

EXAMPLE:

To close the DS subqueue, the user enters:

:BAR D

When a session attempts to access the CS subqueue and that subqueue is closed, the session is not accepted.

Re-Opening a Subqueue

To re-open a closed subqueue so that it can once more be accessed, the user enters:

:ALLOW subqueueuname

subqueueuname The first letter in the subqueue priority-class reference. (Required parameter.)

EXAMPLE:

To re-open the CS subqueue, the user enters:

:ALLOW C

Changing a Circular Subqueue Quantum

The System Supervisor User can change the value of the quantum of a circular subqueue by issuing `:QUANTUM` command. This command is used primarily in on-line tuning of the system response time to best accommodate the current load.

NOTE: Misuse of this command can significantly degrade system operating efficiency.

The command format is

$$:QUANTUM \left\{ \begin{array}{c} \$ \\ * \end{array} \right\} subqueue\text{name}, time$$

<i>\$</i>	Specifies the upper subqueue when the subqueue name references a pair of subqueues (input/output-bound versus compute-bound subqueues). (Optional parameter.)
<i>*</i>	Specifies the lower subqueue when the subqueue name references a pair of subqueues. (Required parameter if \$ is omitted.)
<i>subqueue name</i>	The first letter in the name of the subqueue. (Required parameter.)
<i>time</i>	The new time-quantum, in milliseconds, for this subqueue. This value must lie between 10 milliseconds and 65,536 milliseconds. (Required parameter.)

EXAMPLE:

To change the quantum of the CS subqueue (compute-bound subqueue) to 100 milliseconds, the user enters:

```
_QUANTUM *C,100
```

Rescheduling Jobs/Sessions

A user with System Supervisor Capability can change the subqueue assignment of any job or session submitted to MPE/3000, through the `:PUTJOB` command. This command can apply to all processes within the job/session or to the main process only. If the job/session is rescheduled in a linear subqueue, the operator can also specify its rank within that subqueue relative to other jobs/sessions. The `:PUTJOB` command format is

`:PUTJOB jsid,process,subqueuename[,rank]`

jsid The name of the job or session, as defined by the user, or the job/session number assigned by MPE/3000 when entered, in these formats:

For jobs: [jobname,] username.acctname or #Jnnn

For sessions: [sessionname,] username.acctname or #Snnn

process ALL, to reschedule all processes within the job/session (in the same tree-structure).

MAIN, to reschedule only the job/session main process.
(Required parameter.)

*subqueue*name The name of the subqueue in which the job/session is to be rescheduled (for example, BS, CS, or DS). (Required parameter.)

rank The relative numerical order of the job/session within the subqueue, if this is a linear subqueue. If this parameter is omitted, a default value of 0 (the last place in the subqueue) is assumed. (Optional parameter.)

EXAMPLE:

To reschedule a job named JMS so that it becomes the fifth process in the DS subqueue, the user enters the following command. All processes within the job are rescheduled:

`:PUTJOB JMS, ALL, DS, 5`

Displaying Subqueue Information

A System Supervisor User can display the process identification numbers (PIN's) and types of processes within a particular subqueue, or the names, priorities, and types of all subqueues in the master queue, with the `:SHOWQ` command.

`:SHOWQ [subqueueename]`

subqueueename The first letter in the name of the subqueue to be displayed. If this parameter is omitted, the names of all subqueues in the master queue are to be displayed. (Optional parameter.)

EXAMPLE:

To display the contents of the BS subqueue, the user enters:

`:SHOWQ BS`

As a result, the following type of information is printed. (The three-digit numbers are PIN's, M indicates a job or session main process, and J, and S, indicate jobs and sessions, respectively.)

Priority	Processes
BS* 100	058 J
	015 (M) S
	016 S
	127 (M) J

To display the overall structure of the master queue's subqueues, the user enters:

`:SHOWQ`

The following type of information is output, where asterisks indicate standard subqueues:

Priority	Subqueue
30	AS* L
25	XS C
60	SP C
100	BS* L

EXECUTING PROGRAMS

Programs that have been compiled and prepared, and that therefore exist on program files, can be executed by the `:RUN` command. In this command, a user with System Supervisor Capability can enter two parameters (*subqueue*name and *rank*) not available to other users.

```
:RUN progfile[,entrypoint]
    [;NOPRIV]
    [;LMAP]
    [;MAXDATA = segsize]
    [;PARM = parameternum]
    [;STACK = stacksize]
    [;DL = dlsiz]
    [;LIB = library]
    [;SQ = subqueuename[,rank] ]
```

<i>progfile</i>	The name of the program file that contains the prepared program. (Required parameter.)
<i>entrypoint</i>	The program entry-point where execution is to begin. This may be the primary entry-point of the program or any secondary entry-point in the program's outer block. If this parameter is omitted, the primary entry-point (where execution normally begins) is assigned by default. (Optional parameter.)
<i>NOPRIV</i>	A declaration that the program will be placed in <i>non-privileged</i> mode, used only by programs prepared with the privileged-mode capability. Normally, program segments containing privileged instructions are executed (run in privileged mode) <i>only</i> if the program was <i>prepared</i> with the privileged-mode (PM) capability class. (A program containing legally-compiled privileged code, placed in the non-privileged mode, aborts when an attempt is made to execute it.) If <i>NOPRIV</i> is specified in the <code>:PREP</code> command, all program segments are placed in the <i>non-privileged</i> mode, with <i>NOPRIV</i> overriding the PM capability class. (Library segments, however, are not affected, since their mode is determined independently.) (Optional parameter.)
<i>LMAP</i>	An indication that a listing of the <i>allocated</i> (loaded) program will be produced on the job/session listing device. If this parameter is omitted, the allocated program is not listed. (Optional parameter.)
<i>segsize</i>	Maximum stack area (Z-DL) size permitted, in words. This parameter is included if the user expects to change the value specified when the program file was prepared. (Optional parameter.)

<i>parameternum</i>	A value that can be passed to the user's program as a general parameter for control or other purposes; when the program is executed, this value can be retrieved from the address Q (initial) -4, where Q (initial) is the Q-address for the outer block of the program. The value can be an octal number or a signed or unsigned decimal number. If <i>parameternum</i> is omitted, the Q (initial) -4 address is filled with zeros. (Optional parameter.)
<i>stacksize</i>	The initial size of the user's local data (Z-Q (initial)) area in the stack, in words. This parameter overrides the stacksize estimated by the segmenter, which applies if the <i>stacksize</i> parameter is omitted. (The default is a function of estimated stack requirements for each program unit in the program.) (Optional parameter.)
<i>dlsiz</i>	The initial DL-DB area size, in words. If this value is omitted, a value estimated by the segmenter applies. (Optional parameter.)
<i>library</i>	The order in which applicable segmented procedure libraries are searched to satisfy external references during allocation, where: G = Group library, followed by account public library, followed by system library. P = Account public library, followed by system library. S = System Library. If no parameter is specified, <u>S</u> is assumed. (Optional parameter.)
<i>subqueue</i> <i>name</i>	The name of the subqueue in which the process is entered. If this parameter is omitted, the subqueue of the father (main) process is assigned by default. Within a subqueue, the new process is given the last position, unless a different position is specified by the <i>rank</i> parameter. (Optional parameter.)
<i>rank</i>	The relative rank of the program within a linear subqueue, specified as a one-digit decimal integer. If this parameter is omitted, a rank of 0 (the last position) is assigned. (Optional parameter.)

If values for *segsiz*, *stacksiz*, and *dlsiz* are explicitly specified in the :RUN command, they override such values assigned at preparation time (which are recorded in the program file).

EXAMPLES:

The following command executes a program existing on the program file PROG3, with no special parameters specified. (All default values are used.)

⌘RUN PROG3

The next command executes a program existing on the program file PROG4, beginning at the entry-point SECLAB. This program, prepared with privileged-mode capability, will run in non-privileged mode. The process under which it runs is scheduled as the sixth process in the CS subqueue:

⌘RUN PROG4, SECLAB, NOPRIV; SQ=CS, 6

CREATING PROCESSES

A user with the Standard MPE/3000 Capabilities, plus the System Supervisor Capability, can create a process by calling the CREATE intrinsic. (He need not have the *MPE/3000 Process-Handling Optional Capability* normally required for the CREATE intrinsic. He does, however, require that capability to run the process after it has been created, by calling the ACTIVATE intrinsic.)

The CREATE intrinsic loads the program to be run by the new process into virtual memory, creates the new process as the son of the calling process, initializes its data stack, schedules the process, and returns its process identification number (PIN) to the calling process.

The CREATE intrinsic is written in the following format:

<i>PROCEDURE</i>	<i>CREATE (progrname, entryname, pin, param, flags, stacksize, dlsiz, maxdata, priorityclass, rank;)</i>
<i>VALUE</i>	param, stacksize, dlsiz, priorityclass, maxdata, flags, rank;
<i>LOGICAL</i>	priorityclass, flags;
<i>INTEGER</i>	stacksize, dlsiz, maxdata, pin, param, rank;
<i>BYTE ARRAY</i>	progrname, entryname;
<i>OPTION VARIABLE, EXTERNAL;</i>	

The parameters for this intrinsic call are as follows. All parameters except *progrname* and *pin* are optional.

<i>progrname</i>	A byte array, containing a string terminated by a blank, specifying the name and optionally, the account and group of the file containing the program to be run.
<i>entryname</i>	A byte array, containing a string terminated by a blank, specifying the entry-point (label) in the program where execution is to begin when the process is activated. The <i>primary</i> entry-point in the program can be specified in this array by a blank character alone. If <i>entryname</i> is omitted, the primary entry-point is specified by default.
<i>pin</i>	A word in which the PIN of the new process is returned to the calling process. This PIN is used in other intrinsics to reference the new process. The PIN can range from 1 to 255. (If an error is detected, a PIN of zero is returned.)

param	A word used to transfer control information to the new process. Any instruction in the new process can access this information in location (Q-4). If param is omitted, this word is filled with zeros.
flags	A word whose bits, if on, specify loading options: Bit 15 = <i>ACTIVE Bit</i>. If this bit is <i>on</i>, MPE/3000 re-activates the calling process (father) when the new process terminates. If this bit is <i>off</i>, the calling process is not activated at that time. The default setting is <i>off</i>. Bit 14 = <i>LOADMAP Bit</i>. If this bit is <i>on</i>, a listing of the allocated (loaded) program is produced on the job/session listing device. This map shows CST entries used by the new process. If this bit is <i>off</i>, no map is produced. The default setting is <i>off</i>. Bit 13 = (Not used.) Bit 12 = <i>NOPRIV Bit</i>. If this bit is <i>on</i>, the program is loaded in <i>non-privileged mode</i>. If this bit is <i>off</i>, the program is loaded in the mode specified when the program file was prepared. The default setting is <i>off</i>. Bits 11-10 = <i>LIBSEARCH Bits</i>. These bits denote the order in which libraries are to be searched for the program: 00 = System Library 01 = Account Public Library, followed by System Library. 10 = Group Library, followed by Account Public Library, followed by System Library. The default setting for these bits is 00. Bits 9-8 = (Not used by MPE/3000; the bits in this field remain as set by the user.) If <i>flags</i> is omitted, all default values noted are taken.
stacksize	An integer (Z-Q) denoting the number of words assigned to the local stack area (the area bounded by the Q and Z registers). The default value is that specified during system generation.

dlsz	An integer (DB-DL) denoting the number of words in the user-managed stack area (bounded by the DL and DB registers). The default value is zero.
maxdata	The maximum size allowed for the process' stack (Z-DL) area in words. When specified, this value overrides the one established at program-preparation time. The default value is specified during system generation.
priorityclass	A string of two ASCII characters describing the priority class (subqueue) in which the new process is scheduled. (For example, "CS," "DS," or "ES".) A process can be scheduled in the master queue by specifying x A, where x is an actual priority number. The default value is the priority of the calling process.
rank	The relative rank of the process within a linear subqueue. The rank parameter is ignored if x A (where x is an actual priority number) is specified for the <i>priorityclass</i> parameter.

When the CREATE intrinsic is called, the DB Register must be pointing at the user's stack.

The CREATE intrinsic returns one of the following condition codes to the calling process:

CCE	Request was granted; the new process is created.
CCG	Request was granted; the <i>stacksize</i> and/or <i>maxdata</i> parameters given were illegal, but system default values were used.
CCL	Request was rejected because of illegal <i>progrname</i> or <i>entryname</i> parameters; a PIN of zero is returned.

The creating process is aborted if

- The user does not have the System Supervisor or *Process-Handling Optional Capability*.
- A required parameter (*progrname* or *pin*) is omitted.
- A value specified for a parameter is not within the required range.
- An illegal value (a non-existent subqueue) is specified for the *priorityclass* parameter.

EXAMPLE:

Suppose the user wants to create a process that, when activated, runs a program in the file PROGA, beginning at the entry-point START1. He also wants the libraries searched in the following order: account public library, followed by system library. The process' PIN is to be returned to the word PINRET. The process is to be assigned the priority class CS. The user enters the following intrinsic call. (The byte array PROG contains "PROGA", the byte array ENTER contains "START1," and the tenth bit of the word FLAGGER is on.)

CREATE (PROG,ENTER,PINRET,,FLAGGER,,,,,"CS");

The PIN assigned to the processes is 026, returned to the word PINRET.



RESCHEDULING PROCESSES

When a process is created, it is scheduled on the basis of a priority-class assigned by its father. After this point, its priority-class can be changed at any time through the GETPRIORITY intrinsic. (This intrinsic is restricted to users with either the System Supervisor Capability, or the *MPE/3000 Process-Handling Optional Capability*.) The intrinsic can be called from a process to change the priority of that process or of one of its sons — but not the priority of that process' father.

The format of the GETPRIORITY intrinsic call is

```
PROCEDURE GETPRIORITY (pin, priorityclass, rank);

VALUE    pin, priorityclass, rank;

LOGICAL  priorityclass;

INTEGER  pin; rank;

OPTION VARIABLE, EXTERNAL;
```

The parameters for the GETPRIORITY intrinsic are

<i>pin</i>	An integer denoting the process whose priority-class is to be changed. If this is a son process, the integer is the process' PIN. If this is the calling process, the integer is zero.
<i>priorityclass</i>	A string of two ASCII characters describing the priority-class (subqueue) in which the process is rescheduled. (For example "AS," "BS," "CS," "DS," or "ES".) A process can be rescheduled in the main master queue by specifying "xA," where <i>x</i> is an actual priority number. If the <i>priorityclass</i> parameter is omitted, the intrinsic has no effect.
<i>rank</i>	The relative rank of the process within a linear subqueue. This parameter is ignored if "xA" is specified for the <i>priorityclass</i> parameter. If this parameter is omitted, a default value of 0 (the last position in the subqueue) is assigned.

The GETPRIORITY intrinsic returns one of the following condition codes:

CCE	Request granted; the process was rescheduled, or the <i>priorityclass</i> parameter was omitted (in which case, no action is taken).
CCG	(Not returned by this intrinsic.)
CCL	Request denied because of illegal <i>priorityclass</i> (referencing a non-existent subqueue), or because an illegal PIN was specified.

The process is aborted if the user does not have the System Supervisor or Process-Handling Capability.

EXAMPLE:

To reschedule a process with the priority class "ES," a user issues the following call from within that process.

GETPRIORITY (0,"ES");

PERMANENTLY ALLOCATING PROGRAMS AND PROCEDURES

A user with System Supervisor Capability can enhance system efficiency by storing program files (sets of one or more segments) or procedures (individual segments) permanently in virtual memory. This is, in effect, a permanent allocation of the programs or procedures so that they always reside in virtual memory regardless of whether they are being referenced. Since allocation of a code segment always results in the allocation of external segments referenced by it, when a program or procedure is permanently allocated, all of its external routines are also permanently allocated.

By permanently allocating large, frequently-used routines, a System Supervisor User can greatly reduce the system overhead that would otherwise be used in continually inserting and withdrawing them from virtual memory. Additionally, permanent allocation reduces the number of segment linkage operations required.

Allocation

To permanently allocate a program file or procedure, the user enters the **:ALLOCATE** command:

:ALLOCATE *code,name*

code A directive specifying whether a program file or a code segment containing a procedure is to be allocated.

If PROGRAM is entered for this parameter, a program file (indicated by the *name* parameter) that resides in the system file domain is allocated.

If PROCEDURE is entered for this parameter, the code segment containing the specified procedure (indicated by the *name* parameter) that resides in the system segmented library (SL.PUB.SYS) is allocated. (Required parameter.)

name The name of the program file or procedure to be allocated. If PROGRAM is specified for the *code* parameter, the *name* parameter should be written in the following format (to indicate a program file named *filename* in the public group of the system account):

filename.PUB.SYS

If PROCEDURE is specified for the *code* parameter, the *name* parameter must be a procedure name. (Required parameter.)

EXAMPLE:

To permanently allocate a program residing on a program file named RUNNER, the user enters:

:ALLOCATE PROGRAM, RUNNER.PUB.SYS

To permanently allocate a procedure identified as PROC1, the user enters:

:ALLOCATE PROCEDURE, PROC1

De-Allocation

To remove from virtual memory a program file or procedure segment that has been permanently allocated, a System Supervisor User issues the `:DEALLOCATE` command. The program file or procedure segment is de-allocated immediately if it is not being used by a process; otherwise, it is de-allocated as soon as it is no longer in use. The `:DEALLOCATE` command format is

`:DEALLOCATE code,name`

code A directive specifying whether a program file or procedure code segment is to be de-allocated.

If PROGRAM is entered for this parameter, a program file (indicated by the *name* parameter) is de-allocated.

If PROCEDURE is entered for this parameter, a code segment (indicated by the *name* parameter) is de-allocated. (Required parameter.)

name The name of the program file or procedure to be de-allocated. If PROGRAM is specified for the *code* parameter, the name parameter should be written in the following format (to indicate a program file named *filename* in the public group of the system account):

filename.PUB.SYS.

If PROCEDURE is specified for the *code* parameter, the *name* parameter must be the procedure name. (Required parameter.)

TRANSMITTING MESSAGES

The user with System Supervisor Capability can transmit messages from his job or session to other jobs or sessions currently running, by entering the :TELL command. The message is printed on the receiving job or session list device as soon as possible, but it does not interrupt reading or writing in progress. The format of the :TELL command is

:TELL [jsname], username [.acctname]; message

<i>jsname</i>	The name of the job or session to receive the message. If omitted, no job or session name is assigned. (Optional parameter.)
<i>username</i>	The name of the user whose job or session is to receive the message, as recorded in the system. When the <i>jsname</i> parameter is omitted, the message is sent to all running jobs/sessions associated with the user <i>username</i> . (Required parameter.)
<i>acctname</i>	The name of the receiving job or session's account. If omitted, the transmitting job/session's log-on account is assumed. (Optional parameter.)
<i>message</i>	The message, comprised of any string of ASCII characters. The message is terminated by the end of the record on which it appears, or by a carriage return. (Required parameter.)

The message appears on the receiving list device in this format:

FROM/ [jsname], username [.acctname] /message

<i>jsname</i> <i>username</i> <i>acctname</i>	}	The names of the transmitting job/session and user, and the name of the account under which they are running if different from that of the receiving job or session.
<i>message</i>		The message, in ASCII characters.

If the job/session or user designated to receive the message is not running, the transmitting session will receive a message indicating this.

EXAMPLE:

A user identified as SMITH, running a session named SMITHJOB, wants to send a message to a user identified as BROWN, running a session named BROWNJOB. (Both users are logged-on under the same account.) SMITH enters:

!TELL BROWNJOB, BROWN; SYSTEM WILL BE SHUT DOWN AT 4:30 TODAY.

BROWN receives:

FROM/SMITHJOB, SMITH/ SYSTEM WILL BE SHUT DOWN AT 4:30 TODAY.

APPENDIX A

ASCII Character Set

Character	Octal Value (Left Byte)	Octal Value (Right Byte)	Meaning
NUL	000000	000000	Null (Normally ignored by MPE/3000)
SOH	000400	000001	Start of heading
STX	001000	000002	Start of text
ETX	001400	000003	End of text
EOT	002000	000004	End of transmission
ENQ	002400	000005	Enquiry
ACK	003000	000006	Acknowledge
BEL	003400	000007	Bell
BS	004000	000010	Backspace
HT	004400	000011	Horizontal tabulation
LF	005000	000012	Line feed (Normally ignored by MPE/3000)
VT	005400	000013	Vertical tabulation
FF	006000	000014	Form feed
CR	006400	000015	Carriage return
SO	007000	000016	Shift out
SI	007400	000017	Shift in
DLE	010000	000020	Data link escape
DC1	010400	000021	Device control 1
DC2	011000	000022	Device control 2
DC3	011400	000023	Device control 3
DC4	012000	000024	Device control 4
NAK	012400	000025	Negative acknowledge
SYN	013000	000026	Synchronous idle
ETB	013400	000027	End of transmission block

(Normally
ignored by
MPE/3000)

Character	Octal Value (Left Byte)	Octal Value (Right Byte)	Meaning
CAN	014000	000030	Cancel
EM	014400	000031	End of medium
SUB	015000	000032	Substitute
ESC	015400	000033	Escape
FS	016000	000034	File separator
GS	016400	000035	Group separator
RS	017000	000036	Record separator
US	017400	000037	Unit separator
SP	020000	000040	Space
!	020400	000041	Exclamation point
”	021000	000042	Quotation mark
#	021400	000043	Number sign
\$	022000	000044	Dollar sign
%	022400	000045	Percent sign
&	023000	000046	Ampersand
,	023400	000047	Apostrophe
(	024000	000050	Opening parenthesis
)	024400	000051	Closing parenthesis
*	025000	000052	Asterisk
+	025400	000053	Plus
,	026000	000054	Comma
-	026400	000055	Hyphen (Minus)
.	027000	000056	Period (Decimal)
/	027400	000057	Slant
0	030000	000060	Zero
1	030400	000061	One
2	031000	000062	Two
3	031400	000063	Three
4	032000	000064	Four
5	032400	000065	Five
6	033000	000066	Six
7	033400	000067	Seven

Character	Octal Value (Left Byte)	Octal Value (Right Byte)	Meaning
8	034000	000070	Eight
9	034400	000071	Nine
:	035000	000072	Colon
;	035400	000073	Semi-colon
<	036000	000074	Less than
=	036400	000075	Equals
>	037000	000076	Greater than
?	037400	000077	Question mark
@	040000	000100	Commercial at
A	040400	000101	Uppercase A
B	041000	000102	Uppercase B
C	041400	000103	Uppercase C
D	042000	000104	Uppercase D
E	042400	000105	Uppercase E
F	043000	000106	Uppercase F
G	043400	000107	Uppercase G
H	044000	000110	Uppercase H
I	044400	000111	Uppercase I
J	045000	000112	Uppercase J
K	045400	000113	Uppercase K
L	046000	000114	Uppercase L
M	046400	000115	Uppercase M
N	047000	000116	Uppercase N
O	047400	000117	Uppercase O
P	050000	000120	Uppercase P
Q	050400	000121	Uppercase Q
R	051000	000122	Uppercase R
S	051400	000123	Uppercase S
T	052000	000124	Uppercase T
U	052400	000125	Uppercase U
V	053000	000126	Uppercase V
W	053400	000127	Uppercase W

Character	Octal Value (Left Byte)	Octal Value (Right Byte)	Meaning
X	054000	000130	Uppercase X
Y	054400	000131	Uppercase Y
Z	055000	000132	Uppercase Z
[	055400	000133	Opening bracket
\	056000	000134	Reverse slant
]	056400	000135	Closing bracket
^	057000	000136	Circumflex
_	057400	000137	Underscore
	060000	000140	Grave accent
a	060400	000141	Lowercase a
b	061000	000142	Lowercase b
c	061400	000143	Lowercase c
d	062000	000144	Lowercase d
e	062400	000145	Lowercase e
f	063000	000146	Lowercase f
g	063400	000147	Lowercase g
h	064000	000150	Lowercase h
i	064400	000151	Lowercase i
j	065000	000152	Lowercase j
k	065400	000153	Lowercase k
l	066000	000154	Lowercase l
m	066400	000155	Lowercase m
n	067000	000156	Lowercase n
o	067400	000157	Lowercase o
p	070000	000160	Lowercase p
q	070400	000161	Lowercase q
r	071000	000162	Lowercase r
s	071400	000163	Lowercase s
t	072000	000164	Lowercase t
u	072400	000165	Lowercase u
v	073000	000166	Lowercase v
w	073400	000167	Lowercase w

Character	Octal Value (Left Byte)	Octal Value (Right Byte)	Meaning
x	074000	000170	Lowercase x
y	074400	000171	Lowercase y
z	075000	000172	Lowercase z
{	075400	000173	Opening (left) brace
	076000	000174	Vertical line
}	076400	000175	Closing (right) brace
~	077000	000176	Tilde
DEL	077400	000177	Delete



APPENDIX B

Summary of Commands

SYSTEM MANAGER CAPABILITY COMMANDS

Commands	Parameters	Function	When Issued
:ALTACCT	<i>acctname</i> [;PASS=[password]] [;CAP=[capabilitylist]] [;MAXPRI=[subqueueuname]] [;LOCATTR=[localattribute]]	Changes an account's characteristics.	Job, session, or break.
:ALTGROUP	<i>groupname</i> [:PASS=password]	Changes a group's attributes.	Job, session, or break.
:ALTUSER	<i>username</i> [;PASS=[password]] [;CAP=[capabilitylist]] [;MAXPRI=[subqueueuname]] [;LOCATTR=[localattribute]] [;HOME=[homegroupname]]	Changes a user's attributes.	Job, session, or break.
:LISTACCT	$\left[\begin{array}{c} @ \\ acctname \end{array} \right] [,listfile]$	Lists attributes of an account.	Job, session, or break.
:LISTGROUP	$\left[\begin{array}{c} @ \\ groupname \end{array} \right] [,listfile]$	Lists attributes of a group.	Job, session, or break.

Commands	Parameters	Function	When Issued
<i>:LISTUSER</i>	$\left[\begin{array}{l} @ \\ username \end{array} \right] [,listfile]$	Lists attributes of a user.	Job, session, or break.
<i>:NEWACCT</i>	<i>acctid, mgrname</i> [;PASS=[password]] [;CAP=[capabilitylist]] [;MAXPRI=[subqueueuname]] [;LOCATTR=[localattribute]]	Creates a new account.	Job, session, or break.
<i>:NEWGROUP</i>	<i>groupid</i> [;PASS=[password]]	Creates a new group.	Job, session, or break.
<i>:NEWUSER</i>	<i>userid</i> [;PASS=[password]] [;CAP=[capabilitylist]] [;MAXPRI=[subqueueuname]] [;LOCATTR=[localattribute]] [;HOME=[homegroupname]]	Creates a new user.	Job, session, or break.
<i>:PURGEACCT</i>	<i>acctname</i>	Deletes an account.	Job, session, or break.
<i>:PURGEGROUP</i>	<i>groupname</i>	Deletes a group.	Job, session, or break.
<i>:PURGEUSER</i>	<i>username</i>	Deletes a user.	Job, session, or break.

SYSTEM SUPERVISOR CAPABILITY COMMANDS

Commands	Parameters	Function	When Issued
:ALLOCATE	<i>code,name</i>	Permanently allocates a program or procedure in virtual memory.	Job or session.
:ALLOW	<i>subqueueuname</i>	Opens a subqueue.	Job, session, or break.
:BAR	<i>subqueueuname</i>	Closes a subqueue.	Job, session, or break.
:DEALLOCATE	<i>code,name</i>	Removes a program or procedure from virtual memory.	Job or session.
:PUTJOB	<i>jsid,process,subqueueuname[,rank]</i>	Reschedules jobs or sessions.	Job, session, or break.
:QUANTUM	$\left\{ \begin{array}{c} \$ \\ * \end{array} \right\} \text{ subqueueuname,time}$	Changes a circular subqueue time-quantum.	Job, session, or break.
:RUN	<i>progfile[,entrypoint]</i> [;NOPRIV] [;LMAP] [;MAXDATA=segsize] [;PARM=parameternum] [;STACK=stacksize] [;DL=dlsizel] [;LIB=library] [;SQ=subqueueuname[,rank]]	Loads and executes a program.	Job or session.
:SHOWQ	<i>[subqueueuname]</i>	Displays subqueue information.	Job, session, or break.

Commands	Parameters	Function	When Issued
<i>:SYSDUMP</i>	<i>dumpfile[,auxlistfile]</i>	Copies MPE/3000 to tape.	Job or session.
<i>:TELL</i>	<i>[jsname],username[.acctname]; message</i>	Sends a message to a user's job or session.	Job, session, or break.

APPENDIX C

Summary of Intrinsic Calls

The two intrinsic calls available to the user with System Supervisor Capability are summarized below. For each intrinsic, the complete declaration head appears; the intrinsic call format is distinguished from the remainder of the head by a box. The function of the intrinsic is described. Optional parameters are *bold face* in the intrinsic head. All condition codes that can be returned by the intrinsic are listed.

The intrinsic error-code numbers are also presented. (These are the numbers that appear in the abort-error messages generated when errors are encountered in the corresponding intrinsics.) The functional categories represented by these error numbers are

Error Number Range	Functional Category
1–29	File Management
30–39	Resource Management
40–49	System Timer (Clock)
50–59	Traps
60–79	Utility Routines
80–99	Program Management
100–119	Process Control
120–129	Scheduling
130–149	Data Segments
200–209	Special Utilities

Where intrinsics have special attributes, those attributes are noted as follows:

- Uncallable intrinsics (those that can only be invoked by the user in privileged mode) are indicated by an asterisk (*).
- Intrinsics that can be called in privileged mode even though the user does not have the appropriate capability are denoted by a plus sign (+).
- Intrinsics that can be called by a process when the DB register is not pointing to that process stack are denoted by a dollar sign (\$).

PROCESS-HANDLING OPTIONAL CAPABILITY INTRINSICS

PROCEDURE

<i>CREATE</i> (<i>progrname</i> , <i>entryname</i> , <i>pin</i> , <i>param</i> , <i>flags</i> , <i>stacksize</i> , <i>dlsiz</i> e, <i>maxdata</i> , <i>priorityclass</i> , <i>rank</i>);
--

VALUE *param*,*stacksize*,*dlsiz*e,*priorityclass*,*maxdata*,*flags*,*rank*;

LOGICAL *priorityclass*,*flags*;

INTEGER *stacksize*,*dlsiz*e,*maxdata*,*pin*,*param*,*rank*;

BYTE ARRAY *progrname*,*entryname*;

OPTION VARIABLE, *EXTERNAL*;

FUNCTION: *Creates a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *entryname*,*param*,*flags*,*stacksize*,*dlsiz*e,*maxdata*,*priorityclass*,
rank

CONDITION CODES: *CCE*, *CCG*, *CCL*

ERROR CODE: *100*

SPECIAL ATTRIBUTES: *None*

PROCEDURE

<i>GETPRIORITY (pin, priorityclass, rank) ;</i>

VALUE *pin, priorityclass,rank*

LOGICAL *priorityclass;*

INTEGER *pin, rank;*

OPTION VARIABLE, EXTERNAL;

FUNCTION: *Reschedules a process.*

TYPE: *None*

OPTIONAL PARAMETERS: *priorityclass, rank*

CONDITION CODES: *CCE, CCL*

ERROR CODE: *120*

SPECIAL ATTRIBUTES: *\$*

APPENDIX D

Driver Names, Types, Subtypes, and Sizes

Device	Part No.	Driver Name	Type	Sub-Type	Record Width (octal words)	Approximate Virtual-Memory Disk-Resident Space Used (octal words.)	Approximate Main Memory Resident Space Used (octal words.)
System Clock/ Console Interface	30031A	IOCLTTY	16		36	250	250
ASR 33,35	30124A			0			
DCT 500, 10 cps				1			
DCT 500, 15 cps	30114A			2			
DCT 500, 30 cps				3			
HP 2600A, 10 cps				4			
HP 2600A, 15 cps				5			
HP 2600A, 30 cps	30123A			6			
HP 2600A, 60 cps				7			
HP 2600A, 120 cps				8			
HP 2600A, 240 cps				9			
Asynchronous Terminal Controller	30032A	IOTERM0	16		36	250	250
Hardwired Terminal				0			
Terminal Interfaced Over 103A modem.				1			

Device	Part No.	Driver Name	Type	Sub-Type	Record Width (octal words)	Approximate Virtual-Memory Disk-Resident Space Used (octal words.)	Approximate Main Memory Resident Space Used (octal words.)
Nine-channel Magnetic Tape Driver	30115A-100	IOTAPE0	24	0	128	1100	IOQH = 6 DIT = 5 for each unit SIO = 35
Fixed-Head Disc 1 megabyte 2 megabyte 4 megabyte	30103A -001 -002	IOFDISK0	1	 0 1 2	128	None (Driver resides in main memory.)	IOQH = 6 DIT = 6 SIO = 24 Remainder of Driver = 600
Cartridge Disc Lower Platter only Upper Platter only.	30110A	IOMDISK	0	2 1 0	128	None (Driver resides in main-memory.)	IOQH = 6 DIT = 17 for each unit SIO = 12 Remainder of Drivers = 1100
Disc File	30102A	IOMDISK0	0	3	128	None (Driver resides in main-memory.)	IOQH = 6 DIT = 17 for each unit SIO = 12 Remainder of Driver = 1100
Card Reader	30106A, 30107A	IOCARD0	8	0	40	700	IOQH = 6 DIT = 7 SIO = 10

Device	Part No.	Driver Name	Type	Sub-Type	Record Width (octal words)	Approximate Virtual-Memory Disk-Resident Space Used (octal words.)	Approximate Main Memory Resident Space Used (octal words.)
Line Printer	30108A-001 30109A-001	IOLPRT0	32	0	66	1100	IOQH = 6 DIT = 17 SIO = 24
Card Punch	30112A	IOCDPN0	33	0	40	700	IOQH = 6 DIT = 4 SIO = 132

APPENDIX E

Default Parameter Settings For Configuration

Estimated Number of Users	8	12	16	24	32	
Memory Size						
Configurator Question	32	40	48	56	64	Unit of Measure
CST = xxx.?	192	192	192	256	256	entries
DST = xxx.?	128	160	192	256	320	entries
PCB = xxx.?	48	64	96	112	128	entries
I/O QUEUE = xxx.?	24	36	48	56	64	entries
TERMINAL BUFFERS = xxx.?	24	36	48	56	64	buffers
IOCB = xxx.?	16	24	32	40	48	entries
ICS = xx.?	128	128	192	224	224	words
UCOP REQUEST QUEUE = xx?	8	12	16	24	32	entries
TIMER REQUEST LIST = xx.?	24	36	48	56	64	---
NO. OF SECONDS TO LOGON = xxx.?	120	120	120	120	120	seconds
NO. OF RINS = xxx.?	128	192	256	384	512	---
MAX # OF GLOBAL RINS = xxx.?	32	48	64	96	128	---
MAX # OF JOBS = xxx.?	16	20	25	36	48	---
MAX # OF CONCURRENT RUNNING JOBS = xxx.?	12	15	20	30	40	---
VIRTUAL MEMORY = xxxx.?	1024	1536	2048	2560	3072	sectors
DIRECTORY = yyyy, MAX = xxxx.?	256	256	384	512	512	sectors
TIME-QUANTUM—TIME-SHARE SUBQUEUE = xxx.?	100	100	100	100	100	milliseconds
TIME-QUANTUM—BATCH SUBQUEUE = xxx.?	350	350	350	350	350	milliseconds
# OF MAKE-PRESENT LOOKAHEADS = x.?	0	0	0	0	0	---
MAX CODE SEGMENTS SIZE = xxxx.?	3072	3072	3072	4096	4096	words

Estimated Number of Users	8	12	16	24	32	
Memory Size						
Configurator Question	32	40	48	56	64	Unit of Measure
MAX # OF CODE SEGMENTS/PROCESS = xxx.?	40	40	40	40	40	---
MAX STACK SIZE = xxxxx.?	16384	16384	24576	32600	32600	words
MAX EXTRA DATA SEG SIZE = xxxxx.?	2048	2048	3072	4096	4096	words
MAX # OF EXTRA DATA SEGMENTS/PROCESS = x.?	2	2	3	4	4	---
STD STACK SIZE = xxxxx.?	800	800	800	800	800	words
ALLOWED OVERFLOWS = xx.?	2	2	2	2	2	---

APPENDIX F

MPE/3000 Table Names and Sizes

MAIN-MEMORY RESIDENT TABLES

<u>Table Name</u>	<u>Size</u>
Code Segment Table (CST)	2 words/entry
Code Segment Table (CST) Extension	1 word/entry
Device Reference Table (DRT)	4 word/entry
Data Segment Table (DST)	2 word/entry
Interrupt Control Stack (ICS)	64 words + configuration value
Input/Output Queue (IOQ)	11 words/entry
Input/Output Queue (IOQ) Head	6 words + (device information table size) x (number of units for each DRT defined).
Input/Output Control Block (IOCB)	2 words/entry
Job Process Cutoff Table	(Maximum number of running jobs)/2 + (maximum running jobs)/16 + 4 words
Logical-Physical Device Table	2 words/entry
Logical Segment SIR	1/2 word/entry
Process Control Block (PCB)	11 words/entry
SIR Table	40 + (Maximum number of running jobs)
System Global Value Table	256 words
Terminal Buffers	18 words/buffer
Timer Request Table	4 words/entry



VIRTUAL-MEMORY RESIDENT TABLES

<u>Table Name</u>	<u>Size</u>
Disc Free-Space Table Buffer	256 words
Job Master Table	26 x (maximum number of jobs)
Logical Device Table	5 words per entry + 1 word
Message Table Buffer	256 words
Process-Process Communication Table	2 words/entry
Progenitor (Root Process) Request Queue	40 words
Reply Information Table	256 words
Resource Identification Number (RIN) Table	2 x (number of RIN's) + 12 x (maximum number of global RIN's) + 6
User Controller Process (UCOP) Request Queue	2 words/entry
Virtual Device Directory	31 words for each device accepting data + 31 words
Volume Table	14 words/volume defined + 1 word

APPENDIX G

MPE/3000 Segment Names and Sizes

MAIN-MEMORY RESIDENT SEGMENTS

<u>Segment Name</u>	<u>Size (in words)</u>
IOUTILITY	3334
CROUTINE	2030
MMCORER	2744
MISCO	450

DISC-RESIDENT SEGMENTS

<u>Segment Name</u>	<u>Size (in words)</u>
MMDISKR	2200
CHECKER	1054
PCREATE	2214
RINS	2464
PINT	660
MORGUE	3364
UTILITY	2340
DATASEG	2330
JOBTABLE	2274
PROCMail	1534
ABORTRAP	2504
SYSDEBUG	2744
LOADER1	3344
SYSSTOPS	660
SYSDSPY	3014
TTYINT	4220

<u>Segment Name</u>	<u>Size (in words)</u>
DEBUG	2130
SEGUTIL	620
DIRC	4000
ALLOCATE	2700
DISKSPC	1334
MESSAGE	1564
FILESYS1	4120
FILESYS2	1734
FILESYS3	1254
FILESYS4	1100
FILESYS5	2264
FILESYS6	4470
FILESYS7	2440
CIFILEB	2354
CIFILEM	744
CIRUN	1214
CIMISC	2214
CILISTF	2464
CIUTIL	644
CISUBS	2760
CIINIT	3274

APPENDIX H

Back-up of Disc Files

DUMPING FILES OFF-LINE

Any user with Standard MPE/3000 Capability can obtain a back-up copy of a particular user disc file or fileset, by copying the fileset off-line to a magnetic tape unit by issuing the :STORE command. The files are copied in a special format, along with all descriptive information (such as account name and group name), permitting them to be read back into the system later (by the :RESTORE command).

Multi-file reels and multi-reel files are both supported by :STORE and :RESTORE.

NOTE: :STORE and :RESTORE are NOT intended to be used for interchanging files between HP 3000 installations, because the content of tapes in :STORE-format is highly installation-dependent.

The :STORE command is written as follows:

:STORE [filesetlist] ;destfile

filesetlist A list of one or more files or filesets to be copied, written in this format:

[fileset[,fileset]...]

In this list, *fileset* must be written as one of the combinations noted below. If the entire *filesetlist* parameter is omitted, the default value is @ (all files of the log-on group are copied). In *filesetlist*, only the first *fileset* sub-parameter can be null. (Optional parameter.)

destfile The name of the destination file onto which the stored files are written. This can be any magnetic tape file from the output set. This file must be referenced in either the *filereference* or **formaldesignator* format. (Required parameter.)

The *fileset* parameter allows the user to request dumping of one file alone, or various sets of files. It contains three positional-fields separated by periods: the file, group, and account fields. The file field permits the user to indicate a specific file or all files within the units designated by the other fields. The group field denotes the group to which the files belong. This can be the user's log-on group, any other group, or all groups within the accounts specified by the account field. The account field indicates the account or accounts to which the groups belong. This can be the log-on account, any other account, or all accounts in the system. (To specify *all* files, groups, or accounts, the user enters the character @ in the appropriate field. The omission of an entry in the group or account field denotes the log-on group or account.) For the *fileset* parameter, the combinations of entries shown below are possible:

File Field	Group Field	Account Field	Entry Example	Meaning
<i>filename</i>	<i>groupname</i>	<i>accountname</i>	FILE.GROUP.ACCT	The file named, in the group and account designated.
<i>filename</i>	<i>groupname</i>		FILE.GROUP	The file named, in the group designated under the log-on account.
<i>filename</i>			FILE	The file named, under the log-on group.
@	<i>groupname</i>	<i>accountname</i>	@.GROUP.ACCT	All files in the group named, under the designated account.
@	<i>groupname</i>		@.GROUP	All files in the group named, under the log-on account.
@			@	All files in the log-on group.
@	@	<i>accountname</i>	@.@.ACCT	All files in all groups under the account named.
@	@		@.@	All files in all groups under the log-on account.
@	@	@	@.@.@	All user files in the system.

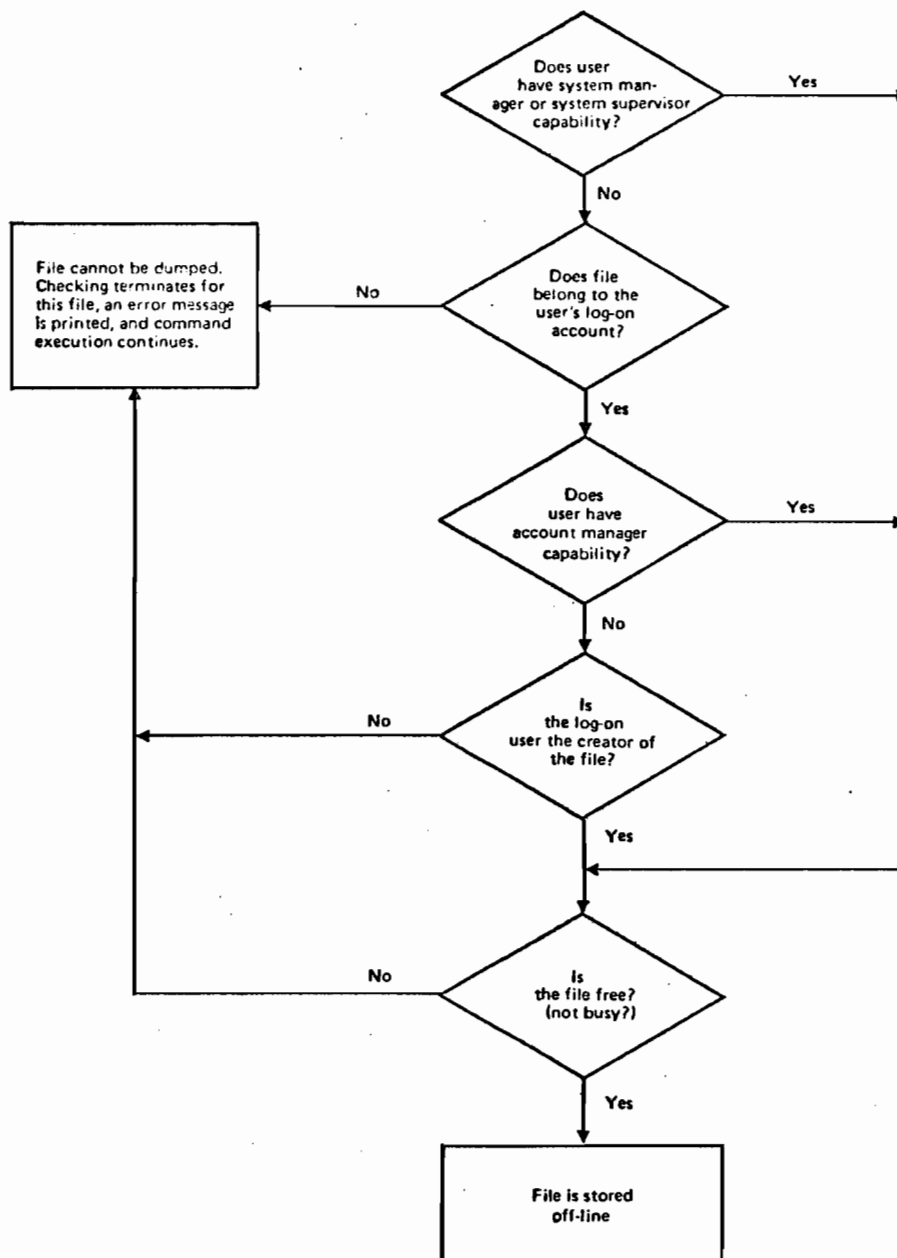
A user with the System Manager or System Supervisor Capability can dump any user file in the system. Any other user can dump any file in his log-on account, if:

1. He has the account-manager capability, *or*
2. He is the creator of that file.

No file that is currently in use by another user can be acted upon by the :STORE command. Furthermore, no file can be restored through :RESTORE if the account and group to which it belongs no longer exist in the system directory, or if an appropriate device is not available for its residence. Therefore, the user must ensure that all files to be transferred by :STORE/:RESTORE are free of these restrictions if he expects to ensure complete back-up.

While a file is being dumped, it is locked by MPE/3000 so that it cannot be altered or deleted until safely copied to tape.

The flow chart below shows the checks performed against a file to ensure its eligibility for dumping:



NOTE: Before issuing a :STORE command, the user must identify destfile as a magnetic tape file. He does this through the :FILE command. In this case, the :FILE command should be written in the following format, including no parameters other than those shown:

:FILE formaldesignator { [=filereference],NEW } [DEV=device]

The device parameter must be the device class name or logical unit number of a magnetic tape unit.

All other parameters for destfile are supplied by the :STORE command executor; if the user attempts to supply any of these himself, the :STORE command is rejected.

After the :STORE command is executed, a listing showing which files were dumped, and which were not, is printed by default on the standard list device (\$STDLIST). The :STORE command executor references this list file by the formal file designator LIST. Thus, the user can request transmission of the listing to another file by issuing a :FILE command equating LIST to another file—for example,

:FILE LIST=MYFILE; DEV=LP

The listing is generated in the following format:

FILES STORED = xxx

FILE	.GROUP	.ACCOUNT	LDN	ADDRESS
filename1	.groupname1	.acctname1	ldn1	addr1
filename2	.groupname2	.acctname2	ldn2	addr2
:				
filenamen	.groupnamen	.acctnamen	ldnn	addrn

FILES NOT STORED = yyy

FILE	.GROUP	.ACCOUNT	FILESET	CONDITION
filename1	.groupname1	.acctname1	fileset#	e/d
filename2	.groupname2	.acctname2	fileset#	e/d
:				
filenamen	.groupnamen	.acctnamen	fileset#	e/d

In this format, *xxx* is a value denoting the total number of files dumped onto tape; *yyy* denotes the number of files requested that were not dumped. The notations *filename*, *groupname*, and *acctname* under the *FILES STORED* heading name the individual files dumped, and their groups and accounts, respectively. The notation *ldn* indicates the logical device number (in decimal) of the device on which the file resides, and *addr* is the absolute address (in octal) of the file label. The notations *filename*, *groupname*, and *acctname* under the *FILES NOT STORED* heading, indicate the individual files not dumped, and their groups and accounts. The notation *fileset#* shows the number of the fileset to which the particular file belongs (relative to its position in the *filesetlist* parameter.) The notation *e/d* is an error code denoting the reason that the file was not dumped, as follows. (These errors do not abort the file storing operation, which continues.)

Value of e	Meaning	Value of d	Meaning
1	The name (indicated by <i>d</i>) could not be found in the system directory.	0	File name
		1	Group name
		2	Account name
2 (This value will not be received by users with System Manager or System Supervisor Capability.)	The log-on user or account (as indicated by <i>d</i>) is different from the referenced file's creator or account.	0 (This value will not be received by users with Account Manager Capability.)	User different.
		1	Account different.
4	The file cannot be locked down for storing because it is busy.	0	(Always output.)

The following example illustrates the use of the `:STORE` command.

EXAMPLE

To copy all files in the group *GP4X* (in the user's log-on account) to a tape file named *BACKUP*, the user enters:

```
:FILE BACKUP; DEV=TAPE  
:STORE *.GP4X; *BACKUP
```

Explicit or implicit redundancies are permitted in *filesetlist*, but once a file has been locked down, any subsequent reference to it in *filesetlist* results in the e/d message 4/0 even though execution of the :STORE command continues.

EXAMPLE

Suppose the file identified as FN.GN.AN is a member of the fileset referenced by @ in the following command.

```

      .
      .
      .
:STORE @,FN.GN.AN; *DUMPTAPE

```

The command is executed successfully, but the fileset# and e/d notations under *FILES NOT STORED* on the listing will show:

<i>filename</i>	<i>groupname</i>	<i>acctname</i>	<i>fileset#</i>	<i>e/d</i>
FN	GN	AN	2	4/0

This same file will also be noted under *FILES STORED*.

RETRIEVING DUMPED FILES

The user can read back into the system, onto disc, any file, fileset, or filesetlist that has been stored off-line (on tape) by :STORE. The files referenced are attached to the appropriate groups and accounts, with previous account and group names all re-instated. File retrieval is requested with the :RESTORE command. This command does not create any new accounts or groups. Any tape file to be restored will only be restored if the account name and group name exist on disc (in the system directory).

```
:RESTORE storedset [;[filesetlist] [;[dev] [;KEEP] ]]
```

<i>storedset</i>	The name of the file on which the <i>filesetlist</i> to be retrieved now resides. This file must be referenced in either the <i>filereference</i> or <i>*formaldesignator</i> formats. A message is output to the console operator telling him which tape to mount, and the device it should be mounted on. (Required parameter.)
------------------	---

<i>filesetlist</i>	The file, fileset, or file set list to be restored from tape. Each file is restored as a permanent file in the system file domain. This parameter is written in the same format, and subject to the same constraints as the <i>filesetlist</i> parameter of the :STORE command. If the <i>filesetlist</i> parameter is omitted, the default value is @.@.@. (Optional parameter.)
<i>dev</i>	The device class name or logical unit number of the device on which all files are to be restored. If omitted, an attempt is made to replace the files on the same device type (and subtype) on which they were originally stored in the system; if this cannot be done, an attempt is made to restore the file to the same device type (ignoring sub-type); if this fails, an attempt is made to restore it to the device class DISC; if this fails, the file is not restored. (Optional parameter.)
<i>KEEP</i>	A specification that if a file referenced in <i>filesetlist</i> currently exists on disc, the file on disc is kept and the corresponding tape file is not copied into the system. If KEEP is omitted, and an identically-named file exists in the system, that file is replaced. If KEEP is omitted, AND a file on tape is eligible for restoring AND a file of the same name exists on disc AND this disc file is busy, the disc file is kept and the tape file is not restored. (Optional parameter.)

The use of the :RESTORE command is subject to the same restrictions as the use of the :STORE command—that is, a user with System Manager or System Supervisor Capability can restore any file from a :STORE tape (assuming the accounts and groups exist in the system directory).

Any other user can restore any tape file in his log-on account if he has account-manager capability or is the creator of that file. The :RESTORE command performs the same checking performed by the :STORE command, to ensure a file's eligibility for retrieval. A listing is produced showing which files were restored and which were not, in this format:

FILES RESTORED = xxx

<i>FILE</i>	<i>.GROUP</i>	<i>.ACCOUNT</i>	<i>LDN</i>	<i>ADDRESS</i>
<i>filename1</i>	<i>.groupname1</i>	<i>.acctname1</i>	<i>ldn1</i>	<i>addr1</i>
<i>filename2</i>	<i>.groupname2</i>	<i>.acctname2</i>	<i>ldn2</i>	<i>addr2</i>
<i>⋮</i>				
<i>filenamen</i>	<i>.groupnamen</i>	<i>.acctnamen</i>	<i>ldnn</i>	<i>addrn</i>

FILES NOT RESTORED = yyy

<i>FILE</i>	<i>.GROUP</i>	<i>.ACCOUNT</i>	<i>FILESET</i>	<i>CONDITION</i>
<i>filename1</i>	<i>.groupname1</i>	<i>.acctname1</i>	<i>fileset#</i>	<i>e/d</i>
<i>filename2</i>	<i>.groupname2</i>	<i>.acctname2</i>	<i>fileset#</i>	<i>e/d</i>
<i>⋮</i>				
<i>filenamen</i>	<i>.groupnamen</i>	<i>.acctnamen</i>	<i>fileset#</i>	<i>e/d</i>

In this format, *xxx* denotes the total number of files restored; *yyy* denotes the number of files requested that were not restored. The notations *filename*, *groupname*, and *acctname* under the *FILES RESTORED* heading name the individual files restored, and their groups and accounts, respectively. The notation *ldn* indicates the logical device number (in decimal) of the device on which the file now resides, and *addr* is the absolute address (in octal) of the file label. The notations *filename*, *groupname*, and *acctname* under the *FILES NOT RESTORED* heading, indicate the individual files not restored, and their groups and accounts. The notation *filesset#* shows the number of the filesset to which the particular file belongs (relative to its position in the *filessetlist* parameter.) The notation *e/d* is an error code denoting the reason that the file was not restored, as follows. (These errors do not abort the file-restoring operation.)

Value of e	Meaning	Value of d	Meaning
0	The file already exists and KEEP was specified.	0	(Always output.)
1	The referenced file does not exist on tape.	0	(Always output.)
2	The user is not the creator of the file.	0	(Always output.)
3	A non-existent name was specified.	0	This was the <i>group</i> name
		1	This was the <i>account</i> name.
4	The file already exists.	0	The file was being created at a time during the execution of :RESTORE that prohibited its replacement with the tape version. This prohibition is effective whether or not KEEP was specified.
		Any other value.	This value is the error code for FOPEN obtainable through the FCHECK intrinsic. KEEP was not specified and the old file cannot be purged.
5	The creator of the tape file is not the creator of the disc file of the same name. KEEP was not specified. (This message can be received by System	0	(Always output.)

Value of e	Meaning	Value of d	Meaning
5 (Cont.)	Manager or System Supervisor users.)		
6	The file cannot be restored because of lack of space.	0	All devices of the specified class were searched for adequate space.
		A value greater than zero.	The logical device number of the only device searched.
7	A disc output error occurred during transfer of the file to disc.	A value greater than zero.	The logical device number of the device on which write errors occurred.
8	A user without System Manager or System Supervisor capabilities is specifying a file whose account name is different from that of the log-on account.	0	(Always output.)

EXAMPLE

To retrieve from the file named BACKUP all files formerly belonging to the user's log-on group, the user enters:

```

:FILE BACKUP; DEV=TAPE
:RESTORE *BACKUP; @; MDISC; KEEP

```

If a tape file satisfying the @ specification already exists in the system, it is not restored.

:STORE/ :RESTORE VS. :SYSDUMP/RELOAD

The **:STORE** and **:RESTORE** commands apply only to non-system files. The **:SYSDUMP** command is used to back-up all system-owned files. (However, in future MPE/3000 releases, **:SYSDUMP** also will optionally and implicitly issue the command "**:STORE @.@.**") Thus, the **:SYSDUMP** command alone will provide total system back-up for disc.

The tapes produced by **:SYSDUMP** and **:STORE** are compatible. If **:SYSDUMP** is invoked, **:SYSDUMP** writes all system-related data onto the first two files of the output tape, the header label onto the third file, and the tape directory onto the fourth file. Then **:SYSDUMP** (optionally) will invoke **:STORE**, which will write files 5 through *n* onto the tape. If **:STORE** is invoked explicitly by a user, **:STORE** first writes four null files onto tape to reflect the absence of the first four files normally produced by **:SYSDUMP**.

The **RELOAD** option of the MPE/3000 Initiator is the inverse of **:SYSDUMP**, in that it restores logical **:SYSDUMP** tapes. **RELOAD** cannot use **:RESTORE** implicitly, however; this is one reason for making **:SYSDUMP** and **:STORE** tapes compatible. (**:SYSDUMP**, **:STORE**, and **:RESTORE** use the MPE/3000 file system in accessing the tapes, but **RELOAD** cannot.) Another reason for compatibility of tapes is that **:RESTORE** can be used to reinstate specific files from **:SYSDUMP** tapes.

In future MPE/3000 releases, **:SYSDUMP** will provide three options regarding non-system files:

1. Dump no user files.
2. Dump all files.
3. Dump all files whose "last modification date" is greater than or equal to a date parameter provided during **:SYSDUMP** configuration.

In all three cases, the entire MPE/3000 system will be dumped unconditionally onto the first two files of the output tape.

The special entry point in the **:STORE** program used by **:SYSDUMP** has a date parameter, which is essentially $-\infty$ when **:STORE** is invoked explicitly by a user.

Because the user cannot directly specify a date parameter value when explicitly using **:STORE**, and because partial system-wide back-ups must provide for capturing the MPE/3000 accounting log (to be made available in future releases), the **:STORE** command is not suitable for use as the standard mechanism for obtaining partial back-ups of disc files. Instead, the following procedure is recommended (when the three **:SYSDUMP** options described above are made available):

On a given day *n*, obtain a complete back-up by using **:SYSDUMP** with the "dump all files" option. Afterward, obtain partial back-ups periodically by using **:SYSDUMP** with the "dump all files changed since day *n*" option, which will dump the current MPE/3000 system plus all files created or changed *on* or *after* day *n*. In order to minimize the number of logical tape reels involved in rebuilding a complete system from back-up reels, it is recommended that *all* partial back-ups be made relative to the complete back-up taken on day *n* and not relative

to the last partial back-up. Either way would work and each has its advantages, but the suggested method accumulates changes on one logical reel whereas the other accumulates logical reels.

The RELOAD option of the MPE/3000 Initiator requires that system back-up tapes be supplied in the reverse order of their creation (in other words, most recent partial back-up tape first, and complete back-up tape last). This method eliminates the possibility of having to overwrite or expand an already-restored file with a more recent version during RELOAD.

