MPE

Multiprogramming Executive for HP 3000

HEWLETT hp PACKARD

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

MULTIPROGRAMMING EXECUTIVE

A Description of the Operating System for the HP 3000

5952-4469(22)

15 MARCH 1972

Printed in U.S.A.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Section Page

SECTION III – THE MPE USER (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION TO MPE/3000

The Multiprogramming Executive Operating System (MPE/3000) is a general-purpose, disc-resident software system that supervises the processing of all user programs submitted to an HP System/3000 Computer. MPE/3000 relieves the user from many program control, input/output, and other housekeeping responsibilities by monitoring and controlling the input, loading, compilation, run preparation, 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, time-sharing, or real-time), 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, time-sharing, and real-time programs concurrently.

Batch Processing

Batch processing lets programmers submit one or more compiler-language programs as a single unit, called a *job*, to the computer. Jobs contain all instructions, programs, and data required for their execution. Once a job is running, no further information is needed from the programmer. Jobs are loaded through on-site (local) devices such as card readers or tape units. 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 programs within it are compiled sequentially and executed on a multiprogramming basis. MPE/3000 outputs the job locally on a device such as a line printer, tape unit, or card punch. If one job temporarily halts, (for example when the program requires additional data) another immediately enters execution. When many jobs are active in the system, users can maintain uninterrupted processing and high throughput.

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. They simultaneously share 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.

MPE/3000 allows time-sharing users to transmit batch-processing jobs from within their on-line sessions. Thus, during a session, a user can enter commands to compile and execute (in batch mode) a program from a previously-prepared source file. The user can then begin an interactive time-sharing program, interrupting it periodically to request MPE/3000 to check the status of the batch job.

Real-Time Processing

Real-time processing allows users to perform time-critical work that requires very rapid information processing. A real-time program, called a *task*, generally responds to or controls external events. Such programs are used primarily in process-control applications, where data gathered during a physical process must be input and operated upon so rapidly that the results can be used to influence the process as it develops. Real-time processing is also used in many commercial applications, such as instantaneous information retrieval, where all inquiries must receive an immediate response. Other applications are instrumentation monitoring and data acquisition, production control, and automatic testing.

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 the flowcharting and editing of files.

Each language translator and subsystem is accessed by a unique MPE/3000 command. The programmer need learn only one set of conventions to use any of these subsystems, 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.

EASE OF USE

MPE/3000 is easy to initialize, operate, monitor, and shut down. Its operation is overseen by the MPE/3000 System Manager, who assigns programming capabilities of various levels to each user. *Standard capabilities* allow the typical "general applications" programmer to interact with the computer only 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 by the System Manager. These allow him to access more sophisticated system resources for such tasks as running real-time programs, obtaining core-residency of code and data, and executing privileged instructions.

INPUT/OUTPUT CONVENIENCES

MPE/3000 handles all programs and data entering and leaving the system as labeled files. The programmer then gains access to the devices used for file input and output by file names rather than by device types or logical unit numbers — the devices themselves are treated as files.

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 acces, 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 disc files by many programmers.

MPE/3000 provides for efficient use of peripheral devices by offering the option of transferring (spooling) batch input or output temporarily onto disc. Under spooling, pending jobs are transmitted to disc until they can be run, maintaining a backlog of jobs that keeps the central processor continually in use. Similarly, output is spooled to disc so that it can be transmitted to the user the moment the required device is free. In this way, output devices such as line printers can be run at peak efficiency. When spooling is in effect, at a given instant, a file name may refer to an actual card reader or line printer or to a temporary file on disc.

PROCESSING EFFICIENCY

MPE/3000 offers the user maximum efficiency because it takes best advantage of the System/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, time-shared, and real-time modes possible.

Re-Entrant Code and Private Data

Within MPE/3000, many user and system functions can be active simultaneously without mutual interference. 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, and 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 generate many micro-instructions built into the central processor hardware. Microprogramming eliminates the repetitive coding and main memory requirements otherwise needed for recurring operations, thereby placing the operating burden on the highly-efficient microprocessor 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 a swapping area on disc.

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, sessions, and tasks 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 next highest priority. In general, real-time tasks have highest priority followed by time-sharing sessions, and then batch jobs.

SYSTEM-TO-SYSTEM COMPATIBILITY

All HP System/3000 Computers operate under one version of MPE/3000. This means that programs prepared on one System/3000 can be run on any other (of comparable hardware configuration) without modification. 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 AND FILE SECURITY

Each user operates in an environment protected from interference by other users. Program protection is provided by the hardware; file protection is provided by a software facility that allows the programmer to specify the degree of security desired.

DIAGNOSTIC MESSAGES

To aid in correctly establishing and executing batch and time-sharing programs, MPE/3000 provides the user with diagnostic messages that describe errors encountered in using the operating system.

SYSTEM MANAGEMENT

MPE/3000 provides many tools for system management that are available to those persons designated as the system manager and the system operator. The system manager is responsible for creation and control of users and accounts while the system operator maintains day-to-day control of the system including the following:

- Provide system back-up.
- Alter system configuration.
- Control program scheduling.
- Control spooling of batch jobs.
- Maintain the System Library.
- Modify the automatic collection of accounting and system-efficiency data.

SOFTWARE

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

- *System Configurator* — Used to configure, and initialize MPE/3000.
- *Command Interpreter* — Handles the commands that allow the user to interact with MPE/3000 through a terminal or batch input device.
- *File Management System* — Provides uniform access to disc files and standard input/output devices.
- *Memory Management System* — Dynamically allocates main memory among contending users.
- *Dispatcher* — Allocates central-processor time among programs in execution.
- *Input/Output System and Drivers* — Schedules, initializes, monitors, and completes input/output requests for standard devices (accessed through the file system).
- *System Library* — Used to store frequently-used routines sharable among many users.
- *Segmenter Subsystem* — Used to segment programs, load programs, and resolve 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 is:

- HP System/3000 Computer, including cpu and 48K bytes of core memory.
- Multiplexer Channel
- Moving-Head Disc; 4.9 megabyte cartridge, less than 35 msec average access time; 246K byte/second transfer rate.
- Console Terminal
- Magnetic Tape Controller and Drive.
- System Timer.

OPTIONAL HARDWARE

- Memory
  Up to 131,072 bytes total; 65,536 words.
- System Timers
- High-Speed Selector Channel (up to 4)

- Moving-Head Cartridge Discs (multiple controllers allowed)
  Up to 4 cartridge disc units per controller.

- Moving-Head Disc Packs (multiple controllers allowed).
  Up to 8 disc packs per controller.
  47 megabyte capacity.
  29 millisecond average access time.
  312K byte/second transfer rate.

- Fixed-Head Disc (1 disc per controller)
  1,2, or 4 megabyte swapping disc.
  8.7 millisecond average access time.
  500K byte/second transfer rate.

- Magnetic Tapes (up to 4 units per controller)
  7 track, 45 inches per second, 200, 556, or 800 cpi.
  9 track, 45 ips, 800 or 1600 characters per inch.

- Card Readers
  600 or 1200 cards per minute.

- Card Punches
  35 or 250 cards per minute.

- Line Printers
  200 or 600 lines per minute; upper/lower case option

- Terminals
  Hard copy: 10 or 30 characters per second.
  CRT screen: 10-120 characters per second.

- Paper Tape Readers and Punches

Data Communication Interfaces

a. The asynchronous serial interface provides a single line connection for a terminal device such as a teletypewriter or CRT display. The terminal may be hardwired or connected remotely through a Western Electric 103 or 202 type modem. I/O speed is software programmable from 50 bps to 200 Kbps.

b. A complete line of synchronous interfaces will connect a System/3000 to Western Electric type 201, 203, 301, 303 and 306 type modems at speeds from 2400 bps to 1.344 million bps. Mode of operation may be simplex, half duplex or full duplex. Hardware CRC is available as an option.

c. The sixteen line asynchronous terminal controller controls up to 16 terminals either hardwired or via Western Electric 103 or 202 type modems. Each line is individually programmable to operate at speeds from 75 bps to 2400 bps. MPE will provide automatic speed sensing of dial-up terminals and will provide software support for the following terminals:

Teletype ASR-33,35	10 cps
Teletype ASR-37	15 cps
UNIVAC DCT-500	10,15,30 cps
Execuport	10,30 cps
Hewlett-Packard	
2600 CRT	10 to 240 cps
DATAPOINT 3300 CRT	10 to 240 cps
Memorex 1240	10,30,60 cps
GE Terminet 300	10,15,30 cps
IBM 2741	14.8 cps

EXAMPLE SYSTEM CONFIGURATIONS

With the wide choice of 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 spooling of programs and data awaiting further processing.

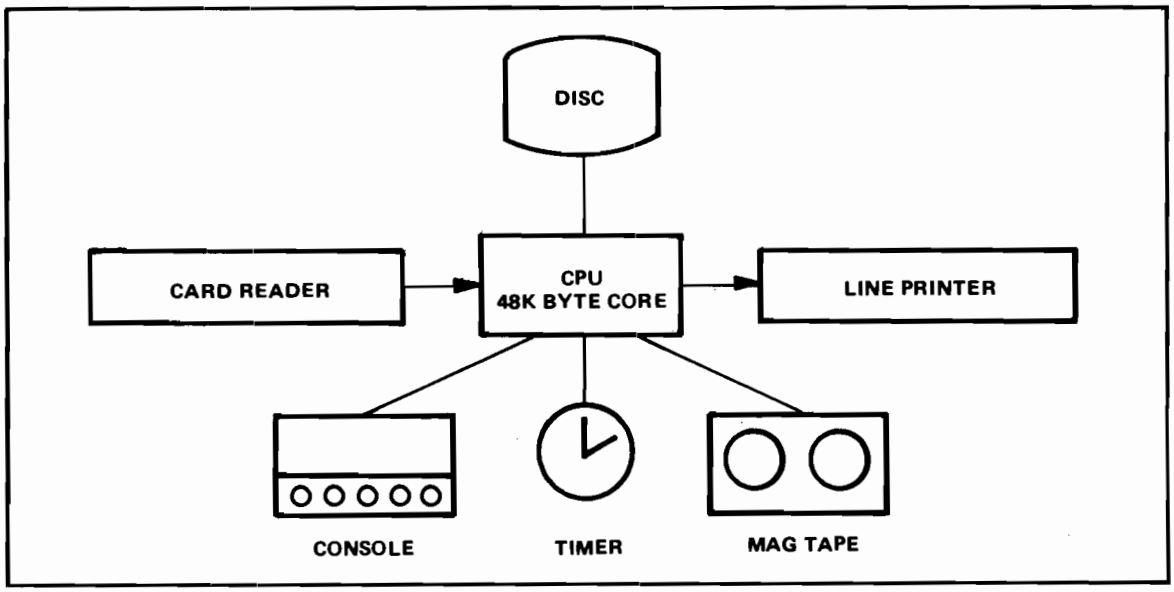


Figure 1-1. Small Batch System

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.

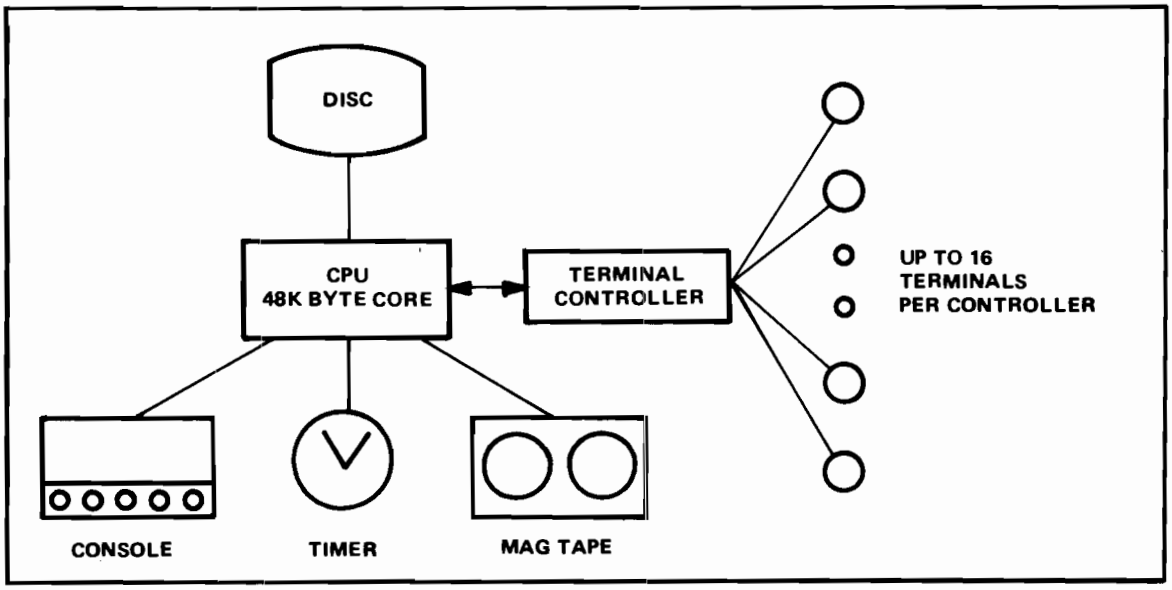


Figure 1-2. Small Time-Sharing System

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

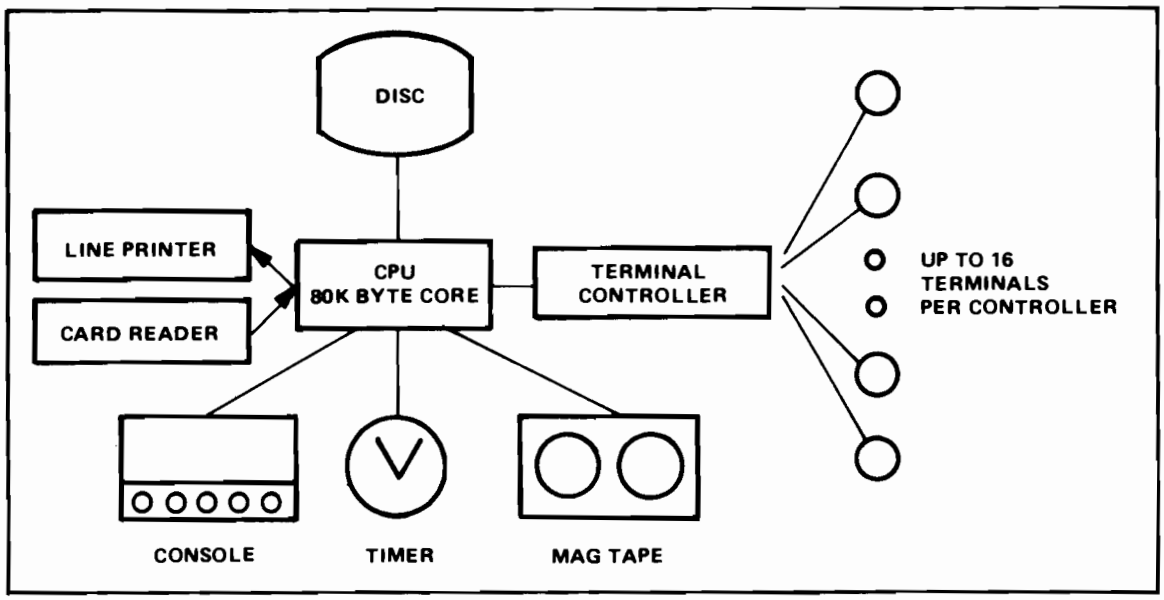


Figure 1-3. Combined Batch and Time-Sharing System

A combined batch and real-time system is presented in figure 1-4. The card reader and line printer are used for batch input/output; special devices such as voltmeters, pressure gauges, or other measuring devices, are used for real-time input. Control units, other electro-mechanical devices, terminals, or printers are used for real-time output. Some time-sharing terminals could also be connected to the system. Real-time tasks have the highest priority in the system, followed by time-sharing programs, and then batch jobs.

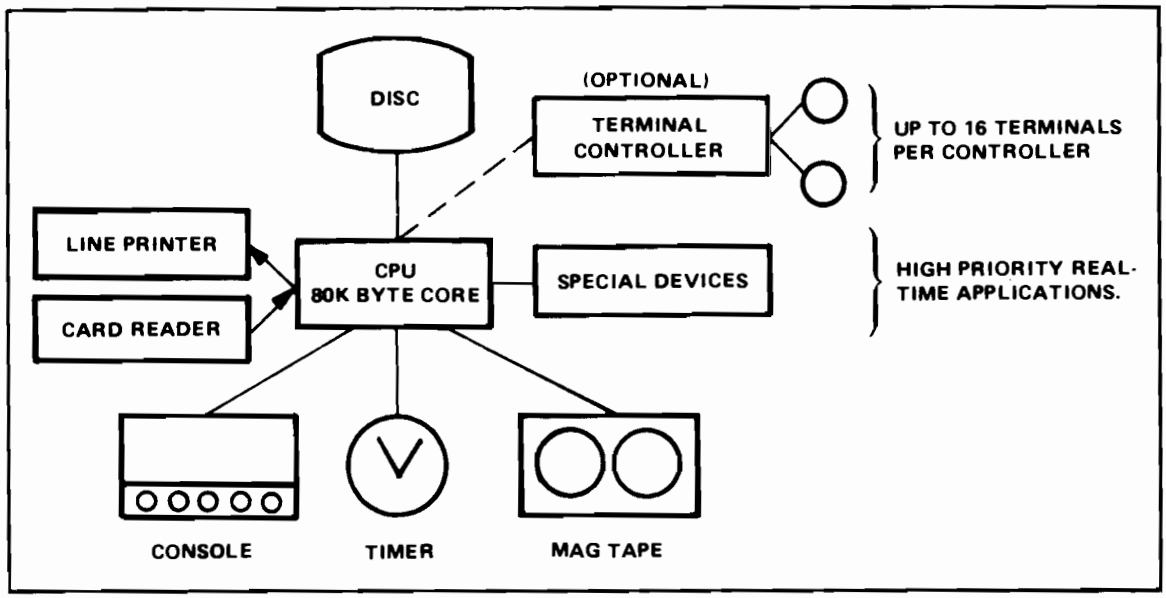


Figure 1-4. Combined Batch and Real-Time System

A large processing system is shown in figure 1-5. This system incorporates a large central memory, fixed-head and moving-head discs, many input/output devices for multiple-job batching, and several terminal controllers, each connected to up to 16 terminals for an extensive time-sharing network.

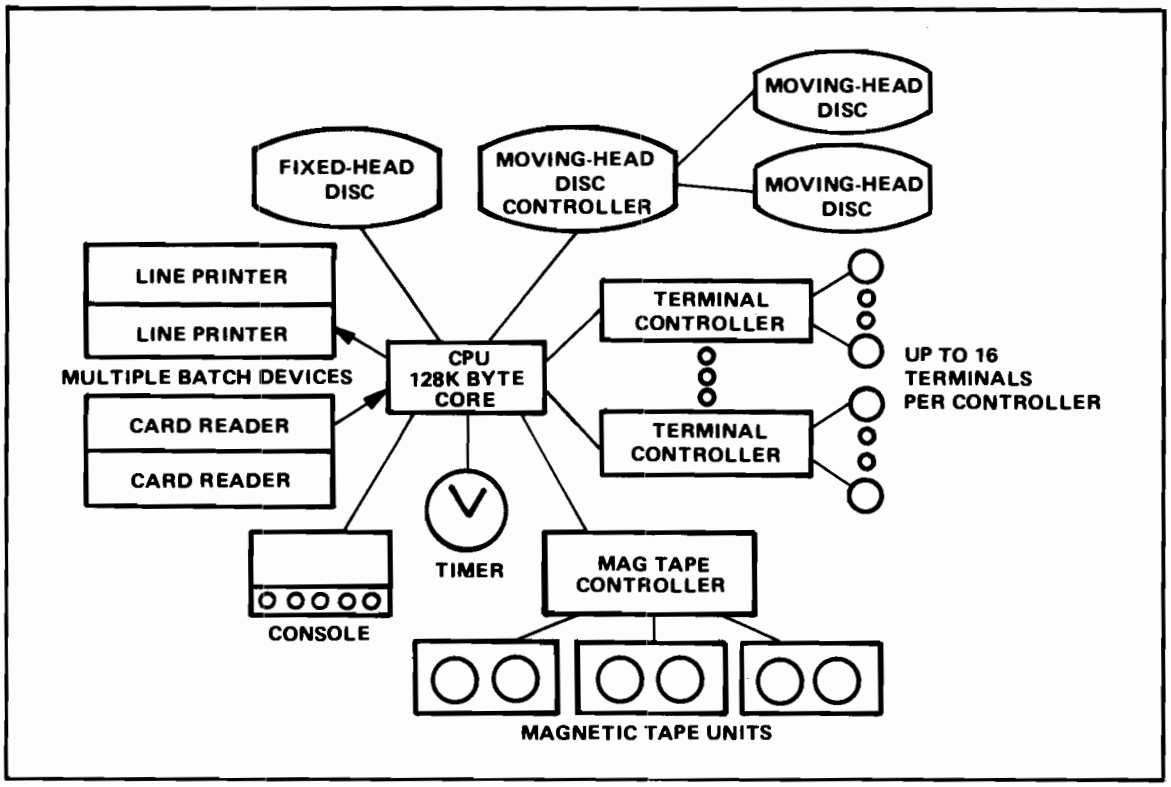


Figure 1-5. Large Processing System

SECTION II

AN APPROACH TO A 16-BIT MULTIPROGRAMMING COMPUTER

Most computers can be classified into three categories: 1) control computers – programmed data collectors or device controllers; 2) dedicated single usage computers – usually core based and in full control of the system; and 3) general purpose computers – usually disc based and under the control of an operating system. Traditionally, large scale computers have been associated with the third class and minicomputers have been used in the first two categories.

Controllers are generally small, fixed-program processors designed to handle a specific task or set of tasks. They have a limited amount of read-write memory, which is used for data storage, not for programs; they are usually connected either directly or indirectly via some data recording media to a larger computer. These controllers may be hard-wired or they may be minicomputers programmed (firmware or software) to do the job. In many of these cases, even the use of a small word-size minicomputer with a limited instruction set overpowers the situation.

The 8, 12, 16 and 18-bit minicomputers are more efficiently used in a dedicated single user situation. In this case the machine is dedicated to a single program at a time and therefore its limited instruction set is not a disadvantage. Most minicomputer architectures are designed to inexpensively implement the programming associated with the concept of a single user in full control of the entire machine. There is little or no need to protect one portion of the user's program from another, so features like environment-switching and memory protection are usually not included.

Because of the attractive pricing of these minicomputers they have been successfully applied to numerous problems. Applications such as BASIC language time-share systems and multiprogramming Real-Time Executives were programming feats of the first order. Features that were needed, but missing in the hardware were implemented in software. Some manufacturers have recently announced new models of their old minicomputers with some additional instructions and features to ease the programming burden for these types of systems. However, this is a limited attempt to make a single-user computer look like a general-purpose computer.

In the past this upward stretching of single user computers into the general-purpose category was due to the large gap in price between the two classes. Most general purpose machines like System 360, or B6500 or Sigma 5 had a 32 or 48-bit word length and a minimum cost of several hundred thousand dollars. Many of the lower-priced models had only a disc operating system with no provision for multiprogramming. In the System 360 one cannot run OS-MVT (OS/360 with Multiprogramming with Variable Tasks) on any model smaller than a Model 50 with 512K bytes of core.

The problem was how to fill the gap between the minicomputers and the large general-purpose computers. How could total system cost be minimized (not just the hardware) while maximizing the capability of the system? The two major factors to be considered in maximizing the capability were the types of services possible and the raw speed of execution ("number crunching"). The type of service seemed to be the key issue. The gap referred to

above was brought into focus by examining the minimum cost of an operating system that was capable of multiprogramming Real-Time processes, Terminal Oriented Processing and Batch processes. There was a distinct price barrier below which one was not able to obtain such an operating system at any performance level.

This multiprogramming goal places certain requirements on the system architecture. First, there must be a mechanism for keeping only portions of active programs in main memory. The two main methods are Paging and Segmentation. The former is typified by the System/360 Model 67 and has the characteristic of dividing a program into arbitrary sized pieces based on physical boundaries. The latter is found on Burroughs B5500 & B6500 Systems. It breaks the programs up on logical boundaries and doesn't restrict segments to a uniform size. Neither of these systems is perfect. Paging is easy to implement in hardware but leads to internal fragmentation which wastes memory. Segmentation, on the other hand, requires more extensive management to control the external fragmentation that can result from non-uniform sized segments being swapped in and out of main memory.[1]

Swapping programs and data required an efficient and automatic relocation technique to be part of the addressing structure of a multiprogramming computer. The operating system cannot take time to manually adjust all addresses in a program, nor can it guarantee to put something in the same physical location every time.

Re-entrancy is necessary so that parts of the operating system may be used by several processes without having to be concerned about the code being changed or temporary variables being destroyed by the other processes. This goes along with recursivity which allows more sophisticated techniques to be applied by the operating system.

Both of the above criteria make it easy to meet the requirement of code sharing. It would be extremely wasteful of main memory to keep multiple copies of programs in memory. However, the ability to share one copy places requirements on the system architecture that must be accounted for in the hardware if code sharing is to be efficient.

One of the key items in designing a multiprogramming operating system is that of user isolation and system protection. If the operating system and the users are not completely protected from the intentional or unintentional destructive actions of another user, the system will crash so often as to be unusable. This protection includes programs, data and files that exist in the system. Quite often the protection features are tied to the segmentation or paging scheme employed, which in turn influences that choice.

Main memory requirements must also be considered. The operating system should not occupy a large portion of main memory nor should user programs be overly large if many users are to be multiprogrammed. This implies that the architecture and instruction set should provide for good code compression; that is, the number of words necessary to implement an algorithm should be as small as possible. This criterion is reflected in a figure of merit for a machine which is the average number of bits per word. For code this is determined by counting the fractional word, full word and multiple word instructions along with the number of indirect address words amortized by the number of in-line references to each one. For data one must consider the number of wasted fractional words due to address boundary conventions as well as the number of multiple precision operands. When this figure of merit begins to approach the next standard word size machine, then the basic assumptions of a design should be re-evaluated.

Not only must system programs occupy as little memory as possible, but they should execute efficiently in order to minimize the operating system overhead and maximize the system throughput. Each instruction should execute a large amount of computation to decrease the instruction-fetch overhead. The instruction set should reflect the needs of the operating system, language compilers and the compiled output code. The latter is necessary to allow extensive but efficient use of high-level programming languages.

If a small word-size machine is being considered, the amount of addressable memory is likely to be limited. To take full advantage of it, a dynamic storage allocation scheme is useful. All temporary and local variables should be assigned physical memory only when needed at procedure or block entry and should be de-allocated upon exit. This characteristic of a procedure-oriented high level language should be reflected in the hardware design. Another aspect of storage allocation which should be avoided on small machines is the use of fixed or variable memory partitions. Partitioning places arbitrary and limited restrictions on the most valuable resource in a multiprogramming system. The goal of a uniform Real-Time, Terminal Oriented and Batch environment suggests the absence of partitioning and the use of priorities to control resource usage.

Multiprogramming and its usage of main memory dictate the need for concurrent I/O and CPU processing if the system is to operate efficiently. Many minicomputers that have Direct Memory Access actually stop the CPU during the cycle(s) I/O is accessing memory. This slows down CPU computation and may impair system throughput. Directly related to this question is the hardware modularity of a multiprogramming computer system. It should be easy to add new equipment or remove malfunctioning equipment without having to go through a major system reconfiguration. In addition the system should be easy to diagnose and service when it breaks down. The down time of an on-line system is crucial and must be minimized.

The Hewlett-Packard System/3000 is an attempt to meet all of the above multiprogramming requirements while minimizing the total system cost. The cost objective dictated a 16-bit hardware design with an architecture that supports the Multiprogramming Executive (MPE/3000) environment. It should also facilitate the use of a high level machine-dependent Systems Programming Language (SPL).

Designing a 16-bit computer presents several special problems that are not encountered in large word length machines. For example, the ability to directly address as much memory as possible. The system designer must also provide an extensive instruction set, without raising the average number of bits per word by using multiple word length instruction formats.

Traditionally, 16-bit computers have been von Neumann-like machines with little or no distinction between program code and data. In a multiprogramming environment there is much to be gained from separating the two. In the System/3000 a typical user's environment consists of one or more code segments and a data segment (figure 2-1). All code is non-modifiable while active in the system, allowing the use of overlay techniques without having to write the code back out on the swapping disc since an exact copy already exists there. The data area consists of global data and a push-down stack that is handled automatically by the hardware. The data is non-executable except through the Execute instruction.

Segmentation was chosen over paging as a more flexible and logical way to handle programs. This technique has been used previously only on large computer systems like the Burroughs

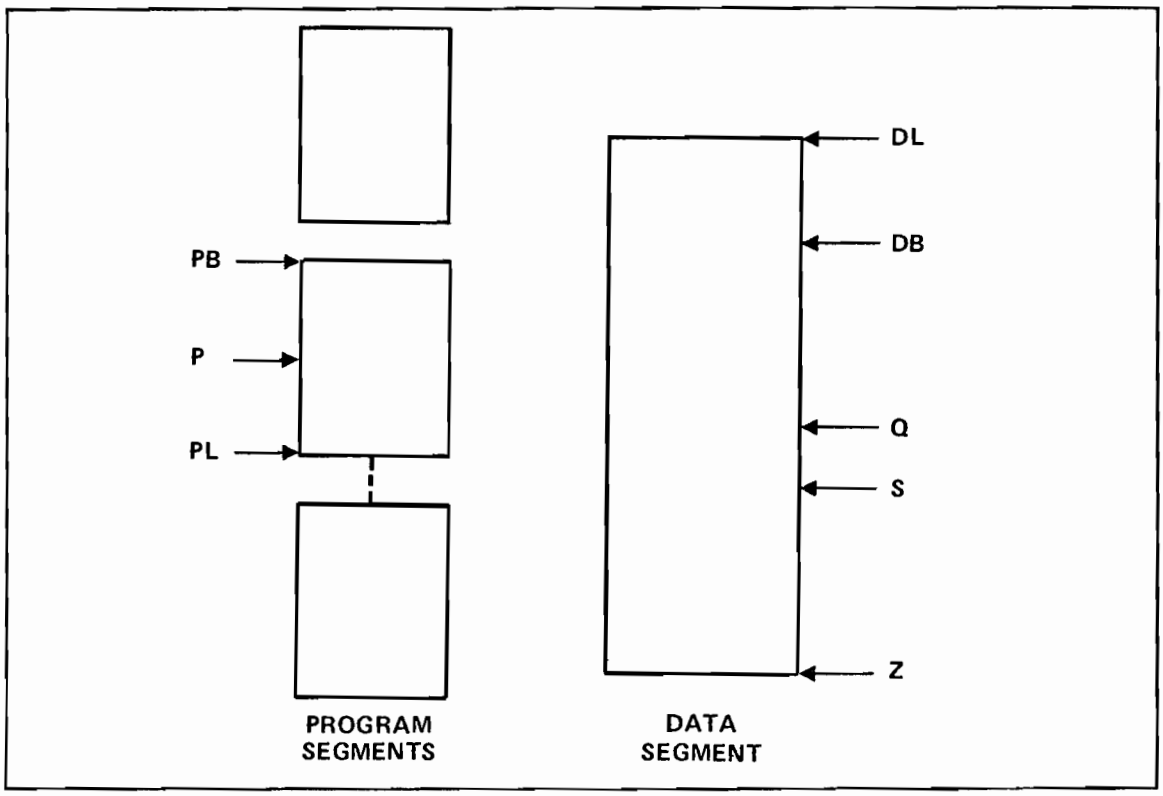


Figure 2-1. Typical User Space

B5500 and B6500.[2] The code is grouped into logical entities – segments – each consisting of one or more procedures. Due to limitations imposed by the use of a 16-bit descriptor, each segment may be up to 16K words long. There is a master directory, the Code Segment Table (CST), that contains one entry for each segment that is currently active on the system (figure 2-2). The CST is maintained by the operating system (MPE) and is used by the microprogram in the CPU for procedure entry and exit. The table does not occupy a fixed position in memory but is located at the address given in absolute location 0. Each two word CST entry contains the beginning address and the length (divided by 4) of the segment. There are also four bits that are used by the microprogram: 1) the Reference bit is set by the hardware each time the descriptor is accessed and is used to implement a software least-recently-used overlay algorithm; 2) the Trace bit causes a procedure call to the Trace routine if set; 3) the Mode bit specifies whether the segment will be run in Privileged or User mode; 4) the Absent-from-main-memory bit causes a procedure call to the make-present routine if set, and it implies that the second word of the CST descriptor is a disc address. The maximum number of entries in the CST is 255, limited by the number of bits available in other descriptors. When a segment is given control of the CPU, the Program Base (PB) and Program Limit (PL) register are set from the CST entry of that segment.

Each segment has a Segment Transfer Table (STT) which is used in conjunction with the CST to locate procedures within a segment (figure 2-3). There are two basic types of entries in an STT: 1) Internal labels give the starting address of procedures relative to the PB register. Procedures may be marked Uncallable so that they cannot be called from outside the current segment; 2) External labels give the Segment-number and Subsegment-number of all procedures that may be accessed from the current segment. The STT is built by the Segmenter when a segment is made active and includes all the necessary labels; the grouping of procedures into segments is decided by the user at load time.

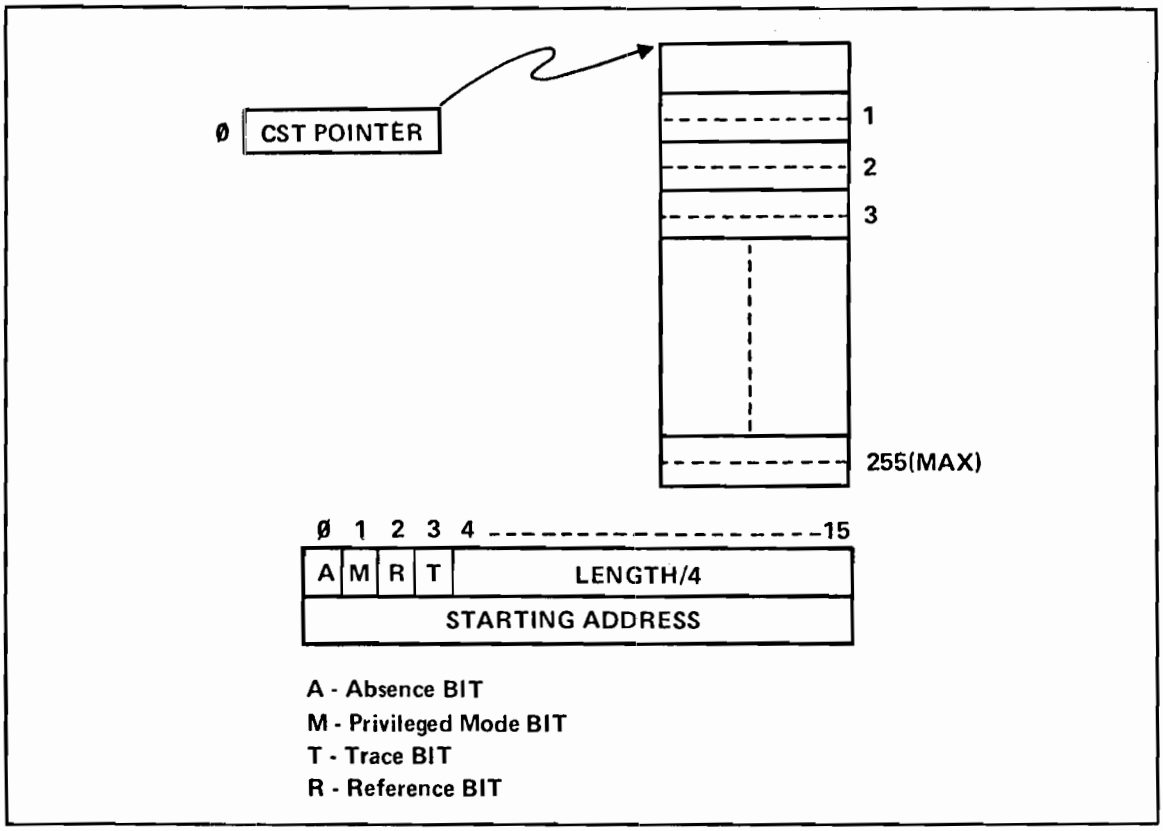


Figure 2-2. Code Segment Table

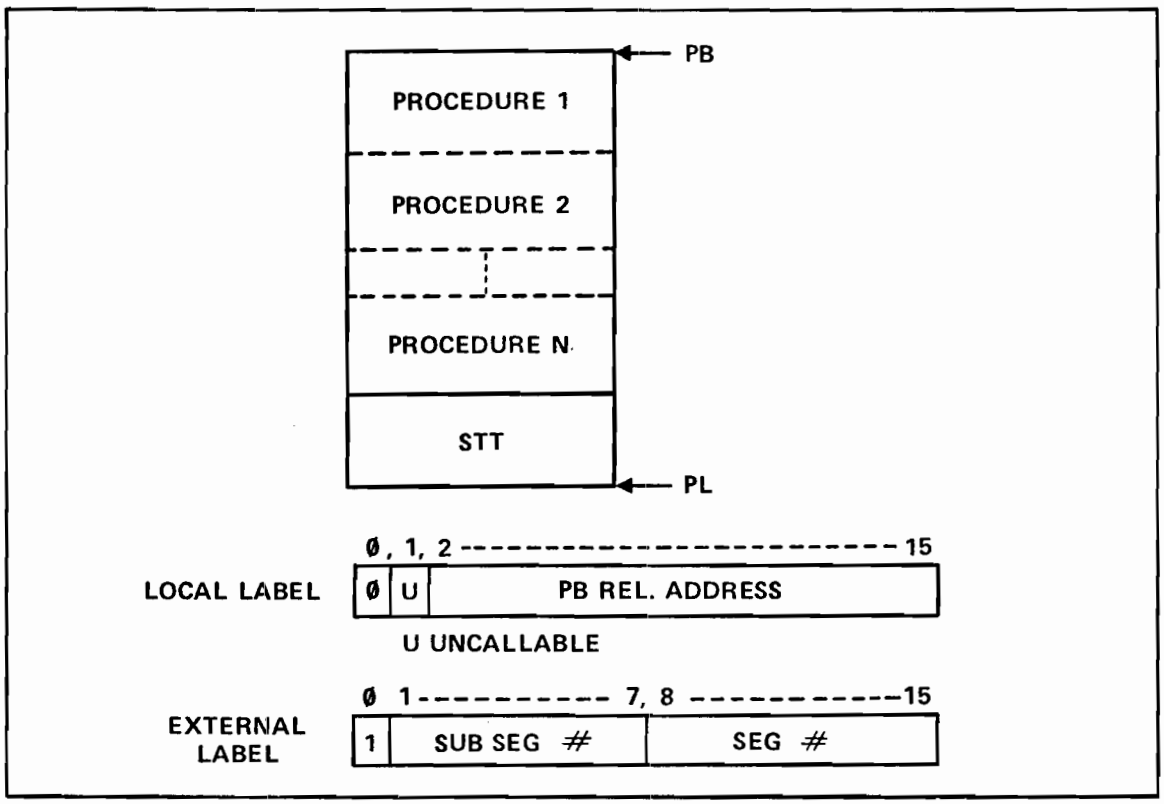


Figure 2-3. Typical Code Segment & STT Label Types

Data Segmentation is handled by the MPE/3000 operating system. A normal user has only one data segment, which is limited to 32K words. This maximum size is dictated by byte addressing restrictions. Up to four additional data segments may be requested by a process.

Relocation becomes the normal mode of operation because of the relative addressing capability of the System/3000. Figure 2-4 gives the memory reference instruction format. The address mode bits have been Huffman-coded to give the maximum displacement range on the most frequently used modes. In the code segment, normal addressing is Program Counter (P) relative with a range of up to ±255. Indirect addressing is the same except that the content of the indirect cell is assumed to be relative to its own location. This simplifies the Segmenter's job yet maintains total relocation even on indirect references. Procedure calls use descriptor addressing through the STT, which ultimately gives a PB-relative address. Certain instructions in the Move group can have their source address as PB-relative.

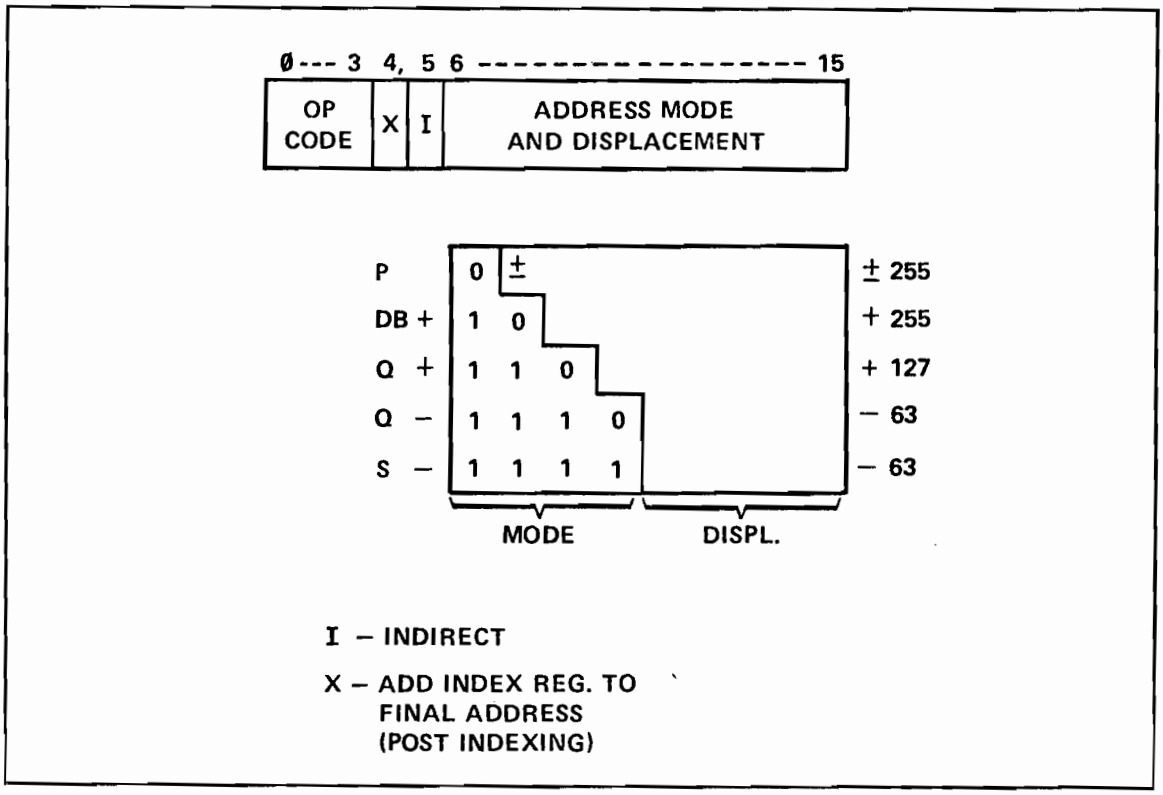


Figure 2-4. Memory Reference Instruction Format

In the data segment the addressing modes are designed to match the types of data encountered in a procedure oriented language. Figure 2-5 shows the organization and common usage of the data area. Global variables and pointers are stored relative to the Data Base (DB) register; the DB+ mode has a direct range of up to 255 words without indexing or a 64K word range with indexing. Parameters that are passed to procedures are pushed onto the stack before the procedure call which saves the machine status on the stack and sets the Q register to point at the stack marker. These parameters are then accessed by Q- addressing, while the local variables used by the procedure are accessed by Q+ addressing. The addressing in the negative direction with respect to the Stack Pointer (S) register is useful for accessing temporary results left on the stack during processing. The area between the Data Limit (DL) register and DB may be addressed only indirectly and is used for such purposes as storing symbol tables and the like.

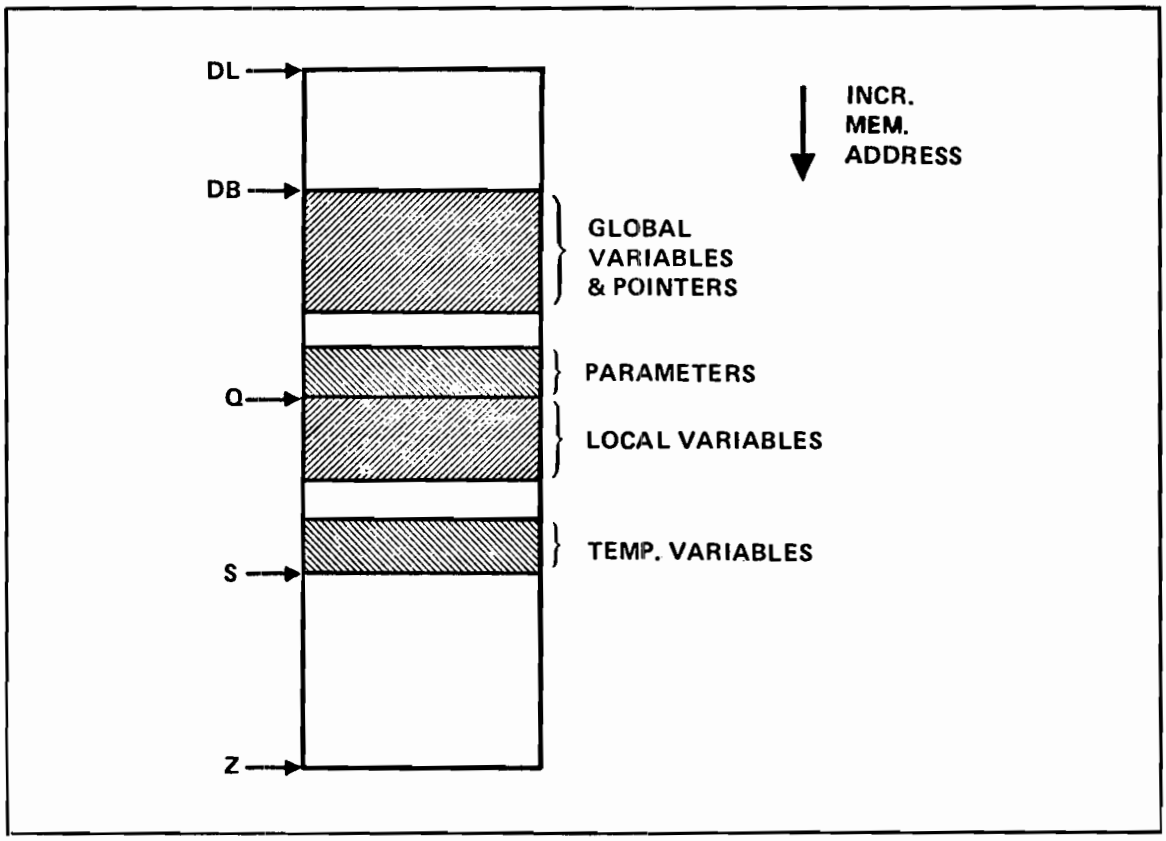


Figure 2-5. Data Area Addressing

There are instances where the operating system needs to have access to absolute locations in memory and for this purpose there are two privileged instructions that load and store into absolute addresses. The I/O system also uses absolute addresses for direct memory access and therefore must be handled by the operating system to prevent a user from obtaining access outside his space through the I/O system.

The separation of code and data, the use of a push down stack with Q+ and Q- addressing modes and the non-modification of code make re-entrant code the natural way to write System/3000 programs. Re-entrant code in conjunction with the use of the Code Segment Table as the master directory of all active segments allows code segments to be shared between users. Control is transferred through the STT and CST to the proper segment number of the shared code as determined by the loader when the segment was made active. In this manner only one copy of a compiler or a library or the operating system intrinsics need be available, saving valuable space in main memory. This concept of code sharing is simple, but is not easy to implement without a single central table like the CST to co-ordinate the automatic hardware transfer of control from one segment to another.

User isolation and protection takes several forms on the System/3000. Programs may execute in one of two modes: Privileged or User. In Privileged mode no bounds checking is done except for stack overflow ($S > Z$), and all instructions are available for use. All system interrupts including external (I/O) interrupts are handled on a separate Interrupt Control Stack so that the user running when the interrupt occurs is fully protected.

In User mode, access is limited to within the user's own code and data areas. This protects the system and other users from a given user but makes no attempt to protect him from

himself. In the user's code area he is limited to memory references within the range from PB to PL. There are no instructions that can modify memory which are P-relative (e.g., Store, increment memory). All transfer of control instructions must have a target address within the PB to PL range or go through the CST to change PB and PL to protect another segment. All procedure exits check the mode of the destination segment against the current mode to insure that a user cannot put himself into privileged mode. User mode addressing within the data area is limited to the range from DL to Z; in addition, stack overflow (S > Z) and underflow (S < DB) are also checked. Memory protection checks are usually overlapped with the operand fetch and therefore do not slow the execution down. Data words may not be executed except by placing an instruction on the top of the stack and using the Execute instruction.

In addition to the hardware memory protection, files are protected by the MPE/3000 File Management System. Access to files may be controlled at several levels which range from unrestricted access by anyone to controlled access available only to the creator of the file. File access modes are Read, Write-append only, Write, Save, and/or Locked (for use during critical file updating). This file protection mechanism supplements the hardware protection to provide complete user isolation and protection.

Code compression is achieved on the System/3000 through the use of both single-address instructions and zero-address (stack) instructions (figure 2-6). All instructions except the sixty-four stack operations are in a single word format. The stack ops may be packed two per word to further enhance the code density. This latter feature is not always used since two stack operations may not occur consecutively. In this case the presence of a NOP in the second position is detected by the hardware and no time is wasted on its execution. There are 170 unique and meaningful instructions in the System/3000. Many of these have multiple actions which give a high complexity-to-instruction ratio.

Not only is the hardware data space organized in a manner conducive to procedure-oriented languages, but so is the instruction set. The stack concept and operations performed on a stack nicely match the way a compiler evaluates expressions, as well as the way it transfers control from one procedure to another. There are many high level instructions that permit efficient coding and implementation of high level languages.

For writing compilers:

- Move block of words from source to destination.
- Move byte string while alphabetic, or while numeric, or while either.
- Scan byte string until one of two bytes found, return count.
- Scan byte string while identical to test byte, return count.
- Load Relative address (used in working with pointers).

For efficient execution of compiler-produced code:

- Modify a variable or index register, test it against a signed limit, branch if limit not exceeded (used with DO loops).
- A uniform arithmetic set for integer (16-bit 2's complement), double integer (32-bit 2's complement), logical (16-bit positive integer), and floating point (32-bits including 23-bit precision mantissa).
- Special instructions like triple normalizing shift to aid software multiple precision floating point.

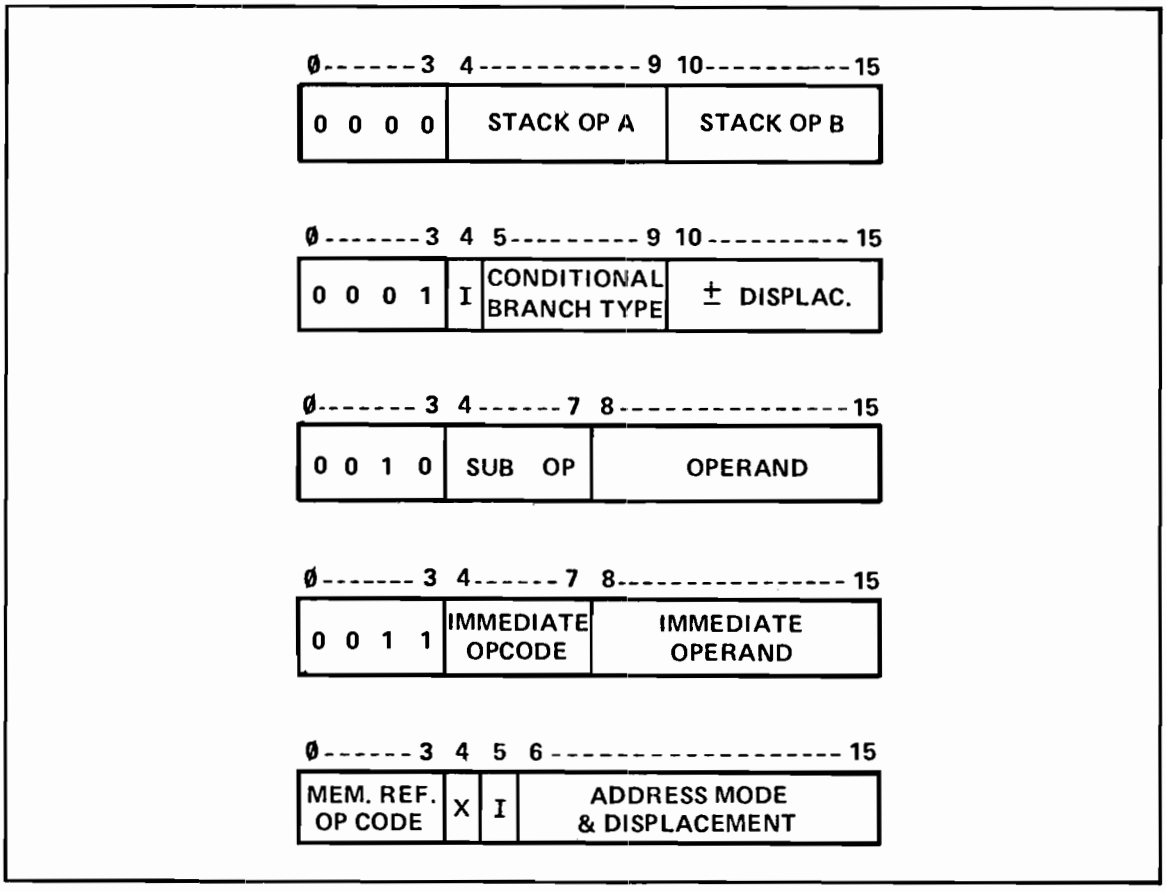


Figure 2-6. Major Instruction Formats

For helping the operating system:

- Linked List Search (for use in finding free space for the segmentation memory management).
- Load label (used to ease the loader's problems caused by passing procedures as parameters).
- Procedure Call and Exit (change the environment automatically and minimize interrupt bookkeeping).
- Privileged Load and Store into absolute address (used for maintaining fixed location tables and pointers).
- Start Input/Output (SIO) (causes a device controller to begin execution of its independent I/O program).

The stack concept, which on the System/3000 is fully utilized for the first time in a 16-bit machine, allows dynamic storage allocation on a procedure level. Local storage in a procedure is allocated only upon entry and is automatically freed upon exit. This allows reuse of that area of memory by other parts of the program. The stack also provides automatic temporary storage of intermediate results until they are needed later in a computation. This is transparent to the programmer, and the compiler doesn't have to be concerned with saving and restoring registers.

On a more global level the System/3000 memory management section of the MPE allocates memory dynamically on a priority basis among contending users. There is no partitioning

of memory; Real-Time, Time Share and Batch programs may be put anywhere in available memory. They are dispatched according to their priority in the master queue (see figure 2-7). There are both linear and circular sub-queues to fit the type service the process requires.

The Input/Output system of the System/3000 consists of three parts: 1) Direct I/O, 2) Programmed I/O, and 3) the interrupt system. Under Direct I/O one word is transferred between the specified device and the top of the stack. All of the I/O commands test the device to see if it is ready before issuing the actual command. The Direct I/O operations are read (RIO), write (WIO), test status (TIO), control device (CIO), set interrupt (SIN), set interrupt mask (SMSK) and read interrupt mask (RMSK).

The programmed I/O is an independent I/O program mechanism that is initiated by the CPU executing a Start Input Output (SIO) instruction against the specified device. The device controller in conjunction with the Multiplexer Channel then executes the SIO program independently of the CPU's further actions. Figure 2-8 shows the SIO program double word instructions. The controller continues to fetch and execute its program until it encounters an END order. Through data chaining, the SIO program can do scatter-reads and gather-writes. The program pointer for each device is the first word of the device's four word entry in the Device Reference Table (DRT). There may be up to 253 device numbers on the computer.

The other three words in a DRT entry are associated with that device's interrupt. One contains the PB register value of the interrupt receiver program, one contains the DB register value, and the third is used by the MPE Dispatcher. An external interrupt causes the CPU to transfer control from the present user to the interrupt program in a manner similar to what would happen on a procedure call except that all interrupt routines use a common stack, the Interrupt Control Stack (ICS).

At the end of the interrupt routine control does not return to the interrupted program but goes to the MPE Dispatcher. This portion of the operating system then initiates the highest priority program ready for execution.

The interrupt structure is a multilevel priority network. A device's interrupt priority is independent of its I/O program memory-access priority. The advantage of this separation is shown by the example of a disc memory and a real-time alarm. The disc needs high DMA priority to prevent data overruns but only needs to interrupt when it is through with the data transfer. The alarm on the other hand transfers very little data but needs a very high priority interrupt.

To fully understand the power and flexibility of the System/3000 one must also examine the physical hardware organization. Figure 2-9 shows the modular structure. The central feature is the Module Control Unit (MCU) Bus which can service up to seven independent and asynchronous modules. Communication on the MCU Bus is done in sync with the Bus clock. Therefore there is no need for "hand shaking", and the Bus operates at the highest possible data rate.

Because of the MCU Bus structure, the modules attached to it are technology-independent. This includes the memories, permitting either magnetic cores or semiconductors. The minimum useful memory cycle time is limited by the MCU Bus data rate. There may be up to four memory modules, and these independent modules may be optionally 2 or 4 way interleaved to make more effective use of slower but proven technologies. The maximum amount of memory that may be addressed by the System/3000 instruction set is 128K bytes, limited by the 16-bit word length.

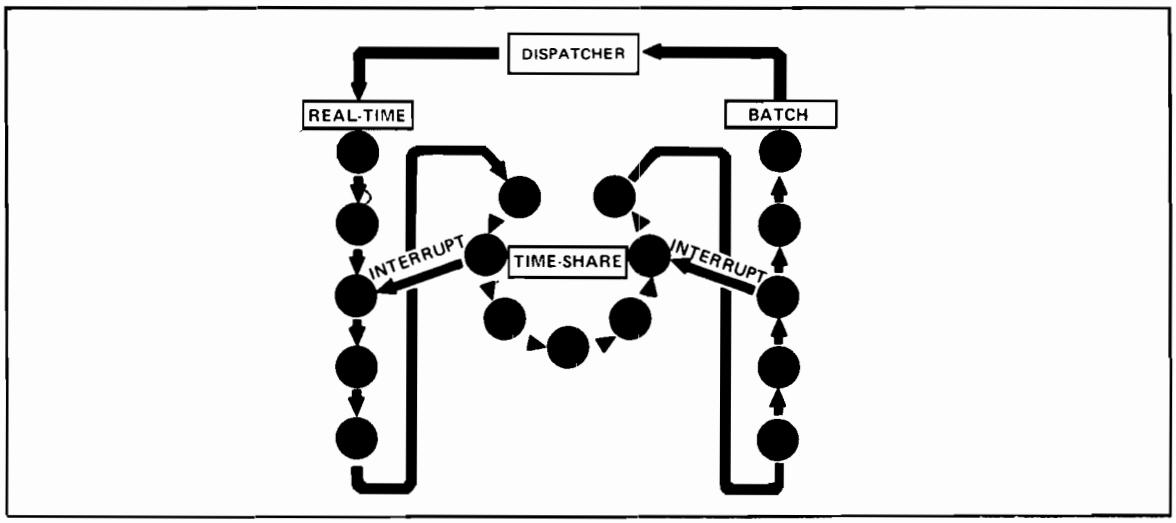


Figure 2-7. MPE/3000 Dispatcher Queue

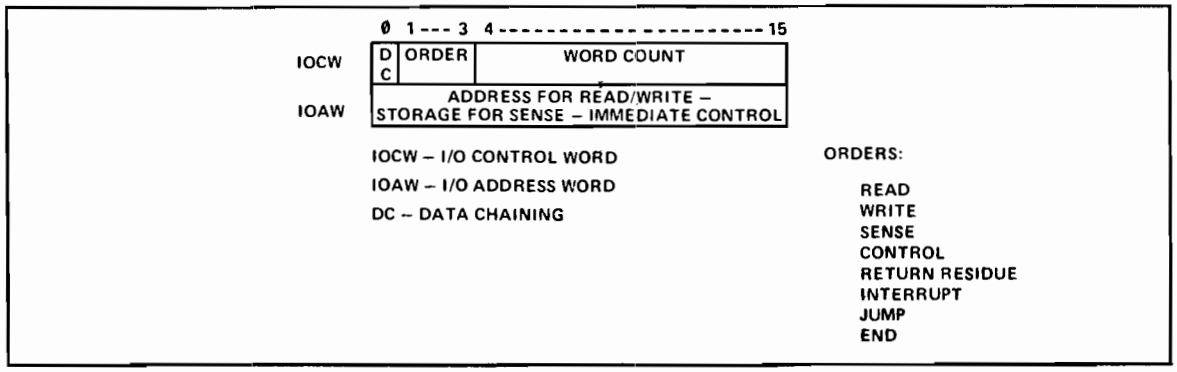


Figure 2-8. SIO Program Double Word Format

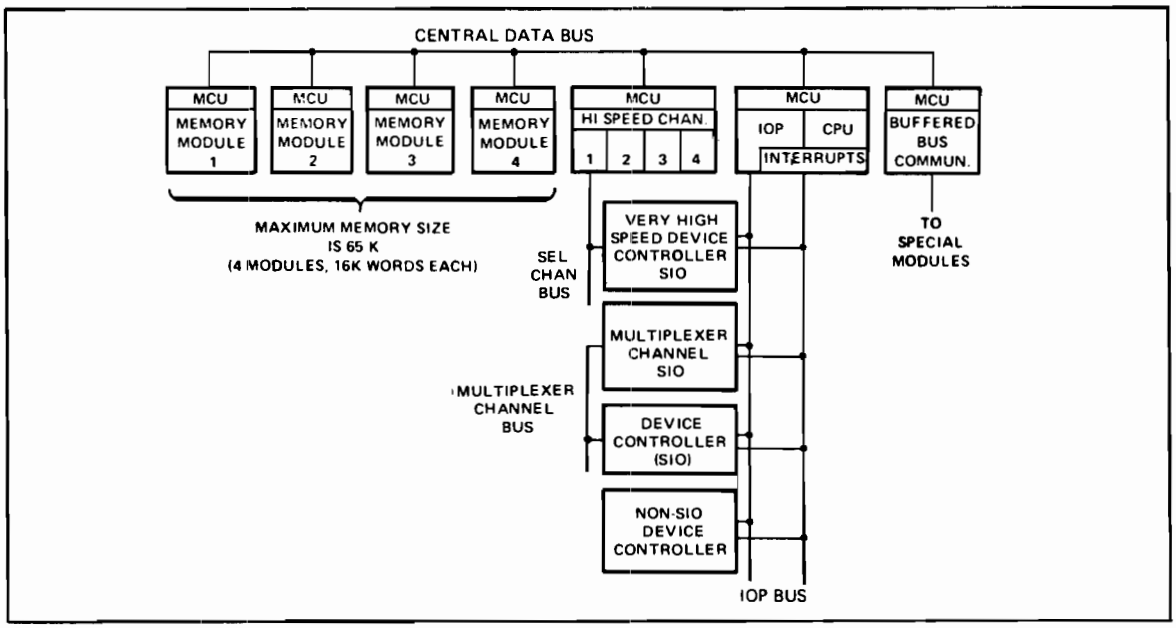


Figure 2-9. HP System/3000 Modular Organization

The CPU is a microprogrammed processor utilizing Large Scale Integration bipolar ROM's in a "vertical" format. The microword is 32-bits wide. The basic cycle time is equal and synchronous to the MCU Bus clock of 175 nsec. The microprocessor structure is pipelined to obtain this fast clock rate using TTL logic. The CPU contains the ten environment registers previously mentioned (PB, P, PL, DL, DB, Q, S, Z, Status, X) as well as 4 hardware top of stack registers that are transparent to the programmer but speed up the execution times. There are hardware stack-manipulation micro-orders to ease the microprogramming job. Software instructions are prefetched during the execution of the previous instruction. Some typical execution times using a 980 nsec core memory are:

Load	2.10	microseconds
Store	2.10	microseconds
Add	1.22	microseconds
Flt Add	14	microseconds
Flt Mpy	18	microseconds

The I/O Processor (IOP) shares the same module number as the CPU for access to the MCU Bus. This facilitates the execution of the direct I/O instructions. When conflicts over bus use arise, the IOP takes priority in order to guarantee service to fast synchronous I/O devices. The IOP is basically a doorman for the Multiplexer Channel. This channel provides a 950 kilobyte per second aggregate data rate when used with a 980 nsec memory. Direct I/O commands may be executed concurrently with SIO programs against a device. As discussed above, the SIO data priority is independent of the interrupt priority and both are independent of the device number, physical position in the card cage and the interrupt mask group.

When the aggregate data rate begins to approach the limit of the Multiplexer Channel, a High Speed Channel module is available. This is a selector channel and may have only one of the devices (maximum of eight) attached to it active at any time. There may be up to four of these channels in the module, each with a potential maximum data rate of 2.8 Megabytes per second. The High Speed Channels are used with high data rate devices like fixed head swapping discs or high speed analog-to-digital converters. The High Speed Channels are transparent; that is, a device uses the same direct I/O and the same SIO program whether it is attached to the Multiplexer or Selector Channel.

In addition to the logical and physical descriptions of the HP System/3000 one must talk about the serviceability of such a multiprogramming system. On the software side there are symbolic debugging programs available (TRACE/3000) as well as the system analysis routines that use the trace bit in the CST descriptors. In addition there is a microprogram in the CPU that will dump the entire contents of memory and all the CPU registers onto a device selected by the control panel switch register. This data will provide an ongoing data base for analysis of system crashes in an effort to improve MPE/3000.

For on-line preventive maintenance or verification of correct operation of peripheral devices there is a software subsystem, the System Diagnostic Monitor (SDM/3000), that allows the operator or service engineer to exercise a particular device while the rest of the computer is still under the control of the normal operating system. To handle tougher hardware problems there is a set of stand-alone software diagnostics that will check out peripherals, memory, and the CPU instruction set by assuming full control of the hardware.

Since the CPU and several of the more sophisticated device controllers (disc and magnetic tape) are microprocessors, there is a set of microdiagnostics available to help the service engineer find the problem. These diagnostics start with a small kernal of assumed good hardware to check out ever-increasing portions of the hardware. There is a hardware maintenance panel that is used in conjunction with the microdiagnostics which gives the service engineer the ability to watch the internal status of the microprocessor, as well as the ability to set breakpoints on the ROM Address Register.

The microdiagnostic hardware, the hardware maintenance panel and the auxiliary control panel may be connected remotely to a computer over a modem-common carrier line to allow factory assistance on difficult problems (see figure 2-10) and therefore reduce the down time.

This, then, is the approach to a 16-bit multiprogramming computer: define the objectives, determine all of the requirements, design the architecture from the ground up to meet the hardware, software and service requirements, build the system so that it runs reliably and leave as much potential for future expansion of the design as is consistent with the cost goals.

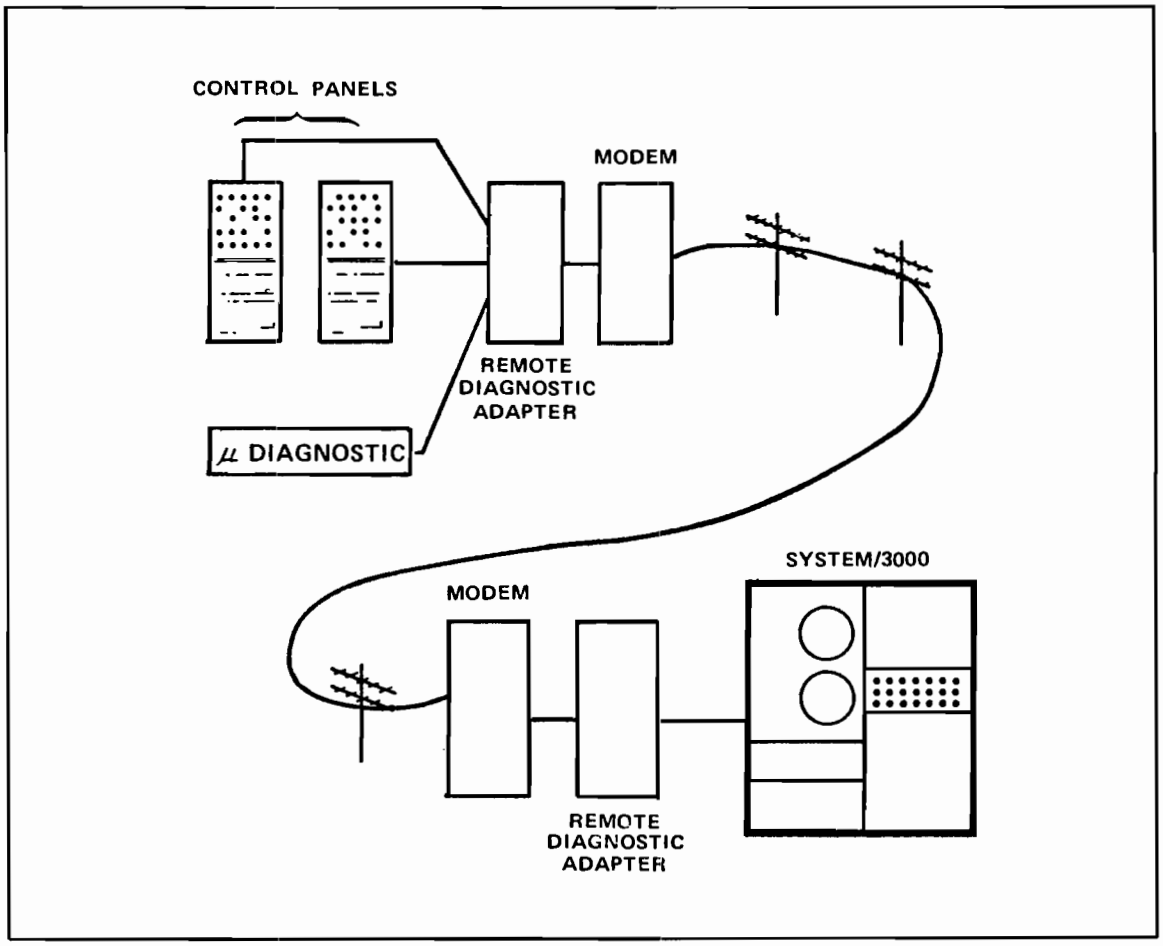


Figure 2-10. Remote Diagnosis on HP System/3000

BIBLIOGRAPHY

1. Randell, B. and Kuehner, C.S., "Dynamic Storage Allocation Systems", CACM, Volume 11, Number 5, May 1968. pp 297-306.

2. Cleary, J.G., "Process Handling on Burroughs B6500", Proceedings of the 4th Australian Computer Conference, Adelaide, South Australia, 1969, Volume 1. pp 231-239.

SECTION III

THE MPE USER

HOW MPE VIEWS ITS USERS

The users of MPE are organized into unique accounts. The System Manager controls these accounts by creating them, deleting them, and changing their characteristics. Each account has associated with it a list of users and a list of groups. Users are individuals who are given the capability of accessing the MPE System under a particular account name. Groups are used to partition the set of files belonging to an account. It is the responsibility of the account manager to create, delete or change the characteristics of the users and groups in his account.

THE ACCOUNT

Accounts are defined by the system manager. Each account has a unique name and an optional password which must be used to access MPE on behalf of this account. In addition, MPE will store a list of user names and group names recognized by this account, a maximum job priority (the highest priority at which any job in this account may be scheduled), limits on this account's usage of disc file space, CPU time, and connect time. MPE also maintains running counts of each resource that the account has actually used.

THE USER

Once an account has been defined by the system manager, the person designated as account manager identifies valid users of that account by name and optionally, a password. In addition, each user has a specified home group which is assigned when the user does not specify a group while logging into MPE. Each user also has a maximum job priority which may not be greater than that of the account he belongs to.

THE GROUP

Groups are also specified by the account manager. Each group within an account has a unique name, and optionally, a password. The account manager may also define limits on disc file space, CPU time and connect time used by a particular group. MPE will maintain running counts of resource usage for each group within an account. The sum of these running counts of the groups will always equal that of the account in total, and cannot exceed the account limit.

Groups are used to partition the file domain of an account. Each account "owns" a unique set of files separate and distinct from every other account. This ownership of files is indirect in the sense that only groups may own files directly. Therefore, every file belongs to a group, every group belongs to an account, and every account belongs to the system. This tree structure can be shown as in figure 3-1. From this diagram it can be seen that since each group defines a private file domain, file names must be unique only within a particular group.

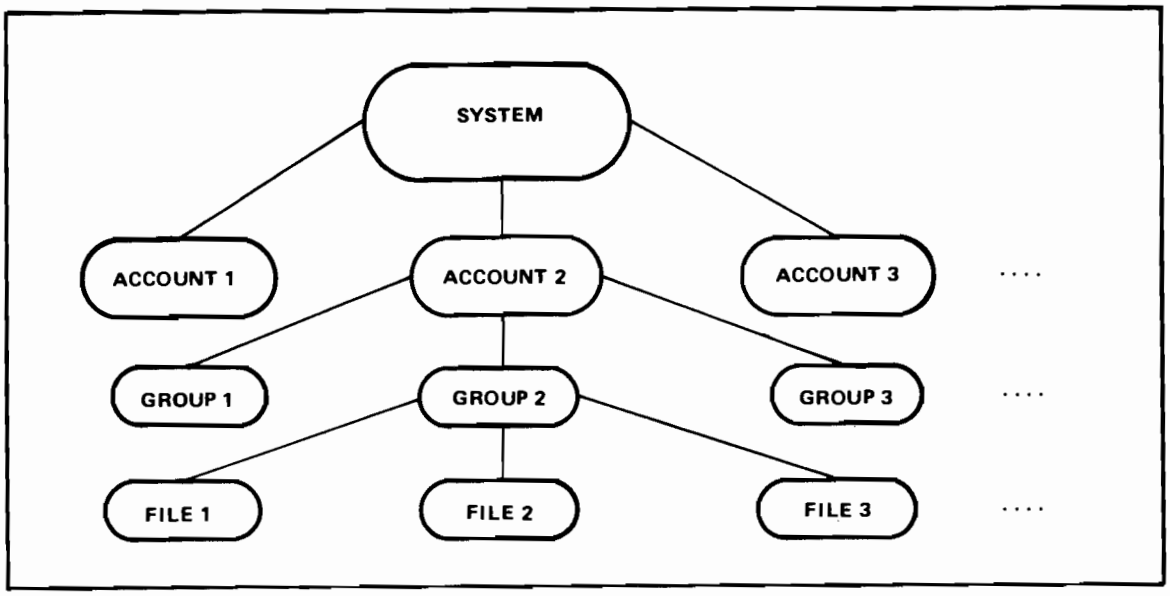


Figure 3-1. Sample Tree Structure

Figure 3-2 shows a user structure example for a system used to provide time-sharing within a large, multi-division company. Each division and various corporate groups would be assigned accounts by the system manager according to their functional needs. In this example, Division A has four defined accounts for Engineering, Marketing, Production and Finance. The Engineering account is shown in detail; there are three users in Engineering capable of accessing this account. Each user has been given a private group as his home group where he will store his private programs and files. In addition, the account manager has defined two special groups for design projects currently being worked on. When any one of the engineering users uses the system for design work on project 1, he will log-on to that group specifically and his time and disc space will be charged to that group (i.e., project). Two other groups have been defined, one for work on schedules and other administrative tasks such as departmental budgets. Also, a public group with no password is defined for storage of general purpose utility programs available to all engineering users.

CAPABILITIES OF THE MPE USER

A typical installation will have a large variety of users. These range from a user who simply wants to run a BASIC program to a systems programmer who may actually be modifying MPE. Therefore a set of capabilities is assigned to each account, and to users within that account, so that the system manager can maintain control over his system, allowing access to special capabilities only to those people who understand how to use them correctly.

The total capability set is divided into three categories:

1. User attributes
2. File access attributes
3. Capability classes for access to MPE resources.

USER ATTRIBUTES

1. System Manager (SM) – created at system generation time. He is responsible for management of accounts and overall system management.

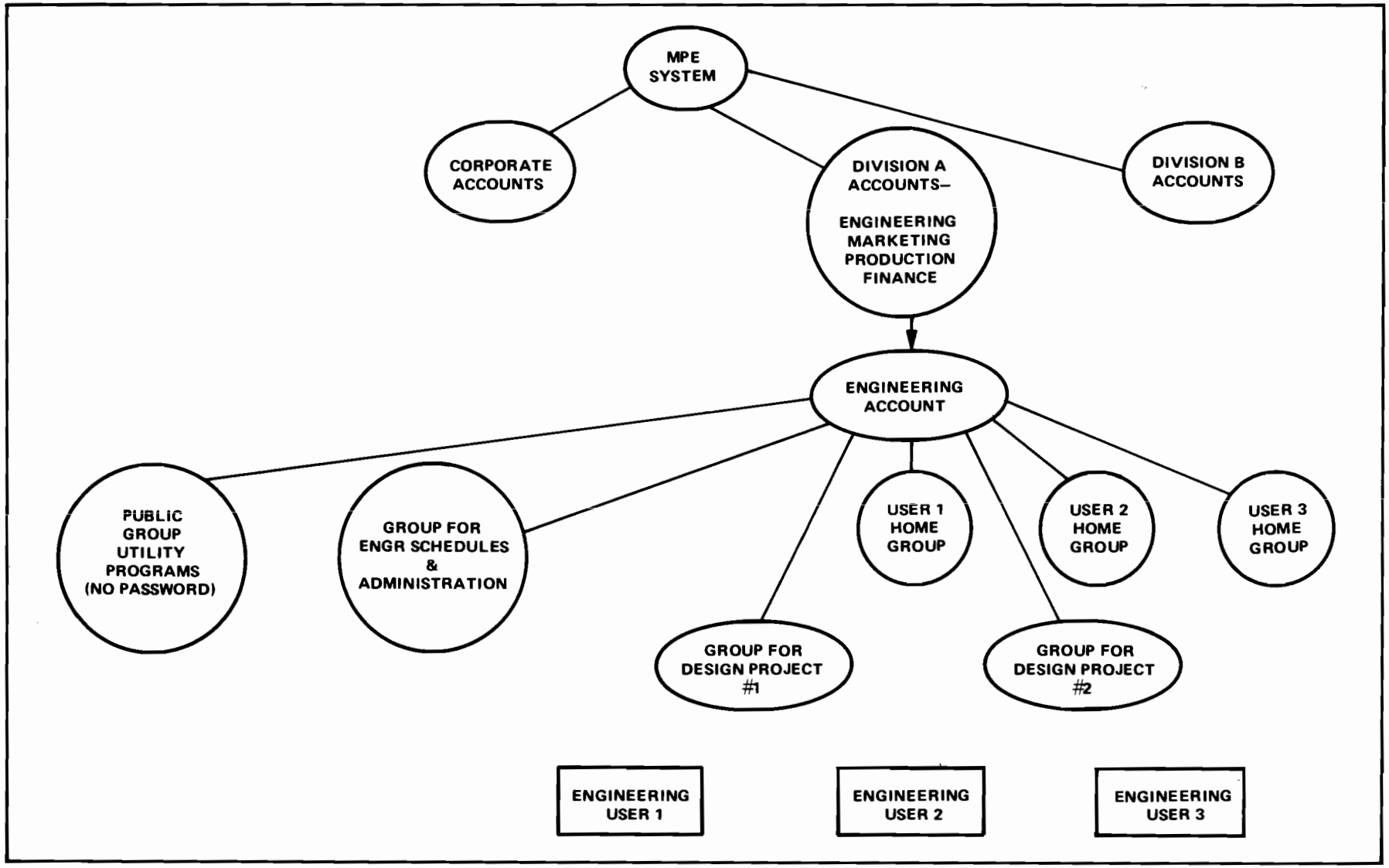


Figure 3-2. MPE User Structure Example

2. Account Manager (AM) – created by another account manager or at account creation time by the system manager. He is responsible for user and group management of his particular account.

3. Account Librarian (AL) – designated by an account manager. The account librarian may have special file access capabilities assigned to him for maintenance of files throughout his particular account.

4. Group Librarian (GL) – these are similar to account librarians, however their special file access capabilities are limited to their home group. Group librarians are defined by the account manager. It should be noted that librarians do not have any explicit power above and beyond a standard account user. Cases may arise, however, where files must be stored within a group, but because of security, only one user of that group should be capable of altering the file. This special access may be given by designating that user as the group librarian and making his file access capabilities greater than that of all the other users within the group.

5. Diagnostician (DI) – designated by the system manager. He is permitted to run diagnostic programs under the MPE System Diagnostic Monitor for on-line check-out of HP System/3000 hardware components.

6. System Operator (OP) – designated by the system manager. This special user is responsible for day to day, operational control of the system.

7. Standard User (SU) – all users with the exception of managers and operators are designated as standard users.

FILE ACCESS ATTRIBUTES

1. Permits saving permanent disc files (SF)
2. Allows creation of private tape files (PT)
3. Allows acquisition of non-sharable devices, such as a line printer (ND)

ACCESS TO GENERAL RESOURCES

The following capability classes refer to the ability to access special MPE facilities.

1. Process handling (PH)
2. The ability to acquire extra data segments (DS)
3. Unused (Reserved for future use)
4. Multiple RIN capability (MR) (See Resource Management Facility)
5. Real Time Capability (RT)
6. Core Residency (CR)
7. Operate in or dynamically acquire privileged mode (PM)
8. Interactive access (IA)
9. "Local Batch" access (BA)

The majority of users in a typical MPE installation will simply have classes 8 and 9. Classes 5, 6, 8, and 9 have scheduling implications for the system which will be described later under "Scheduling Jobs Under MPE". The special capability classes described above are described in detail in the section "The MPE User with Special Capabilities".

The total capabilities of a user from the three sections described above are used in two ways. First, MPE uses these capabilities to restrict command access to the system. Secondly, if a user is running a program which he did not create initially, the user attributes and file access capability for that program will be the same as those of the user who is running it. However, the general resource capability set used by MPE will be that of the creator of the program. Therefore, a user does not need to have the same general resource capabilities as the programs he runs under MPE.

SYSTEM ACCESS

Three types of jobs are recognized by the MPE system and may coexist within the system at any one time. The term job (lower case letters) will be used to reference the total collection of types rather than a specific type. The three types are:

1. SESSION (terminal-oriented job)
2. JOB (batch job)
3. TASK (real time job)

SESSION

A SESSION begins when MPE recognizes and accepts a request from an on-line terminal. After the user is connected to the MPE command interpreter and after a successful log-on sequence, he may then enter commands to use compilers, enter subsystems such as the Text Editor, run programs, or modify his file domain. The SESSION exists until the user types :BYE or the operator forcibly aborts the SESSION.

JOB

A batch request is identified by the recognition of a :JOB command. Once a JOB request is accepted, the MPE command interpreter accepts additional commands from the JOB input device until termination. JOB termination will occur if the following happens:

1. An end of JOB (:EOJ) or new :JOB command is recognized by the command interpreter.

2. The system aborts the batch job due to an error in command interpretation or execution.

3. The system operator aborts the JOB.

Batch jobs may be introduced into MPE in two different ways. First of all, a :JOB command may be recognized from a non-interactive input device, such as a card reader. When this occurs, one of two conditions may apply:

1. The JOB is executed directly as commands and data are read from the input device.

2. The JOB images are "spooled" to a disc file for later processing.

Second, MPE is also capable of recognizing a :JOB command from an interactive terminal operating in SESSION mode. When a :JOB command is recognized from a terminal, the remaining JOB images (until :EOJ) are read and spooled to disc for later processing. Control is returned to the terminal user after the end-of-job command is recognized so that he may continue normal interactive work. MPE returns to the interactive user with a special JOB identification number so that he may later ask for the status of his "spooled" job.

TASK

A TASK is a special freestanding job within MPE used to control certain real-time functions. A full description of TASKS and their capabilities will be found in the section labeled — "User with Real-Time Capability".

A TASK job may be initiated from a JOB or a SESSION if the user has the appropriate capability credentials. The TASK may be initiated either programmatically or directly through the command interpreter in SESSION mode. A TASK runs independently of the job that originated it, and is not affected by the original job being terminated. A TASK job is driven directly by the external interrupt structure of the System/3000. A special identification number is returned to the user when he initiates the TASK so that he may later interrogate its status or possibly terminate the TASK. Termination may only occur by a job with the proper capability credentials or by operator intervention.

SCHEDULING JOBS UNDER MPE

All jobs running under MPE are scheduled by means of a single master queue, ordered by priority. This master queue is divided into areas called priority classes. Each area is bounded by two priority numbers known to the system manager, but not to an individual user. The user is aware of priority classes, but not priority numbers. These priority numbers range between 1 and 255, with 1 being the highest priority.

A logical area or priority class may be a portion of the master queue, a circular subqueue, where all jobs are dispatched in a round-robin fashion, or a linear subqueue, where jobs are dispatched in a normal linear fashion, i.e., the highest priority job runs to completion unless suspended for input/output processing. Figure 3-3 is a graphic illustration of the scheduling scheme under MPE. Each subqueue extending off the master queue is labeled with a two letter mnemonic.

"AS" is a linear subqueue containing jobs that are core resident. Its priority number is 30, and it is accessible only to users with capability class 6.

"BS" is a linear subqueue containing real-time processes which are connected to the external interrupt structure of the System/3000. Its priority number is 100 and it is accessible only to users with capability class 5.

"CS" is composed of two circular subqueues, each having a time-slice quantum determined at configuration time. This subqueue is intended for time-sharing SESSIONS. The reason for two circular subqueues is as follows: Associated with the normal time-sharing subqueue is an I/O subqueue one priority number higher. If a time-sharing job becomes I/O bound, MPE automatically switches it from the normal subqueue to the slightly higher priority I/O subqueue. MPE defines a job as being I/O bound when for a certain number of consecutive times, the job has been suspended because of I/O rather than being timed out at the end of its time-slice. Moving it into the higher priority subqueue gives this job a higher priority until it ceases to be I/O bound. This dynamic rescheduling of highly interactive time-sharing jobs assures quick response at the terminal.

"DS" is composed of two circular subqueues, each having a time quantum or a time-slice determined at configuration time. This subqueue is intended for multi-programming batch jobs. Note that again, a subqueue one priority number higher than the normal batch subqueue is provided for I/O bound batch programs. This assures that batch programs with large amounts of input/output will get their fair share of machine resources when competing with heavily "compute-bound" batch jobs.

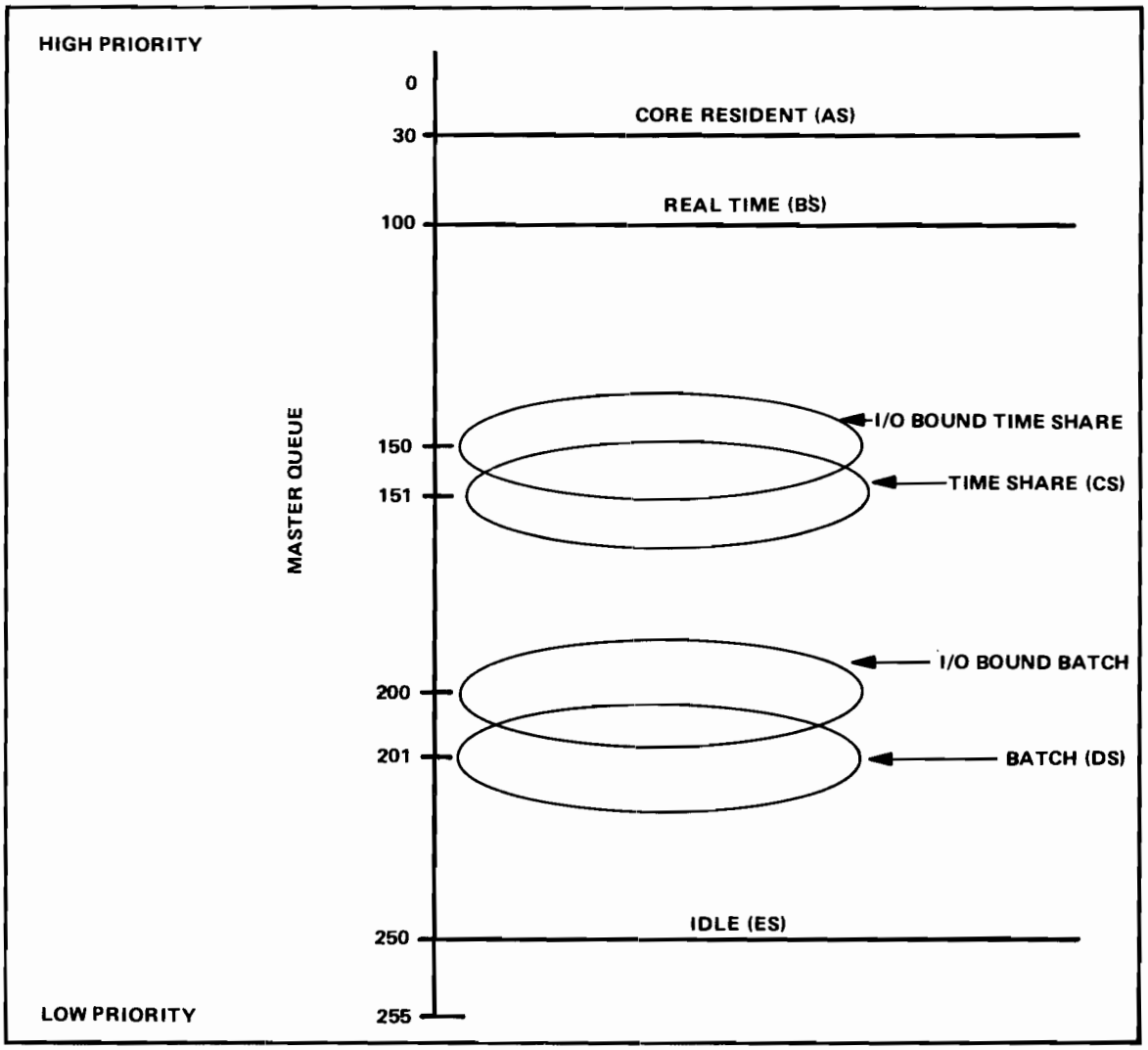


Figure 3-3. MPE Scheduling

The system manager and/or the operator has the ability to:

1. Modify on-line the value of the time quantum of any circular subqueue.

2. Add or delete, at system configuration time, special subqueues to the master queue up to a maximum of 13.

3. Examine the master queue either at configuration time or on-line while the system is running.

4. Open and close any subqueue to make it available or unavailable to users. Only "ES" cannot be made unavailable.

5. Move a particular job from one subqueue to another. This capability allows a batch job to be placed in the time-sharing circular subqueue guaranteeing the job a portion of CPU time along with the rest of the time sharing jobs on the system.

   In addition, users with capability class 5 (Real-Time Capability) have the ability to initially place a TASK on any subqueue and programmatically request that the priority within the subqueue be changed relative to other jobs within that subqueue.

MPE COMMANDS AND INTRINSICS

MPE COMMANDS

When a SESSION or JOB is initiated under MPE, a portion of MPE called the command interpreter reads each command image from a terminal or batch input device, checks its validity, and causes the appropriate action to take place. After a particular action is satisfactorily completed, control always returns to the command interpreter for the next sequence of operations. In addition, the command interpreter may be called programmatically so that certain system commands may be executed from a running program by passing the command interpreter ASCII images which will be interpreted as a command statement.

Note that all commands shown in this manual are preceded by a colon (":"), which is the MPE prompt character. When submitting a batch job, all command statements within that JOB must have a colon in column one; this helps MPE to distinguish the statement as a system command, and also aids in detecting end of data in a batch job stream. During a SESSION, however, the colon will be printed automatically on the interactive terminal by MPE to alert the user that MPE is ready to accept a new command. This prompting by MPE will be shown throughout this manual by underlining the prompt character.

If an error is detected in a command specification, MPE responds accordingly depending upon whether the user is in SESSION mode or in JOB mode. In SESSION mode, MPE responds with an error message indicating an error number and possibly the parameter number that is in error. If the user now types a carriage return, MPE will prompt for the next command, and the user may respecify the command entirely. If however, the user types a colon, ":", MPE will print a descriptive message describing the command error, and then prompt for the next command. If a command error is detected during JOB input, the error message and descriptive message will be printed on the JOB list device. All card images for that JOB will be ignored up to :EOJ or :JOB.

Appendix A of this manual gives a complete listing of all MPE commands, and a brief description of each. Each of the commands referenced in the Appendix will be described in detail in the following sections labelled "file management", "program management", "subsystem access" and "utility commands".

MPE INTRINSICS

Special MPE system procedures called intrinsics are available to users who wish to perform certain system functions programmatically. This interface to the operating system is done through external calls from user programs. These external links from user programs to operating system procedures are satisfied by the MPE segmenter when the user program segments are allocated prior to execution.

Intrinsics are written in SPL (System Programming Language) and follow the rules and constraints of that language. They may be called directly from any SPL program, and also may be called from other languages such as FORTRAN or BASIC.

When an intrinsic is called to perform a system function, two types of error conditions may occur. First, a recoverable error condition will be indicated to the calling program by setting the condition code, carry or overflow bits of the System/3000 status register when the intrinsic EXITs. The condition code will indicate whether the request was granted or not and what conditions existed pertinent to the request. A request to an intrinsic which requires a special capability class not possessed by the calling program will require the system to abort the program. If the user has not specified an appropriate "trap procedure", a batch job will be removed from the system, a terminal session will resume control at the terminal with a prompt for a new command, or a TASK will be removed from the system, and the operator notified.

Users may specify special actions to be taken if an abort error is encountered by initially calling special system trap intrinsics. These will be described under the heading "Utility Intrinsics".

Appendix B contains a complete list of all system intrinsics available under MPE and a brief description of each. The following sections of this manual will describe the operation of each intrinsic in some detail.

The combination of intrinsics and commands allows two levels of access to MPE providing the user with a very powerful mechanism to implement a wide variety of applications.

DISCONNECT CAPABILITY

A SESSION may be suspended and continued at a later time with the command:

:DISCONNECT

The system will respond with the session number given to the user at log-on time, the date and time of day, and the connect time and CPU time used up to this point in the session. The port that this user was logged-in on now becomes available to any other user. The session is now in a suspended state which means that connect time charges are no longer being accumulated. Other resources such as virtual memory space, temporary file space, etc. will be accounted for during the suspension period.

To resume the session, the user logs-on with the same account and user name as the previous session. At that time, MPE will print a message allowing the user to resume the disconnected session. The user has the option to abort the previous session and begin a new one.

Disconnect will also occur automatically if a system power fail occurs or the line drops on a dial-up port. If the user "hangs up" while his program is running, an implicit "break" will precede the disconnect.

Note that to benefit from the disconnect capability, an account must be designated as disconnectable. Since this implies that the system must monitor SESSIONS according to particular user names, duplicately named users are not allowed on disconnectable accounts. For those situations requiring duplicately named users, the disconnect facility does not apply.

FILE MANAGEMENT

A major objective of MPE is to simplify as much as possible the software overhead of dealing with physical devices. This is accomplished through a set of MPE system intrinsics known as the file system. The file system allows user programs to perform I/O by interacting with logical "files" rather than physical devices. Since physical devices such as a line printer typically have complex and inconsistent software interfaces, the MPE file system isolates these from the user by transforming his logical file request into the actual commands and device addresses necessary to interface to a particular I/O device. Within a program, files are addressed by name and accessed through a variety of intrinsics available in the file system. These intrinsics may be called directly from any language or may be made invisible to the user by the file capabilities built into standard programming languages such as FORTRAN and BASIC. In addition, there are various file management commands that are provided for file specification at execute time.

FILE DOMAIN

Disc file access is organized into a three tiered hierarchy. The first level being the account, the files under each account being organized into groups, and each group containing a collection of files. When a user is accessing MPE, he is associated with a particular account, and with a particular group under that account, which forms the base for his local file references. The MPE file security mechanism is flexible enough so that any file in the system is potentially accessible by any user in the system if the normal security provisions have been altered to allow such access.

FILE ADDRESSING

The user addressing a local file, i.e., one stored in his log-on group, specifies a local file name which may be a character string of up to eight characters, and optionally a lockword (password for a file). The two parameters are separated by a slash. An example of a local file reference is:

```
PAYROLL/XZ2AD9
```

If the referenced file is not within the group that the user specifies at log-on time, but rather is in another group within that user's account, the user may reference that file by appending the group's name to the local file reference. An example would be:

```
PAYROLL/XZ2AD9.GRP7
```

If a user wishes to refer to a file which is not within his account, he must also append the name of the account. An example would be:

```
PAYROLL/XZ2AD9.GRP7.ACCT47
```

DEVICE SPECIFICATION

When a user specifies a "device" to the file system, it may be specified either as a device class name or a logical device number. The device class name is related at system generation time to one or more logical device numbers in the System/3000. This device specification is used by programs that wish to make a generic reference to a device, and thus be independent of the logical device number assignments in a given System/3000 installation.

The logical device number is related at system generation to a specific physical hardware I/O address. This number is only used when a particular program operating environment makes it necessary to allocate a specific device for the running of that program. As an example, the System Diagnostician would be asked by a running diagnostic program for a specific logical device number on which to perform a diagnostic test.

ASPECTS OF DISC FILES

Disc files may be created by the user under program control using the system intrinsic FOPEN or by specifying the system command :BUILD. When a file is initially created, it will be divided into a set of equal sized extents, each extent containing an integral number of contiguous disc sectors. With :BUILD only the first extent need be actually allocated at the time the file is opened, thus providing a way for the disc file to dynamically grow as required. Up to a maximum of 16 extents may be specified by the user and any number of extents (up to the maximum specified) may be actually allocated at FOPEN time. If a disc file is opened for sequential access, records may be of variable length. However, if the file is opened for direct access, the records must be a fixed length which is specified by the user at file creation time. In addition, the user may specify a blocking factor to indicate that multiples of the given logical record size will be packed into blocks. The size of the blocks determines the size of the buffer used by the I/O system of MPE for accessing the file. Therefore, whenever a logical record is accessed, the actual unit of physical I/O which takes place will be the block size. This, of course, is masked to the user when automatic buffering is requested.

The mapping of fixed length sectors onto physical disc sectors (which are always 128 words long in System/3000 disc hardware) is a function of the logical record size and the blocking factor. The effective disc space required for N fixed length records of size S (16-bit words) with a blocking factor B is:

Sectors = [N/B] * [(S*B)/128] .

Each expression within brackets is evaluated separately and rounded *up* before the final multiplication. As an example, consider a file of 100 records of 50 words each with a blocking factor of 1:

[100/1] * [(50*1)/128]
= [100] * [1]
= 100 sectors

Since each "block" must be physically aligned on a sector boundary, this choice of blocking factor has resulted in inefficient disc utilization. Note how the efficiency improves if we choose a blocking factor of 3:

[100/3] * [*(50*3)/128]
= [34] * [2]
= 68 sectors

Optimum efficiency occurs if we choose a blocking factor of 5 in which case only 40 disc sectors are needed.

The maximum disc file size available under MPE is limited only by the particular direct access storage device on the system (if the system has an HP2888A disc file available for one complete file, the maximum size would be 46 million characters). The maximum logical record size available in the MPE file system is 64K bytes.

FILE SECURITY

MPE provides a very flexible file security mechanism for accounts as a whole, groups as a whole and files within a particular group. Each of these organizational categories has a set of default file security provisions. These default provisions may be modified by the "responsible user" where the term "responsible" is defined as follows:

Accounts: The only user allowed to change the security provisions for an account as a whole is the system manager.

Groups: The only user allowed to alter the security provisions for a group as a whole is the account manager.

Files: The only user allowed to alter the security provisions of a specific file is the user who created the file.

The file system recognizes five types of access modes. They are:

R READ (General read capability)

A WRITE,APPEND ONLY (User may not modify current contents of file or even read current contents unless READ access is specified)

W WRITE,GENERAL (Includes capability of purging the file)

S SAVE FILES (Pertains to the ability of an account or group having the capability to save permanent disc files)

L STATIC OR DYNAMIC FILE LOCKOUT (Special capability used when several users concurrently access the same disc file)

In addition, there are seven categories of users known to the file security mechanism:

POSSIBLE ACCESSORS

1. System Manager
2. Account Member
3. Account Manager
4. Account Librarian
5. Group Librarian
6. Log On or Home Group User
7. Creating User

} For Account Security

For Group Security

For File Security

Accessors 1, 3, 4 and 5 are users who have been defined as possessing the corresponding manager or librarian attribute. The group for which the "Group Librarian" is librarian is his home group.

For descriptive purposes the notation "X:Y" indicates that all the elements of access mode list "X" are restricted to accessor list "Y" or to put it another way, only members of accessor list "Y" may perform access mode "X". The absence of a specification of any one access mode in a restriction list indicates no restriction for that access mode.

The following default settings are associated with each type file domain.

Accounts: R,A,W,L:1,2

This specifies that the system manager and any account member has read, append, write and locking capability on files stored within this account. The ability to save files is always limited to account members. Note that the above default specification excludes members of other accounts from accessing files within this account.

Groups: R,A,W,S,L:1,3,5,6

All modes of access and the ability to save files are allowed the system manager, account manager, group librarian and the log-on or home group users of this group. Note that this default specification excludes access to the files of this group by other members of the account who do not have this group as a home group or have log-on access to this group.

Files:

There are no restrictions in the default case on files saved within a group, however, access to a particular file must pass the default group and account security.

Note that the total security provisions for a given file are a composite of the security provisions for the file together with the provisions for the particular group and account the file is saved under.

The following examples are given to clarify security provisions:

Example 1.

A user wishes to save a file in his home group. The file is to be completely private to him so that even other users who have log-on access to his home group will not have access to this file. The user specifies:

R,A,W,L:7

which says that only the creating user is allowed access modes read, append, write and lock.

Example 2.

A user wishes to save a file in his home group which only he can modify. However, other users having log-on access to his home group must have "read only" access to the file. The user specifies:

A,W,L:7 (creator has append, write and lock capability)

R:6 (log-on group user has read capability)

Note that this specification overrides the default group security which would ordinarily allow the system manager and account manager access to this file.

Example 3.

A user wishes to create a data collection file within a special account. The security provisions will be such that anyone who has access to the system may access this file and append data to it. Of course, the accessor must be aware of the file name (and possibly a lockword), group name, and account name under which the file is saved (see File Addressing). No one except the creator of the file, however, will be able to read or modify the file.

First of all, the system manager must change the default account security for this particular account, since default account security does not allow access by other users outside this account. The default account security will be changed to:

R,W,L:2

Note that the "write, append only" (A) access mode has been eliminated from the list, therefore, no restrictions apply to it. Secondly only account members have R,W,L access; even the system manager has been excluded.

The group default security must also be modified so that members outside of the account may access a file stored within this group. The account manager redefines the security of the group where this file is to be stored as:

R,W,S,L:6

Note that again no restrictions have been placed on "write, append only" access, but only the log-on group user has general write and read privileges.

For the file itself, if the creating user is the only user having access to the group under which the file is saved, no additional security provisions are necessary. If by chance, however, other users had access to this group as log-on users, the creator of this file could guarantee the privacy of his file by specifying:

R,W,L:7

Note from these examples that by changing any of the default security settings for the account, group or file itself, it is possible to achieve almost any access security requirement.

Many times a need exists to save general purpose or utility programs in public groups or accounts which may be accessed by all users system wide or all the users within an account. MPE allows for this by defining a special system account named "SYS" and a public group named "PUB" which may exist under any account. The security provisions are as follows:

"SYS" account: no restrictions (but remember that only account members may save files in an account)

"PUB" group: A,W,S,L:1,3,4,5,6

These provisions make the "PUB" group of an account readable by any member of that account, but alterable only by the system manager, account manager, account librarian, group librarian or log-on user of the "PUB" group. Note also that the "PUB" group of the "SYS" account may be read by anyone since no restrictions apply to the "SYS" account.

The MPE file system also provides the ability to associate a special lockword or password with any given file. Once a lockword has been attached to a file by its creator, it must be supplied in order to access the file in any way.

To attach a lockword to a file, the user command:

:SETLOCK filereference,lockword

is provided. The lockword may be any alphanumeric string up to eight characters in length, the first of which must be alphabetic.

When the file protected by this lockword is accessed, the lockword may be passed programmatically, or if the user is in SESSION rnode, the lockword may be supplied in response to a prompt by MPE at the terminal.

Note that this lockword protection acts in addition to all of the file security provisions outlined earlier.

Two other commands are pertinent to file security under MPE.

:RELEASE filereference

Releases all security related to the referenced disc file with the exception of the file lockword. This allows a file to be potentially accessed in any fashion by any user of the system.

The opposite command:

:SECURE filereference

Restores all security provisions for the file to their settings previous to :RELEASE.

CONCURRENT FILE ACCESS

If the security provisions of a file allow locking to occur, the user may request MPE to "lock out" other users from concurrent access to this file until this user closes the file. Locking may be either static or dynamic. In the static mode when the user opens a file, he may specify:

a. Fully Exclusive Mode — guarantees that the caller is the only program accessing this file in any manner.

b. Semi-Exclusive Mode — similar to fully exclusive with the exception that "read-only" access is permitted other users.

In both cases, the FOPEN intrinsic will return to the calling program with an error if another user is already accessing the file.

Dynamic locking is also available by specifying the dynamic locking capability when the file is opened. The result of this will be the FOPEN intrinsic acquiring a temporary RIN (Resource Identification Number. See Resource Management Facility). Thereafter, as the program accesses the file, it may call the FLOCK and FUNLOCK intrinsics to temporarily lock and unlock the file using the acquired RIN. All other programs accessing the same file concurrently will also use the FLOCK and FUNLOCK intrinsics to assure that only one program is changing the file at any given instant. Obviously, all programs accessing the file must cooperate with each other in utilizing this capability to preserve the integrity of the data in the file.

PRIVATE DISC PACKS

The system manager may assign specific disc packs to an account. The system manager uses a special command:

:ASSIGN accountname,volumeid[,volumeid] . . . .

Once a volume has been assigned to a particular account, the account manager may then associate any one volume assigned to his account with a group: All files of that group will then always be created on that volume.

FILE MANAGEMENT COMMANDS

THE :FILE COMMAND

This command allows the programmer to make detailed specifications regarding a file at program execute time. The specification may be very simple and only involves establishing a linkage between a logical file reference in the program and a previously saved file. On the other hand, the :FILE command is flexible enough to provide complete specifications for a new file required for this program run.

MPE maintains several pieces of information describing a file. This information falls into two general categories; 1) information pertaining to the file itself such as file size, record size, binary vs. ASCII, etc. 2) Information pertaining to this particular access of the file such as read-only access, dynamic locking, and exclusive versus shared access.

This descriptive information comes from four different sources:

1. If the file is an "OLD" file already permanently saved, the file label will contain all pertinent information describing the file itself.

2. Any :FILE Commands entered before the program :RUN command may contain additional specifications regarding the file itself and this particular access.

3. The call to the FOPEN intrinsic which actually "OPENS" the file may contain additional specifications as parameters.

4. In the absence of information from the above three sources, system-wide default values will be assumed.

To aid the user when specifying certain "standard" files with the :FILE command, MPE has predefined the names of the following files:

$NULL	A "phantom" file which may be designated to "throw away" undesired output from a program run, or simulate an empty input file.
$STDIN	Standard input device; for batch jobs. This would typically be the card reader or for SESSIONS, the user terminal.
$STDINX	Same as $STDIN except command images (colon in column 1) may be read without indicating End-of-Data to the running program. Exceptions are :JOB, :EOJ, :HELLO, :DATA, :BYE and :EOD (End of Data) which may never be read.
$STDLIST	Standard List output device. This would be the line printer for batch jobs, and the user terminal for SESSIONS.

$NEWPASS $OLDPASS	Two temporary "nameless" files created exclusively to pass data from one program to another within the current job stream.
*filedesignator	This construct allows a simple reference to be made to a file designator from a previous :FILE command (see following example #2).

The following examples illustrate some typical uses of the :FILE command.

Example 1.

A BASIC user is accessing MPE from a remote terminal. He knows that during the course of his SESSION he will want a listing of his program on the system line printer. Before calling the BASIC interpreter, he uses the :FILE command to specify a file for printout by entering:

```
:FILE PRINTER;DEV=LP          (assumes LP is defined as a line
                               printer in this configuration)
:BASIC

>FOR I=1 TO 10

> .
  .
  .
  .
  .
  .
  .
> LIST,PRINTER
```

The :FILE command creates a temporary file named PRINTER as a line printer file. Once the user has entered BASIC, he may request a listing on his terminal by simply typing LIST. By entering the optional file parameter "PRINTER" the listing will be directed to that file, i.e., the system line printer.

Example 2.

In this example, two FORTRAN programs are to be executed within one JOB stream. Before the first program is executed, a new file must be created with the following specifications: a binary disc file, direct access with fixed length records of 100 words each. Maximum size is to be 5000 records, divided into 10 extents, however, only one extent, (500 records) is to be allocated initially. The initial FORTRAN program expects this file to be accessed as logical unit number 8. The user wishes to save the file under the name FDATA1. To specify this file the user enters:

```
:FILE FTN08=FDATA1,NEW;SAVE;REC=100,F,BINARY;DISC=5000,10,1
```

This file will be temporily created at FOPEN time and permanently saved under the user's log-on group at FCLOSE time. The first FORTRAN program may be executed, accessing this file for disc I/O. Before the second program is executed, however, the :FILE command must be used again since the second program expects to access this file through logical unit numbers 20 and 21. Rather then reenter the above lengthy specification, the user may simply enter:

```
:FILE FTN20 = *FTN08

:FILE FTN21 = *FTN08
```

and proceed to execute his program.

Example 3.

In this last example a FORTRAN program expects to reference a magnetic tape file for input data as logical unit number 7. The file was previously saved with the name DTAP12. In addition, an "OLD" disc file named FD15/LOCK is to be referenced as logical unit number 8. Finally, list output from this program (predefined by FORTRAN as logical unit 6) is to be "thrown away" during this particular run.

```
:FILE FTN07 = DTAP12, OLD; DEV = TAPE

:FILE FTN08 = FD15/LOCK, OLD

:FILE FTN06 = $NULL
```

MISCELLANEOUS FILE MANAGEMENT COMMANDS

:BUILD filereference

This command creates a new, saved, unextendable disc file with the name "filereference". Record size is by default 128 words, but may be optionally defined to some other value. File size may also be optionally defined. If none is specified, the default value is a system wide constant determined at configuration time.

:SAVE filereference

The SAVE command changes a temporary file into a permanent one, attaching it to the group specified or implied by "filereference".

:PURGE filereference

This command deletes a file from the system.

:LISTD

This command will provide a description of a specific file, all the files within a group, or all the files within an account. The listing can include the following information for each file:

1. File name and access permitted this user.
2. Type of file, record size, number of records.
3. Creation date and the last access date.
4. Extent size and number of extents.
5. The creator's identity.
6. If the user is the creator of this file, the file access security provisions for the file.
7. If the user is the Account Manager, the lockword for this file will be listed.

:STORE fileset, destination

This command stores the specified set of files to a binary file specified by "destination."

:GET filereference1, storedset

GET retrieves a particular file named "filereference1" from the "storedset" defined by a previous :STORE command and attaches it to this user's log-on group.

:RESTORE, storedset

This command restores the entire set of files saved by a previous :STORE command. Any identically named files already present in the user's log-on group will be deleted.

:RENAME oldfilereference, newfilereference

In addition to simply renaming a file within a particular group, this command can effectively move a file from one group to another by specifying a different group as part of the "newfilereference". (See File Addressing)

:GIVE filereference,username

This command changes the ownership of a file named "filereference" to the named user.

:ALTSEC filereference; accesslist:accessorlist . . .

This command alters the file security provisions of the file named "filereference". Access list is a list of access modes (R,A,W,L); accessor list is a list of digits between 1 and 7 specifying to whom access is restricted (see File Security). As many pairs of "accesslist:accessorlist" specifications as necessary will be accepted by the command. Lockword security remains unchanged.

FILE INTRINSICS

The following is a list of intrinsics available under MPE for file management. This list will not attempt to describe in detail the function of each intrinsic, but rather its general capability.

FOPEN This intrinsic is called to initiate access to any file. It establishes an access link between the calling program and the file reference.

In general, FOPEN completely specifies the file from parameters supplied by the calling program and parameters supplied by previously entered :FILE commands. It also performs device allocation, security checking on the file, file label processing, allocates extents of the file if it is a new disc file and constructs control blocks used by the operating system throughout the current access to this file.

FREAD Transfers a logical record or a part of a logical record from a sequential file to the user's stack.

FWRITE Transfers a logical record from a user's stack to a sequential file.

FUPDATE Allows the last accessed sequential file record to be updated. This intrinsic only works with disc files.

FSPACE Moves a logical pointer forward or backward within a sequential file. The displacement, i.e., the number of logical records to be spaced is a parameter passed to the intrinsic.

FPOINT Positions a pointer to any logical record in a fixed-length record disc file. The purpose is to allow record positioning for sequential access within a particular fixed-length record.

FREADDIR Performs a read operation on a specific record of a direct access file.

FWRITEDIR Performs a write operation from the user's stack to a specific direct access record.

FREADSEEK Allows the user to anticipate the need of a particular record before actually performing an FREADDIR. FREADSEEK will seek the particular direct access record and buffer it from the disc to core memory.

FCLOSE This intrinsic is the opposite of FOPEN and signals completion of access to the file. All buffers and control blocks are removed from the system data structure. The job temporary and permanent file domains are altered accordingly.

FCHECK FCHECK is used to transmit details of a file error condition to the user's program. It may be called by the user if he receives an error condition code upon the return from any other file intrinsic.

FGETINFO Returns all pertinent information relating to a file after it has been opened. This includes the complete description of the file characteristics plus this particular access mode.

FCONTROL Performs control operations against the file. Examples might be: write a tape mark on a magtape file, set a terminal timeout interval for the terminal handling driver for all FREAD operations, or rewind a magtape file.

FSETMODE This intrinsic allows the user to activate or deactivate special file access mode options. An example of this is "critical output mode" which indicates that all logical I/O issued to a file should be verified as physically complete before control is returned to the user. Another example is "embedded terminal control" where the user is assuming the burden of controlling the terminal, i.e., he is embedding terminal control characters in records written to a terminal file.

FLOCK This intrinsic will lock the user RIN (see Resource Identification Number). This will effectively lock out access to the file by any other program until the FUNLOCK intrinsic is called by the user.

FUNLOCK This intrinsic unlocks the user RIN which has been locked previously with an FLOCK request.

PROGRAM MANAGEMENT UNDER MPE

It should be noted that this section describing program management applies to FORTRAN and SPL programs only. BASIC programs are handled differently due to the nature of the BASIC interpreter; all commands pertinent to creating, saving and purging BASIC programs are actually part of the BASIC interpreter command structure. For further details, see the HP System/3000 BASIC reference manual.

Three steps are involved in taking a program from source form to an executable state. The first step is the compilation of a user's source program into relocatable binary modules or RBM's. These relocatable binary modules are automatically stored in a specially formatted file called the User Subprogram Library (USL). Step 2 takes the USL file and prepares it into a Program File. Preparation consists of repairing and segmenting the code and defining the initial stack for this program at execution time. The final step is for the system to

allocate entries in the code segment table of System/3000 for all the segments in the Program File and to allocate an entry in the Data Segment table for this program's stack. At this point, the segments are loaded into virtual memory, all external references to library procedures are satisfied and the program is scheduled for execution according to its priority. Note that many of these steps will be invisible to the user during a normal compilation and run. When necessary, however, the more sophisticated user can advance through each of these steps, completely controlling what happens each step of the way.

PROGRAM PREPARATION

:PREP Command – Prepares a program file from a single USL file. Optional parameters may be added by the user indicating that the prepared program is to be sharable by several users and that the stack may be larger or smaller than the default stack size computed by the Segmenter subsystem. In addition the standard capability list may be modified in terms of general resource capability.

:RUN Command – Causes a prepared program file to be allocated and scheduled for execution. Optional parameters that may be specified with the RUN command are: 1) a stack size other than that computed at :PREP time 2) a general parameter to be passed to the program when it begins execution 3) a secondary entry point where execution is to begin, in lieu of the primary entry point of the program, and 4) the order of library search to resolve external references in the prepared program. The normal search sequence is: 1) the log-on group library, 2) the account's public library, and 3) the system library. The user may optionally specify any one of the three libraries to be searched initially for improved efficiency.

:PREPRUN Command – Combines the PREP command and the RUN command. It takes as input a USL file name, prepares the RBMs in that USL file into a program and causes that program to be scheduled for execution. Optional parameters are identical to the PREP and RUN commands.

SEGMENTER SUBSYSTEM

The Segmenter Subsystem has two purposes: The first is to allow a user to manage USL's by adding, deleting, activating or deactivating RBM's within a USL, the second is to manage the various code libraries that resolve external references from the program. The Segmenter Subsystem is entered through the command:

```
:SEGMENTER
```

At this point, the Segmenter begins prompting the user with its own special prompt character which is a dash (–).

USL MAINTENANCE

As has already been mentioned, a program is prepared for execution from RBM's contained in a USL. In most programming environments, the user will want to add new features or otherwise modify his programs from time to time. Also, because of MPE's segmented multiprogramming environment, the user may want to alter the segmentation of a program to improve its run time efficiency. In many systems, recompilation is required to change the segmentation of a program. With MPE this is not necessary since segmentation of a program is modified by simply rearranging the RBM's and then preparing the USL into a new program file.

Each RBM has associated with it a symbolic segment name which is defined at compilation time. When the program is prepared, those RBM's having the same segment name are used to form a single code segment. Each RBM has a unique symbolic name, defining its primary entry point. In addition, an RBM may have several secondary entry points. Each entry point has an activity bit. When a compiler creates an RBM and places it in a USL, all activity bits are in the active state. The user may modify these activity bits at his discretion by using the Segmenter subsystem. When a program is prepared, all RBM's having at least one active entry point will be segmented and placed in the program file.

A convenience feature within the Segmenter subsystem is the ability to have different versions of the same routine within a single USL. The Segmenter commands have an optional index parameter specifying the "i'th" most recent definition of an entry point within a particular routine. By convention, an index of zero stands for the most recent active definition. By using this optional index parameter, the user has complete control over activation and deactivation of entry points within RBM's of various vintages.

SEGMENTER COMMANDS

–BUILDUSL name — This command creates a new USL file which a user may then fill with RBM's or segments from a previously created USL file by using the COPY command.

–USL name — This command designates the current USL file to be modified by the Segmenter commands described in the following pages.

–USE — This command is used to activate the set of entry points in the current USL. A particular entry point may be activated or all entry points in a particular RBM may be activated.

–CEASE — This command deactivates a set of entry points in the current USL with the same guidelines as the USE command.

–PURGERBM — This command is used to purge a particular RBM from the current USL or to purge an entire segment from the USL.

–NEWSEG — This command allows a new segment name to be attached to a particular RBM within the current USL.

–COPY — This command copies a particular RBM or an entire segment from a named USL to the current USL. Segment names are retained.

–LIST — This command lists all RBM's in the current USL.

–EXIT — This command causes an exit from the Segmenter subsystem and returns control to the MPE Command Interpreter.

PROCEDURE LIBRARIES

As mentioned previously, when a program is allocated and scheduled for execution, MPE searches up to three libraries for unresolved external references. In order of search, these are: 1) the library of the user's log-on group, 2) the library of the public group of the log-on account, and 3) the library of the public group of the system account. Each of these three libraries can possess one or two files, categorized as that group's library programs. These files are named PROCLIB, a segmented library (SL) containing procedures in segmented

form and RLOCLIB, a relocatable library (RL) containing procedures in RBM form. When a particular library is searched, the RL file is searched first, then the SL file.

Procedures contained in the SL file are in prepared form, that is, they are segmented. This means that an individual procedure such as the SIN routine may be the only procedure in that segment or may be one of many procedures in the segment. When a particular procedure is needed, the segment containing the procedure is loaded, as are all external references from that segment. Because the segmentation has been pre-defined in this manner, these procedures may be shared between programs, and only one copy will exist at any time in virtual memory even though several users may have required a particular procedure concurrently.

Procedures contained in the RL file are not in segmented form, rather they are in RBM form and must be segmented before they can be loaded with the user's program. The procedures that come from an RL will be placed into a new segment that will be linked to the user program. Since this segment is specifically constructed for a given program, procedures loaded from the RL may not be shared between different programs and may indeed be duplicated for each user requesting them.

It can be seen that the combination of segmented libraries and relocatable libraries allows great flexibility for storing often used subroutines or procedures. Procedures used system wide are normally stored in the segmented library at the system level. Procedures used by only a few users or a single group are stored in RL's at the group, account or system level.

Special Segmenter commands are available to build an SL or RL within a particular group. The default name of PROCLIB will be used for the SL and RLOCLIB for the RL. In addition, commands are available to add routines to the SL or RL, PURGE routines and LIST the procedures contained in either the SL or RL.

SUBSYSTEM ACCESS

THE BASIC SUBSYSTEM

The command :BASIC invokes the BASIC interpreter and causes the special BASIC prompt character to be issued to the user's terminal. This indicates BASIC is ready to accept its own set of commands or program syntax.

In special situations, the BASIC user may wish to specify files other than his terminal or the card reader for command input, program data input and/or output. This may be easily specified by optional parameters entered with the :BASIC command. For example, the command

```
:BASIC,FDATA,OUT25
```

specifies that input data for any programs to be executed will be read from the file FDATA and all program output will be directed to the file OUT25.

ACCESSING COMPILERS

Three commands are available for the FORTRAN compiler and the SPL Compiler. These commands allow a user 1) to compile only, the result being RBM's stored in a permanent USL file. 2) to compile and prepare, the result being a permanent program file containing prepared segments. The temporary USL file created by the compilation process is lost to the user. 3) to compile, prepare and execute. The temporary USL and program files created during the process are lost to the user.

For FORTRAN these commands are:

To compile only :FORTRAN
To compile and prepare :FORTPREP
To compile, prepare and run :FORTGO

Access to the SPL compiler is identical except that the commands are :SPL, :SPLPREP, and :SPLGO.

UTILITY COMMANDS

:TELL username [.acctname] ,message

The TELL command sends a message to the specified user name and optionally the user name under the specified account if the user is not part of the sending user's account. The message will be received with an indication of who the sender was.

:SETMSG [ON/OFF]

This allows a user to control his receptiveness to messages from other users. ON permits messages to be accepted, OFF will disable reception. The default value is OFF when a user logs on. If one user attempts to send a message to another user who is not allowing message reception, the sending user will receive an appropriate indication.

:TELLOP message

This command sends a message to the system operator's console.

:SHOWTIME

The SHOWTIME command prints the date and time of day.

:SHOWCHARGES [*]

This command prints accumulative charges for the log-on group since the last billing period not including the current job. If the "*" is specified, the charges for the current job up to this point are displayed.

:SHOWUSERS [NAMES/NAME=username[.acctname]]

This command returns the number of users currently accessing the system, or if name is specified, their user names and accounts. If a user name is specified, the command will tell the user whether that account user pair is logged into the system.

:SHOWJOB [jobnumber/jobname]

The SHOWJOB command will retrieve the status of a job. The various stages the job may be in are:

a. Spooling in
b. Waiting to be processed
c. Processing
d. Waiting to be spooled out
e. Spooling out
f. Disconnected

In addition, the date and time of submission of the job will be displayed.

BREAK FUNCTION

Depressing the "BREAK" key at the terminal allows a user to interrupt execution of his job. At that point it is possible to:

a. Abort the job with: :ABORT

b. Issue "minor" system command to send messages, examine status, create/delete files, etc. Commands which require termination of the current program will print:

ABORT? YES/NO

before executing the command. Those commands which are permitted during break are noted in Appendix A.

c. Continue the job. Restoring the job to its "pre-broken" state with:

:RESUME

:XEQ filename

This command allows command input to be switched to a file. The command interpreter will recognize the file name as the new job input device and will read commands appropriately interspersed with data until an end-of-file or :EOJ is encountered, or a new :XEQ command is read. This command file capability is especially useful for applications requiring a complicated sequence of commands to be executed in a fashion that is invisible to the user at a remote terminal.

UTILITY INTRINSICS

CONVERSION INTRINSICS

BINARY Converts the ASCII representation of a signed decimal, or octal number to binary. An octal conversion is attempted if the first character is a "%" otherwise a decimal conversion is attempted.

ASCII Converts a 16-bit binary number to its equivalent ASCII representation. The user specifies decimal or octal conversion and the length of the resultant character string.

PRINT AND READ INTRINSICS

When a job is established two devices are recognized:

1. The Job List device which will normally be the terminal for SESSIONS or the system line printer for JOBS.

2. The Job Input device which will normally be the terminal for SESSIONS or the system batch input device for JOBS.

READ This intrinsic reads an ASCII string from the job input device into an array in the user's program.

PRINT This intrinsic will print a message from the user program on the job list device.

PRINTOP This intrinsic is identical to PRINT except the message is issued to the system operator's console.

MISCELLANEOUS INTRINSICS

TIMER Returns a 16-bit count from the system timer with a resolution of 1 msec. This intrinsic is useful for initializing random number mechanisms.

CHRONOS Returns the time in terms of the year, month, day, hour, minute, second and tenth of second.

WHO Enables a program to determine several attributes of the RUNning user, including mode of access, capability class of the user, name of the user, his log-on group, and account, and the logical device number of his job input device.

COMMAND Allows the user program to execute an MPE command programmatically by passing the command image as a parameter.

TRAP INTRINSICS

There are three possible kinds of traps in MPE, and the user may define a special procedure to be executed in case one of the trap conditions should occur. These three conditions are:

a. Arithmetic Traps which occur in the hardware execution of an instruction.

b. Library Traps occurring during execution of a system library procedure.

c. System Traps occurring during execution of a system intrinsic called by the user.

Normally, when one of these traps occur, the execution program is aborted with an error message. By invoking the appropriate intrinsic, the user may specify a particular procedure to be executed whenever a trap occurs. When the special procedure is exited, control will be returned to the instruction following the one that initiated the trap mechanism. Library traps are an exception, in that a trap procedure will be able to specify an abort upon exit of the library procedure.

ESCAPE INTRINSIC

XCONTRAP Provides a special facility for an interactive program executing in SESSION mode. The purpose of this facility is to allow a special procedure to be executed automatically if the user depresses "CONTROL-Y" on his terminal. The user arms this facility by calling this intrinsic and providing the external label of the "special procedure".

As an example, BASIC uses this intrinsic to allow the user to escape from a running BASIC program and return to the BASIC command interpreter.

CAUSEBREAK This intrinsic has the same effect as depressing the BREAK key on a terminal. The executing program will be interrupted and control returned to the command interpreter. The user may execute a RESUME command to initiate execution at the point of interrupt caused by the call to CAUSEBREAK.

RESOURCE MANAGEMENT FACILITY

Sometimes a user may wish to manage the sharing of a common resource such as a disc file between several programs or jobs but the specific resource cannot actually be used by two programs at the same time. A good example of this would be several programs updating a disc file simultaneously. MPE provides a mechanism so that the user may guarantee mutual exclusion of access to the resource. MPE will allocate the user an arbitrary number which he relates to his resource. When a program wants to access the resource, it requests that the number be locked by MPE. When finished with the resource, the program requests MPE to unlock the number so that other users or programs may lock it.

To allocate and manage these numbers in an efficient manner, MPE makes available Resource Identity Numbers (RINs). Each RIN is a number known to MPE but whose meaning is determined by the user who requests it. MPE then provides intrinsics for the user to manage RINs in the following way:

1. A job or process may request that a RIN be locked using the intrinsics LOCKGLORIN (for jobs) and LOCKLOCRIN (for processes).

2. MPE guarantees the request will be granted only if no other job or process already has the specified RIN locked. If another job/process has the RIN, the requestor will be suspended until the RIN is released.

3. Once locked, the intrinsics UNLOCKGLORIN (for jobs) and UNLOCKLOCRIN (for processes) will release that RIN. If another job/process had been suspended in the meantime, it is now resumed using the RIN associated with it.

Two types of RINs are available to the user:

1. Global RINs are global numbers unique within MPE and are assigned to a set of users and freed by command. The global RINs insure exclusive access to a resource between jobs. The commands are:

   :GETRIN

   The parameters that may be specified with this command are pairs of user/account names that will be utilizing this RIN to share the common resource. Once this command has been typed, the system returns with an absolute number between 1 and 1024, which is the actual RIN allocated to this user/account set by MPE. These users may now manage this RIN using the LOCKGLORIN and UNLOCKGLORIN intrinsics previously described. When the RIN is no longer needed, the user who owns it (who issued the GETRIN command) will issue the command.

   :FREERIN rin

2. Local RINS

   MPE also provides RINs to allow sharing of a common resource between processes of a particular job. No commands are utilized to manage these RINs, rather intrinsics are used to acquire the RIN, free the RIN, and lock and unlock the RIN. Up to 255 relative RINs may be requested for a particular job.

THE MPE USER WITH SPECIAL CAPABILITIES

The Standard (Class 0) user has capability classes 8 and 9, that is, interactive access in SESSION mode and "local" batch access (see page 3-4, "Capabilities of the MPE User, Access to General Resources"). To allow for a wide range of users accessing the MPE System, there are more sophisticated capabilities grouped into 7 classes. These capability classes can be assigned to users and accounts in order to allow them to access the corresponding system resource. All users (with the exception of the system operator) will always have at least capability class 0. The special classes are:

1. Process Handling
2. Acquire and use extra data segments
3. Unused – Reserved for future use
4. Multiple RINS – Deadlock Liability
5. Real-Time TASKS, implies Class 1
6. Core Residency, implies Class 5 and 1
7. Operate in or dynamically acquire Privileged Mode

USER WITH PROCESS HANDLING CAPABILITY (CAPABILITY CLASS 1)

A process is an independent entity that can be run within MPE: processes are run on behalf of users and on behalf of the Operating System. Many processes can be running concurrently. The design of MPE is process-oriented: the system deals exclusively with processes (except for interrupt routines and some very central and specialized system functions).

A process consists of a private data area (the stack) used only by this process, a Process Control Block (PCB) that defines the process, and an instruction in a code segment that the process is about to execute. *Note that code segments are used by processes, not owned by them, and may, therefore, be shared by many processes.*

When a user enters MPE, a process is created for him. This process is called a Job Main Process (JMP) in batch mode or a Session Main Process (SMP) in time-share mode. The process is linked into the Command Interpreter which then proceeds to handle user commands.

Under capability class 0, the user is not aware of the Process structure and has no control over it. Under capability class 1, functions are provided which enable the user to create, control, and delete processes. This can be very useful in some applications. For example, this means that class 1 users can have several processes running independently on his behalf and can communicate between them.

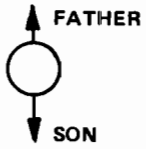
Every process known to MPE is identified by a number called the Process Identification Number (PIN). Most control in MPE is carried out at the process level. A process can run any kind of code (Programs, Procedures, Private Code, Sharable Code . . .) and one of the main elements needed to establish a new process is a Starting Address (that is a Program Label). From this address on, the life of the process follows the sequence of the code until its deletion.

Processes in MPE are logically organized into a Tree structure. This means that each process may have only one ancestor, but may have several immediate descendants.

For a given process, we define its unique ancestor as 'Father' and any of its decendants as 'Son'. It follows that a father may have several sons but a son must have one and only one father.

Each Process carries a set of two logical pointers describing its ties to the rest of the "family":

- one pointer to the father
- a set of pointers to its sons



All these pointers are Process Identification Numbers.

The Tree Structure is demonstrated in figure 3-4.

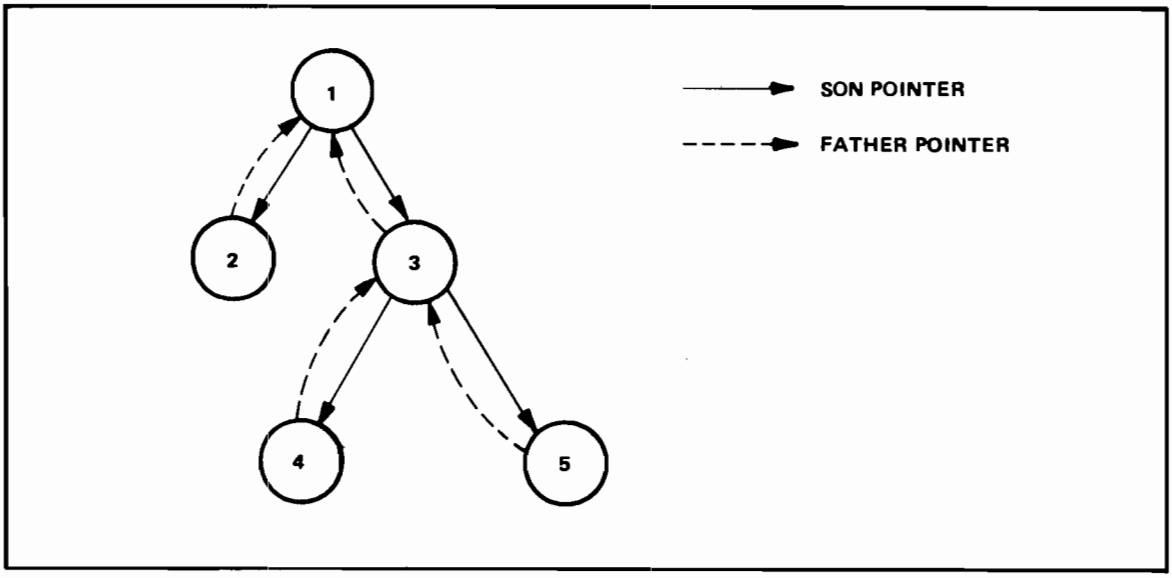


Figure 3-4. Process Tree-Structure

In order to define a finite process tree-structure within MPE, there exists one unique process which does not have a father and thereby qualifies as the root process for the process tree-structure. This process is called the Progenitor.

The Progenitor is the first process established during the initialization phase of MPE. It is the responsibility of the Progenitor, using a set of configuration data specified at System Configuration time, to create its son processes. These processes are defined as System Processes and are used to perform parallel functions on behalf of the System. Such processes may include I/O Processes, Spooler Processes and in particular the User Controller Process. All these Processes may, if required, have their own structure of decendents.

Whereas the Progenitor is the ancestor of all Processes in MPE including System Processes, the User Controller Process (UCOP) is the ancestor of all User Processes currently in existence. The UCOP is thereby the root of the User Process Tree-Structure. The sons of the UCOP are called (User) Main Processes.

The father-son relationship between processes is used mainly to maintain control from top to bottom everywhere in the structure. Roughly speaking, a father is always held "responsible" for what happens to its son: Creation, deletion and some special actions.

Organization of User Processes

When a user establishes himself on the system a Main Process is created for him by UCOP. According to the mode of access, the Main Process can be one of the three types:

- JOB Main Process (JMP)
- SESSION Main Process (SMP)
- TASK Main Process (TMP)

The organization of this structure is shown in figure 3-5.

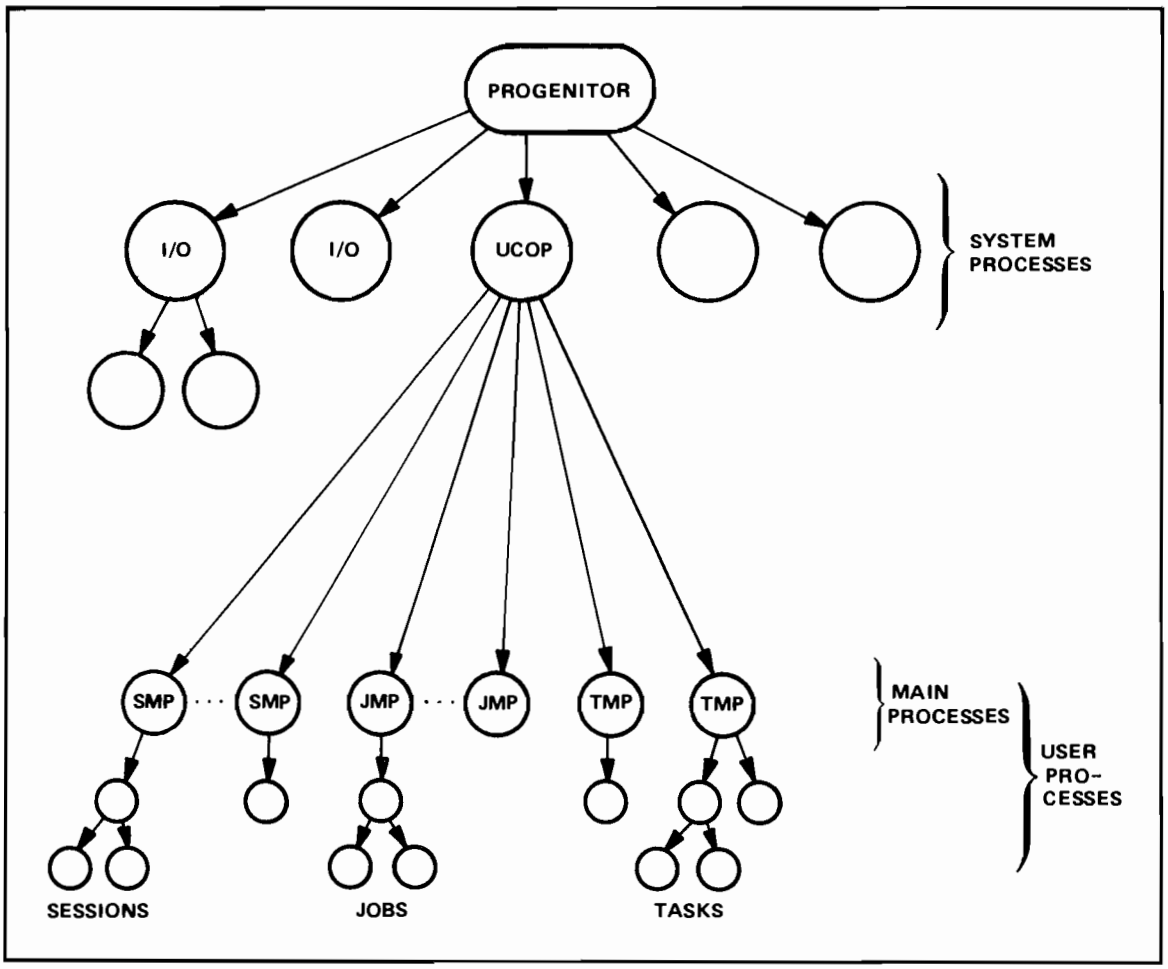


Figure 3-5. MPE Process Structure

Such a distinction results from the different kinds of control that the system provides for those 3 separate entities: JOB is associated with a Batch type of access, while SESSION is for on-line access. A TASK is associated with the Real-Time Mode (see: Class 5 – REAL-TIME). As soon as a given signal is received by UCOP a JMP or SMP is created (depending upon the origin of the signal). A TASK is always created by explicit request from a JOB or a SESSION. The starting address of the JMP or SMP is the Command Interpreter and once the user is validated the Main Process is free to recognize any command.

Process Substates

During its life span, i.e., between its creation and its deletion, a Process finds itself in different substates according to its past and present history as well as its present requirements. Only two of these may be controlled by users: Active Substate and Suspended Substate.

An Active Process is run by the CPU until it suspends itself, terminates, or is killed.

A Suspended Process is not run by the CPU as long as it stays in this substate. In other words a suspended Process is waiting for some kind of a signal which will activate it. When it suspends itself, a Process may specify the origin of its next activation.

A user can control the termination of one of his processes. The termination destroys the process and all its descendents and resets the links of the remaining processes for the SESSION, the JOB or the TASK.

Figure 3-6 graphically illustrates the life cycle of a process beginning with its initial creation from a source file through execution and finally to its termination. Initially, the code exists in a source file which is compiled into the User Subprogram Library (USL). The Segmenter is then invoked to prepare the relocatable binary modules in the USL into a program file. The program file contains code segments and the initial stack-to-be for this process. When the RUN command is used, the CREATE intrinsic is automatically invoked. It and other system intrinsics allocate space in various system tables and essentially make the process alive, but in a suspended state.

The process at this point has been inserted into the MPE scheduling queue, but its active bit has not yet been set, meaning the System Dispatcher will ignore this process until its active bit is turned on. The father of this process may now activate it, causing the active bit to be turned on, which means the Dispatcher will now take note of this process when it looks at the scheduling queue to dispatch the next highest priority process. Before dispatching the process for execution, the Dispatcher must be sure that the data stack is present in core memory. It does this by using a special system intrinsic which makes the data segment present in core memory. This moves the process from the active state to the executable state. Once the process is executable, as soon as it becomes the highest priority process in the scheduling queue, the dispatcher will cause it to execute. The process will now run sharing the CPU with other processes of the same priority, perhaps on a time-sliced basis, if this is a time-sharing process. If a new, very high-priority process is now dispatched, (point A on diagram). The rnemory manager may decide to remove the original process's data stack from core memory and overlay its code segment(s). The process is still active because it is present in virtual memory, but it is no longer executable until the segments are again made present in core memory at point B. The process now continues executing until at point C, it executes an I/O request. In this particular case, the request causes the process to be suspended and moved from core memory into virtual memory on disc. This is not always the case, since the fact that a process is suspended for I/O does not guarantee it will be removed from core memory.

At the completion of I/O, the process again becomes active and then executable, and then runs until it is complete; at which time, it terminates itself by calling the TERMINATE intrinsic. At this point, all resources i.e., system I/O buffers, entries into system tables, process control blocks, etc. are deallocated, that is the process now gives them up and the system has these resources free for any new process. The code for this process still exists as a program file in the file area of the disc and to be executed again, must be created into a new process by using the RUN command.

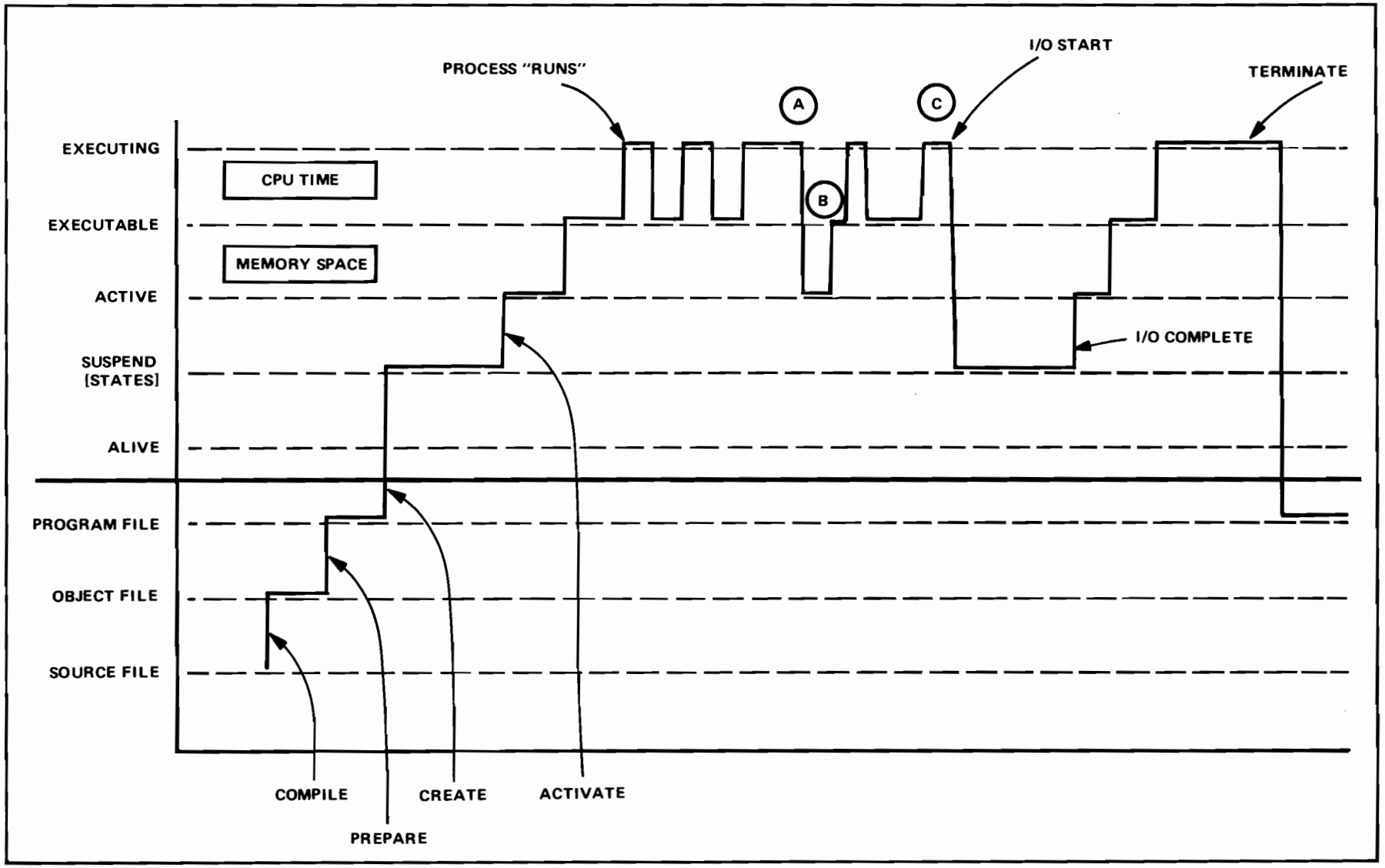


Figure 3-6. Process Life Cycle

Process to Process Communication

MPE provides a means for processes to communicate between themselves. The sending or receiving of information is restricted to either an upward or downward path through the tree. In fact, such communication is allowed only between father and son. Only one transfer is allowed in either direction at any given time.

This method results from the fact that the father is solely responsible for both the existence and actions of his sons. The father created his sons and knows their identification in the process structure; he can therefore, reference them at any time. The sons however, only know their father by the default identification "Father".

Process Priority – Scheduling

Scheduling JOBS under MPE (page 3-6) describes the organization of the scheduling Master Queue. Any process running under MPE belongs to a particular area of the Master Queue depending upon its scheduling requirements. Such an area is called a "priority class" and is logically referred to by a two letter ASCII character identifier. Certain priority classes only are available to users having a particular capability class.

Users having Class 1 capability are restricted in the standard Master Queue configuration to scheduling processes in priority classes CS (Time-Sharing), DS (Local Batch), and ES (Idle).

When a new process is created, its father may specify the priority class of its son. If not, the priority class of the father is taken by default. It is possible, after creation, for either the process or its father to reschedule the process into another priority class at any time.

Intrinsics

The user with process handling capability manages these processes by means of system intrinsics. The system intrinsics provide the following types of control:

a. Process creation; the CREATE intrinsic loads a program, creates a new process which will be the son of the calling process, initializes the stack for this process, schedules the process and returns a process identification number to the calling process.

b. Process deletion; the user has two methods available to delete a process: First the TERMINATE intrinsic deletes the calling process. The last instruction executed by any program should be a call to TERMINATE. This of course is invisible to the class 0 user, but for a class 1 user manipulating additional processes, it is important he be aware of this fact. Secondly, the KILL intrinsic enables a process to delete one of its sons from the system.

   The deletion of a process releases to MPE all of its code, its data segments including the stack, and all other resources it owns, including temporary files. In addition, the deletion of a process always causes its sons to also be deleted.

c. Process Activation and Suspension. As we saw from figure 3-6, once a process is created, it must be activated before it can be executed. The ACTIVATE intrinsic moves a process into the active substate. When a Class 1 user creates a new process as a son, the father process would call the ACTIVATE intrinsic to make the new son process ready to execute. The calling process is suspended once the new process has been activated if the new son has a higher priority. If the father has higher priority the son will become "active" but must wait until his higher priority father suspends in order to run. In some cases a father will create two or three sons which together will complete a certain

operation. In such cases the father must remain in execution long enough to activate all of the new sons. It is legal for a father to activate one of his sons, and it is also legal for a son to activate his "sleeping" or suspended father process. In either case, the process to be activated must expect that it will be activated or the request will be rejected.

The SUSPEND intrinsic is used by the process to suspend itself. The process, once it has called this intrinsic, will be put into the suspended state until it is successfully activated by another process. A third procedure, GETORIGIN, allows a process, once it has been activated, to determine if it was activated by its father or by one of its sons.

d. Communication Between Processes. Three MAILBOX procedures are available to allow process communication between a father process and his sons. The procedures allow a process to send mail, receive mail and test the MAILBOX to see if there is mail waiting.

e. Rescheduling of a Process. The intrinsic GETPRIORITY allows a process to reschedule itself or one of its sons. It may reschedule itself anywhere on the master queue or in a subqueue. See page 3-6 — Scheduling Under MPE. The requested priority must be in accordance with the capability class of this process.

f. Utility Intrinsics. Several intrinsics are available to return the identification number of a process's father process, return the current CPU time that the process has used to a resolution of 1 msec, and also to return identification numbers of any particular son of a process.

USER WITH DATA SEGMENT CAPABILITIES (CAPABILITY CLASS 2)

As explained in Chapter 2 (Hardware Basis for MPE), MPE provides a virtual memory scheme utilizing variable length segmentation. Two primary types of segments exist in the user's program:

a. Code segments containing instructions may be up to 16K words in length; they are re-entrant, never modified and when being used by the CPU, are bounded by the PB and PL registers for protection purposes. All active code segments are defined by two word descriptors in the Code Segment Table of the System/3000 which is recognized by both hardware and software. This table gives the system the ability to dynamically relocate code segments when necessary.

b. Data Segments contain all variable information that a program accesses as it executes. A data segment may belong to a process in which case it is called a "stack" segment, or it may belong to MPE and be used as an I/O buffer. There is another type of data segment available to the user with Class 2 capability, and that is the extra data segment. The normal class 0 user has only a stack data segment associated with his program. The Class 2 user may request up to 4 extra data segments each up to a maximum of 32K words in length. These extra data segments are not directly addressable by the user's program and must be accessed through MPE intrinsics which move data between the extra data segment and the user stack by switching the DB register. The following intrinsics apply to management of extra data segments:

GETDSEG — requests creation of an extra data segment for the calling program. Parameters indicate the length of data segment required and the index of this data segment, i.e., is it the first, second, third or fourth to be requested by this program.

FREEDSEG — destroys a data segment that had previously been acquired using the GETDSEG intrinsic.

DMOVIN – moves a block of data from an extra data segment to the user's stack segment.

DMOVEOUT – copies a block of data from the user's stack segment to an extra data segment.

In addition to a acquiring data segments, the Class 2 user has some control over his stack segment. The initial stack size determined for a user's program may not be sufficient during actual execution of the program, and an attempt to place more data on the stack may overflow the stack. When this occurs, the operating system will increment the stack size by a fixed value. Default values for the size of the increment and the number of times it may occur are specified at system configuration time. The user may use the following intrinsic to change these default values.

STACKOVFLW – overrides the system default conditions by respecifying the number of stack overflows which may occur.

In addition, intrinsics are available to expand the size of the stack area bounded by the DL and DB registers, and to turn on an automatic mechanism for incrementing the DL-DB area when an illegal address internal interrupt occurs. This mechanism provides an automatic stack segment underflow readjustment, whenever this interrupt occurs. It is not foolproof, since the internal interrupt may be caused by bounds violations other than an illegal reference into the DL area. However, it does provide a useful feature for debugged programs, compilers, etc. Parameters to this intrinsic provide the maximum size of the DL-DB area and the maximum number of increments allowable.

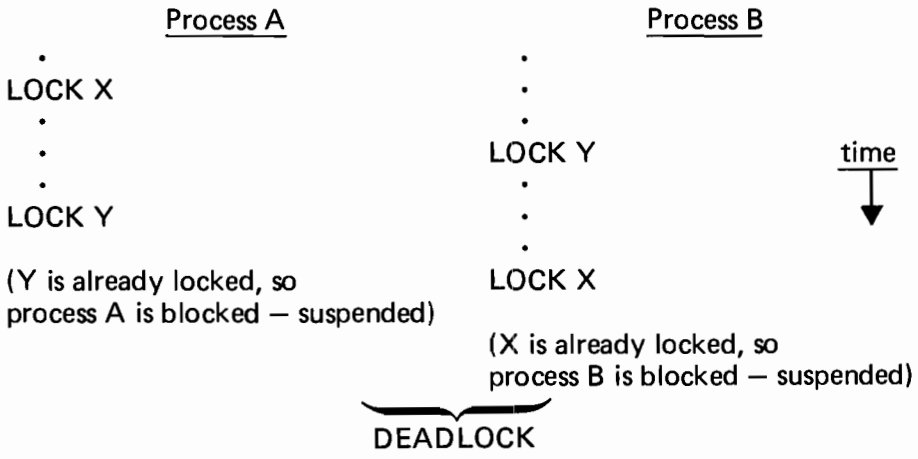
CAPABILITY CLASS 3

This capability class is not used in the initial release of MPE but is reserved for a future capability.

MULTIPLE RINS CAPABILITY (CAPABILITY CLASS 4)

A user with Class 4 capability may lock more than one global RIN at any given time. The system provides no checking and the user can lock as many RINs as necessary. The risk of this is that under multiprogramming it is possible for two programs to reach a deadlock situation.

The following illustration shows how deadlock can occur. Assume process A and process B are managing two RINs, X and Y. If A has X locked and needs Y while B has Y locked and needs X, both processes will be blocked and will have no way of becoming unblocked.



One method to use multiple RINs and avoid deadlocks is for all processes to preorder their resources in the same way, logically speaking. They should then never lock a resource with a lower logical rank than any resource currently locked. Resources can be released in any order.

USER WITH REAL-TIME CAPABILITY (CAPABILITY CLASS 5)

The Class 5 User has three special capabilities available:

1. He may request a higher priority than normal SESSION or batch users to insure that his program will be dispatched quickly.

2. He may connect his program or process to an external interrupt so that the process will be activated when an interrupt occurs. The external interrupt may come from a physical device or the system clock.

3. He may create a TASK which is one or more processes that have been combined into an independent job.

Since all real-time processes are established from either a JOB or a SESSION, it is important to understand that the first two capabilities mentioned above create real-time processes that are dependent on that JOB or SESSION. For instance, if a user in SESSION mode creates a real-time process, and then logs off, that real-time process will disappear along with his SESSION Main Process. The third capability of creating TASKS allows a completely independent job to be connected directly to the User Controller Process (UCOP). This real-time process will continue to function even if the user's Job Main Process or Session Main Process disappears.

A real-time process may be established in one of three ways:

1. The Class 5 user may request that his process be scheduled at "BS" (Real-Time Non-Core Resident) priority (see Scheduling Jobs under MPE). This priority queue is higher than either SESSION mode or batch mode priority. This priority class may be specified at the creation of the process, by the RUN command when the process begins executing, or programmatically during the life of the process itself.

2. A real-time process may be created by connecting a process to a previously loaded interrupt routine. Note that to create an interrupt routine, a user must have even higher capability, i.e., Class 7 – Privileged Mode. If the real-time user is Class 5 only, the interrupt routine he is connecting his process to for activation must be already resident in core in the System/3000.

3. The user may connect his process directly to UCOP making it independent of the creating job and thus creating a TASK. When this is done, MPE will assign a TASK identification number to the new task and communicate that to the user. This task identification number must then be given back to MPE by any new job that requests modification or deletion of that task.

Commands for the Real-Time User

:RUN

The RUN command accepts additional parameters from the Class 5 User. The specific additional parameters are:

1. The name of the subqueue in which the program will be placed to execute its first instruction. The new process will be placed in the last position in that subqueue.

2. A WAIT condition may be specified indicating that the program will not immediately run, but will stay in its suspended state until its Father Process activates it or it is activated by the :GO command.

3. A CONNECT condition may be specified to indicate the new process will be connected to a specific I/O interrupt routine. The interrupt itself will not be armed. This must be accomplished by the Arm Interrupt real-time intrinsic.

:GO

This command allows a program that has been started in WAIT mode to be activated. Additional parameters are available to specify that the program will be made a TASK when it initiates execution. Timing considerations may also be specified for scheduling of this program, such as GO IN X minutes or GO AT a certain time of day. The maximum time quantum that may be specified is 24 hours; resolution is 1 sec.

If the program is not made a TASK, but rather simply run as a Son of a SESSION Main Process or JOB Main Process, execution may be halted using the BREAK key. All process substates are preserved and the process may be restarted using the :RESUME command.

:PUT subqueuename

This command allows a user to specify a new scheduling subqueue for his programs. Either all the processes of his job may be moved to a new subqueue or only the first process created by the :RUN command may be moved. Note that this command applies only to programs running as sons of a JOB Main Process or SESSION Main Process; rescheduling the priority of TASKS requires a separate command (see PUTASK). Once the process has been moved to a new subqueue, execution may be continued by typing the :RESUME command.

The following commands apply specifically to TASKS, that is, real-time processes that have been connected to the user controller process (UCOP). In the following commands "jobid" specifies a TASK number or name.

:SHOWJOB jobid

This command displays the current status of a task such as whether it exists or not, what its substate is, i.e., active or suspended, if it is suspended and will be acivated by the timer, when it will be next activated, and finally, whether it is in the process of terminating itself. In addition, its priority will be displayed, including its priority number and the subqueue it resides on. The linkage specifying the interrupt device the task is connected to is also displayed. Note that this information is dynamic since the task continues execution as its status is being displayed so the information may rapidly become obsolete.

:BREAKTASK

This command is similar to use of the "break" key for a normal program. It stops execution of the specified TASK and saves the substates of all its processes.

:ABORTJOB jobid

This command terminates the TASK in question and deletes its process structure from the System.

:PUTJOB jobid

This command may be used to change the priority of all the processes of a TASK or the Main Process of a TASK. The TASK must not be executing when this command is entered, therefore, the :BREAKTASK command must have previously been entered.

:CLEAR jobid

This command clears the current time scheduling request for a TASK and optionally allows a new specification. This specification includes the ability to run at a specified time of day or to run after a certain elapsed time period.

:RESUMETASK

This command allows the task to be restarted after a :BREAKTASK command has been entered.

In addition to the commands just specified, the real-time user has several intrinsics available for establishing and managing real-time processes and/or TASKS.

Establishing a Real-Time Process

A real-time process may be programmatically created in four ways. Note that this does not necessarily imply that the process is made into an independent task:

1. The user may use the CREATE intrinsic and specify a priority class "BS". Since this subqueue is classified as the Real-Time subqueue, simply placing a process there makes it Real-Time.

2. Use the GETPRIORITY intrinsic to modify the priority of a process to class "BS".

3. Use the CONNECT intrinsic to logically link this process to an interrupt routine.

4. Use the RUNIN or RUNAT intrinsics to link the process to the system timer. These intrinsics are described in the following paragraphs.

Establishing a TASK

A TASK may be established as the son of the User Controller Process (UCOP) by either creating a new process or moving an already existing process structure and attaching it to UCOP. The intrinsics available are:

CREATETASK – creates a new process in a fashion similar to the CREATE intrinsic (see Class 1 – User with Process Handling Capabilities). In this case, however, the process is linked directly to UCOP and a TASK is created.

SETTASK – moves an existing process and all its sons, and attaches it to UCOP. This attachment defines that process to be a TASK. A process may request this transformation for itself or for one of its sons.

Deletion of a TASK

A TASK has a choice of several ways of terminating itself. First of all, the TASK Main Process may call the TERMINATE intrinsic which completely destroys the TASK and informs the operator at the console of the TASK ID destroyed. Another intrinsic is available:

DELETETASK – enables a TASK to be disconnected from UCOP and attached as a new Son of the calling process. A legality check is made so that the calling process has provided the correct TASK Identification Number. An example of this intrinsic is as follows: Assume the initial process structure shown in figure 3-7A with a legal call to the DELETETASK intrinsic by process P_1. The process structure shown in figure 3-7B results – note that the task structure completely disappears and becomes in its entirety a son of P_1.

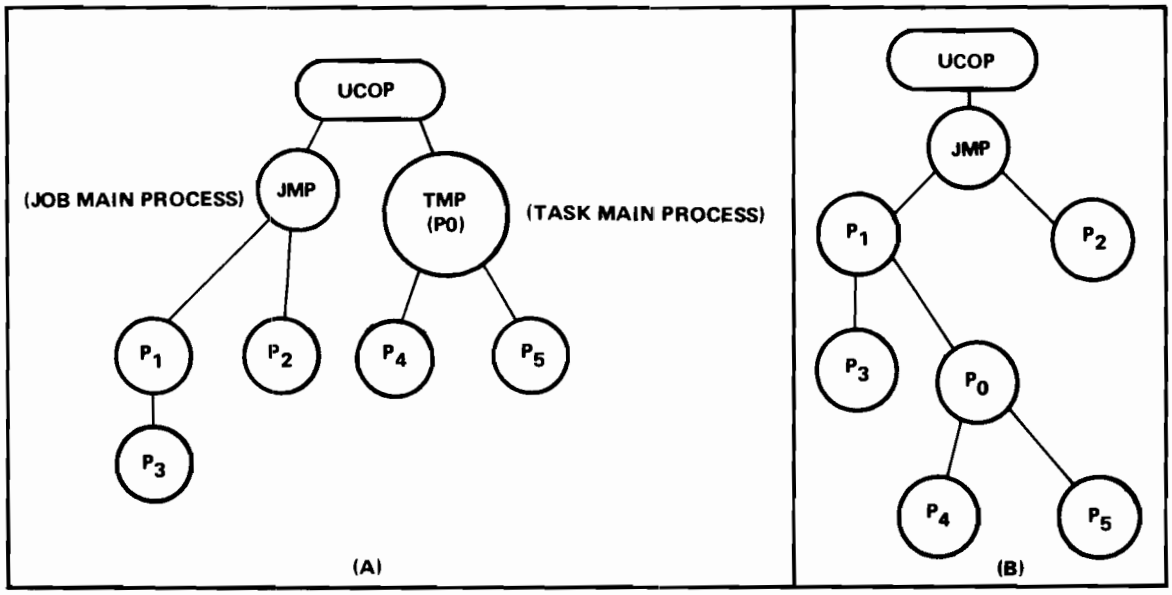


Figure 3-7. Job and Task Structures

Interrupt Routine Connection/Arming

Several intrinsics are provided the real-time user to communicate with the outside world. Remember that we have already made the assumption that interrupt routines that physically "talk to" external devices have already been written and integrated into MPE. The actual writing and debugging of these interrupt routines depends upon Capability Class 7 (see Class 7 – Privileged Mode). When a user connects a process to an interrupt routine, the routine is loaded, frozen in core memory and attached to a special entry in the system Device Reference Table. Special entries in the Device Reference Table for real-time interrupts are defined at system configuration time. No system device will be attached to these entries, thus they are available for any user process and are not controlled by MPE directly. A process may now be activated and dispatched under control of the interrupt routine. Assuming that a process exists, two things must be accomplished to prepare for interrupts: First, a process must be logically linked to the interrupt routine using the CONNECT intrinsic. Secondly, when an interrupt occurs, the routine will request the MPE dispatcher to

activate the CONNECTed process. Activation is allowed only if the process is "Armed". Intrinsics to control this arming are as follows:

CONNECT – Establishes the process specified as a real-time process and connects it logically to the interrupt routine specified. The process must be either the calling process or one of its sons.

DISCONNECT – This intrinsic disconnects the process from an external interrupt routine.

ARMINT – Arms the link between a process and the interrupt routine to which it is CONNECTed. Once it is armed, an external interrupt will cause the process to be activated.

DISARMINT – Disarms the link between the process and the interrupt routine, i.e., the process will not be activated upon receipt of an external interrupt.

Suspension/Activation/Reactivation

A real-time process may be suspended or activated using the same intrinsics are normal processes. In some cases, however, the intrinsics allow more information to be passed for real-time processes.

SUSPEND – Suspends the calling process. The caller may specify where it expects its next activation from, i.e., either its Father, its Son, an interrupt routine, or the system timer.

GETORIGIN – This intrinsic tells the calling process who activated it last.

When a process is connected to an interrupt, another interrupt may occur while the process is still active. The system keeps track of how many times it has attempted to activate the process up to a maximum of four attempts. When the process suspends after servicing the original interrupt, it will be reactivated by the pending interrupt. An intrinsic is provided to manage these pending interrupts:

GETREACT – Returns the number of activations pending for this particular process. The number may not exceed four.

Two intrinsics are available to link a process to the system timer. These are:

RUNIN – Specifies that the process should be activated after the amount of time specified has elapsed. The time specification resolution is 1 ms. The maximum time that may be specified is twenty-four hours.

RUNAT – Specifies that the process will be activated at the "wall clock" time specified.

Scheduling Intrinsics

In addition to the capabilities specifying the priority class "BS" when creating a process or modifying the priority class of a process, an intrinsic is provided to allow a process to jump a number of ranks in the linear subqueue in which it is placed:

JUMP – causes a logical gain of N ranks in priority on the subqueue; N being specified by the caller.

All Capability Class 1 intrinsics are available to the Class 5 User. In addition, the intrinsic:

GETASKINFO – returns a condition code to the calling process, specifying whether the named task is still on the system or not.

USER WITH CORE RESIDENCY CAPABILITY (CAPABILITY CLASS 6)

Core memory in the System/3000 is dynamically allocated to processes on a demand basis by MPE. For some processes, the delay between activation of the process and the execution of its first instruction must be very short. Any time required to fetch a code segment into core, if not present, would be excessive. An example of this would be a real-time process controlling an external device requiring rapid interrupt response. To provide this quick response capability, the Class 6 user may obtain core residency of a) his code segments and stack segment and/or b) any extra data segments that the process may have. The term "frozen" is used in the following discussion to mean core resident. The following intrinsics are available to the Class 6 user:

FREEZEP – loads (if necessary) and locks in core the stack segment of the process as well as all of the private code segments belonging to the caller.

UNFREEZEP – the converse of FREEZEP, this intrinsic unfreezes the stack and all private code segments of the caller.

FREEZEDSEG – requests that an extra data segment be frozen in core. The intrinsic returns the absolute address of the first location of the data segment to the caller.

UNFREEZEDSEG – unfreezes the specified extra data segment so that it is not permanently core resident.

Note that no relationship exists between Capability Class 6 and any scheduling subqueue. Only users with Capability Class 6 may access the subqueue "AS" (Core Residency) through the CREATE or GETPRIORITY intrinsics.

USER WITH PRIVILEGED MODE CAPABILITY (CAPABILITY CLASS 7)

This very special user has all the functions of the System/3000 hardware available to him. He should be fully aware that there are no absolute hardware protection mechanisms to prevent him from damaging other user's environments or causing the system to crash. Certain applications will require this mode, and in general, they will be real-time applications. For the Class 7 user, the extra facilities provided by the privileged CPU instruction set are available plus the capability to write, load and run external interrupt routines under MPE. It is beyond the scope of this document to describe how to write an interrupt routine under MPE, but the following commands and intrinsics are provided to manage this capability.

Two types of operations apply to privileged mode:

1. Permanently privileged – in this mode, programs containing privileged instructions are loaded and executed directly in privileged mode.

2. Temporary privileged – in this mode, programs are initiated in non-privileged mode and the user switches to and from privileged mode using system intrinsics. By "encasing" privileged instructions in this way, system violations will be minimized.

Commands Available in Privileged Mode

The :RUN command loads and executes programs in privileged mode if privileged instructions are contained in the code segments. Temporary privileged operation may be obtained by specifying an optional parameter to this command for a non-privileged program initiation.

:LOADINT – Loads and freezes an interrupt routine in core and attaches it to a specified Device Reference Table entry. This user table entry is declared at system configuration time.

:UNLOADINT – Unloads the interrupt routine attached to a specific Device Reference Table entry.

Intrinsics for the Privileged Mode User

LOADINT – Loads and freezes an interrupt routine in core and attaches it to a DRT (Device Reference Table) entry.

UNLOADINT – Unloads the specified interrupt routine.

GETPRIVMODE – Allows a temporary privileged mode of operation to begin.

GETUSERMODE – Switches from privileged mode back to non-privileged mode.

SECTION IV

SYSTEM MANAGEMENT

Three levels of management are provided by the MPE system.

1. A system manager is a user who has all the capabilities of a normal user and in addition, has final control over access to the system, allocation of resources, modification of system parameters and system monitoring. There may be several system managers with different user names logged onto the system at any one time, but when the system is initially configured, only one exists. This initial manager may then define all other users and/or other managers.

2. The console operator is not really an MPE user. He alerts MPE to a request through the console interrupt button on the System/3000 front panel. When this button is pushed, the console prompts with a special character "=" and expects a single command to be entered. The commands entered in this fashion are special and deal primarily with peripherals, spooling and other hardware system concerns.

3. The system operator is a user with special capabilities and is not associated with the console. Commands that the system operator is responsible for have to do with on-line management of subqueues for scheduling, ordering the priority of jobs which are currently executing, and stopping and starting currently executing jobs.

CONFIGURATION OF AN MPE SYSTEM

The MPE system is initially brought up by a coldload operation. This is accomplished by setting the switch register on the System/3000 with the control byte and device number of the peripheral where the system resides (for example, a magnetic tape unit). The operator then depresses the I/O reset, CPU reset and COLDLOAD buttons on the front panel. The configuration program is automatically loaded into core and is now ready to run.

The configurator runs in one of five modes:

1. WARMSTART – the operating environment for tasks that was present at the time of the last :SHUTDOWN command is restored.

2. COOLSTART – the system is restarted from the disc. The previous operating environment at :SHUTDOWN time is not restored.

3. COLDSTART – the system is read from mag tape. The I/O configuration and system configuration from the tape is used to define the system. This is merged with the directory and files present on the disc.

4. UPDATE – similar to cold start, except that the I/O configuration and system configuration on the disc are used to define the system. In addition, the system Segmented Library and system Relocatable Library on mag tape are merged with those on the disc. This mode would be used to update to a new version of the Operating System.

5. RELOAD – the entire system, directory, files and configuration information are read from magtape.

Actual operation of the configurator consists of a dialogue at the system console between the operator and the configurator. The dialogue consists of approximately 30 questions which the operator answers. Upon completion, MPE is operational.

SYSTEM BACKUP AND RECOVERY

The :SYSDUMP command which is available to the system operator is used to backup the MPE system and user files on magnetic tape. These tapes may be used to reload the system following a failure or to transfer the system to another hardware installation. Optional parameters to this command define a date and an hour of the day. The date is that in which the oldest files to be dumped were last changed; likewise, the hour of the day is the hour on which the oldest files to be dumped were last changed. This date and time would typically be the last occasion of a :SYSDUMP command. A month of 0 implies that the entire library is to be dumped.

If the operator is using :SYSDUMP to create a tape for reloading on a different System/3000, he may specify a new I/O configuration and system table configuration. The :SYSDUMP command in this case will initiate a dialogue with the operator allowing him to make changes to the configuration that will be written on the tape. There are six areas of change, any or all of which may be skipped. These areas are:

1. I/O devices
2. System table sizes
3. System disc allocation
4. Scheduling queues
5. Segment size limits
6. System modules to be included

MANAGEMENT OF ACCOUNTS, GROUPS AND USERS

The System Manager has several commands available to him to create and manage accounts.

:NEWACCT – creates a new account. The following items are specified:

1. The account name and optionally a password.
2. The name of the account manager.
3. A capability list for this account, including whether the account is "disconnectable" or not.
4. The highest priority subqueue that may be accessed by any job under this account.
5. The maximum number of sectors of saved file space the account may use.
6. A maximum limit on CPU time in terms of seconds.
7. A maximum on connect time, including elapsed time for JOBS in terms of minutes.
8. The file access modes permitted to this account.

:ALTACCT – allows the system manager to alter the following account parameters:

1. Password
2. Limits on file space, CPU time, or connect time
3. Capability list
4. Maximum priority subqueue
5. File access modes

:LISTACCT – allows the system managerr to obtain a list of an account's characteristics. If no account name is specified, the characteristics of all accounts in the system will be listed.

:PURGEACCT – allows the system manager to remove an account from the system. Before the account is physically removed from the system, a verification sequence is required by the operator to ensure that accidental destruction of an account does not occur.

:REPACCT – gives the system manager a printed report of the resources currently being used by an account and its groups. Resources shown are file space, CPU time and connect time.

:RESETACCT – allows the system manager to reset the count of a particular resource such as CPU time to zero. Its primary usefulness will be immediately after a billing sequence on the system. If no particular account name is specified, the command resets CPU time and connect time to zero on all accounts.

Once an account has been created, the account manager has commands available to him to create and manage groups and users within his account.

:NEWGROUP – creates a new group within the account. The group ID and an optional password are specified. In addition, limits on file space, CPU time and connect time may be specified. It should be noted that all group limits must of course be less than or equal to the account's corresponding limits.

:ALTGROUP – allows the account manager to alter any particular attribute of a group including its password, limits on file space, CPU time and connect time, file access modes and the volume (i.e., disc pack) on which the group's programs reside.

:PURGEGROUP – allows the account manager to purge a group and its files from the account. A verification sequence is typed by the system to guarantee that the correct group is being purged.

:LISTGROUP – obtains a list of a group's attributes. If no particular group name is specified, the attributes of all the groups within an account will be listed.

:REPGROUP – obtains a printed report of the resources used by a group.

To create a new user, the account manager has the command:

:NEWUSER – creates a new user with a name and optional password, plus an optional home group name. The capability list for the user and the maximum priority subqueue which his programs may access are also specified.

:ALTUSER – allows the account manager to alter the attributes of a particular user including his password, home group name, capability list or maximum priority subqueue for job access.

:PURGEUSER – deletes a user from the account. A verification sequence is typed by MPE to guarantee the correct user name is being purged. All files created by the particular user being purged will automatically be transferred to the ownership of the user issuing the PURGE command. A list of all files that are transferred in this manner will be printed automatically.

:LISTUSER – obtains a list of a user's attributes. If no user name is specified, the attributes of all users within the account will be printed.

LIBRARY MANAGEMENT

The System Manager has two important commands available that allow him to tune the system by permanently allocating a procedure or program in the System/3000 virtual memory. For large, frequently-used routines, this may considerably enhance system performance.

:ALLOCATE – permanently allocates the named procedure in the System/3000 virtual memory. In addition, all code segments that this procedure references will also be allocated.

:DEALLOCATE – performs the opposite function and removes a procedure and its external segments from the system's virtual memory after all processes currently running have finished accessing the named procedure.

ACCOUNTING SYSTEM

MPE automatically accounts for the following resources:

1. CPU Time

   MPE monitors CPU time limits at the account level and at the group level within the account. In addition, for batch jobs a JOB CPU time limit may be specified when the JOB is entered. The default limit is ten seconds.

   At the process level, CPU time for particular processes is accounted for to 1 ms resolution. MPE also accumulates a computed CPU resource number which consists of the process CPU time multiplied by (256-P) where P is the current priority number of the process on the MPE scheduling queue (see scheduling jobs under MPE).

   Finally for JOBS, the system keeps the JOB CPU time limit the current JOB CPU time and the current value of the CPU resource units for this particular JOB. If JOB CPU time becomes greater than the JOB CPU time limit, the JOB will be aborted. If the JOB CPU time is such that the limit for the account or the group under which this JOB is running is reached, the account or group is flagged so that no user will be allowed to log-on under that group or account or enter a new JOB until appropriate action is taken by the system manager or account manager.

2. Connect Time/Processing Time

   Total SESSION connect time or JOB processing time is monitored to a resolution of 1/10 second.

3. The access of certain devices can be monitored in terms of number of logical and physical transfers. The system manager can designate which devices should be monitored.

4. Lines Printed

   A JOB card at JOB input time allows a limit to be specified on number of lines printed which is rounded up to the nearest hundred multiple. If the current printed line counter for this JOB exceeds the JOB limit, the JOB is aborted.

5. Disc Space

   Every time a RUN occurs, the system checks that the user has not already exceeded his log on group's space limit. If the limit is exceeded, the RUN command is refused in SESSION mode; and if in JOB mode, the JOB is aborted.

6. Memory

   For each job the system monitors core memory space as follows:

   1. Maximum stack size is recorded.

   2. Total data space in terms of data segments and I/O buffers is monitored.

   3. The total code space for private code segments is also monitored.

   4. The largest code segment allocated is recorded.

LOGGING FACILITIES

All of the accounting information previously described is automatically logged to a disc file by MPE. This file is a variable record length disc file written through a special system intrinsic. When this file becomes half filled, a message is printed to the operator on the system console. A special utility may then be executed to dump the log file to another file such as magtape or the line printer. Whenever a specific event of interest occurs in the system, the logging facility will create another record in the log file.

The system manager may choose what particular events he wants logged to the file. The following events are defined in the initial release of MPE for logging to the file. Each record written corresponding to a particular event is well documented in terms of its format so that the user may choose to read the file with a program he writes for special analysis of system operation.

LOGFILE RECORD TYPES

1. Head Record
2. Tail Record
3. Job Initiation (JOB/TASK/SESSION)
4. Job Termination (JOB/TASK/SESSION)
5. JOB Input
6. JOB Output
7. Program Completion
8. Monitoring Start
9. Monitoring Stop
10. Monitoring Record (optional)

11. I/O Error
12. Power On (following a Power Fail Interrupt)
13. Directory change (save File, Purge File etc . . .)
14. Configuration change (Time Quantum, Sub Queue open or close)
15. Idle Time Start
16. File Open
17. File Close
18. Time Modified
19. 12-Hour Sampling
20. Modification of User Table

SYSTEM MONITORING FACILITIES

MPE has a special built-in System Monitor which is very useful in tuning an initial installation of MPE to a given load. It also monitors how the installation reacts when a new load is placed on the system. This monitoring program may be initiated by the system operator or the system manager with a special command. Output from the system monitor will be written to a file. This file may be either the standard system log file or it may be a special file designated by whoever initiated the monitoring system.

The Monitor looks at the following system environmental data:

1. The load on the system including the number of processing jobs, number of TASKS, and number of SESSIONS on the system.

2. The load of the I/O system, including the number of requests per queue for each I/O device and the number of I/O channels busy.

3. A core snapshot showing free segments of core available and the fragmentation of core memory.

4. The process structure showing the number of system processes and the number of total active processes in the system.

In addition, operational data is monitored, such as the efficiency of the core management routines, total utilization of the I/O system, and total load on the MPE dispatcher.

SPOOLING

MPE provides the capability of buffering data from/to a user's program from/to a physical unit record device such as a line printer or card reader by use of a disc file. In the "normal" case, a program running under MPE does not communicate directly with a specific peripheral device. Rather, it communicates with a virtual device (actually, a disc file) created by the spooling mechanism. In this way, several jobs may be communicating concurrently with the same physical device, since each job is actually seeing a virtual device through a disc file. The system operator has control over which peripheral devices will be spooled on the system by the user of some special commands (note that the prompt character at the operator's console is a "=" to distinguish console commands from the standard MPE commands):

=SPOOL – allows the operator to specify that a particular device is to be spooled. The operator also specifies the name of a program file which will handle the spooling. In addition, a maximum space on the disc to allow for virtual device files may be specified in sectors.

=STOP – allows the operator to discontinue spooling on a particular device.

=SHOWFILES – displays the current set of virtual files being used by the different spooler programs. A spooler file typically is read once and only once, either by a user program as input data or by a spooler program as program output. Once read, it is deleted from the system, unless multiple copies are required.

CONSOLE COMMANDS

In addition to the spooling commands previously defined, the following console commands are available to the system operator:

=WELCOME – allows the operator to define a welcome message for both JOBS and SESSIONS.

=ACCEPT – defines a device as being allowable for job input streams. This command permits initiation of both JOBS and SESSIONS from the defined device. SESSIONS, however, will be refused from non-interactive devices.

=REFUSE – the opposite command to ACCEPT.

=DIRECTIN – defines a mapping of a requested priority for JOBS onto a particular scheduling subqueue of MPE. If no subqueue is specified, the particular priority listed in this command will no longer be recognized by the system.

=DIRECTOUT – defines the mapping of all jobs requesting a particular output device class onto the physical list devices available in the system. An example of the use of this command would be a job requesting all output to a tape file. This operator command would define the logical device number of the tape to which all such output would be directed. The default case is the standard list device for the job.

=SHOWQ – allows the operator to display the configuration of a particular subqueue or the total master queue. Information such as priority numbers, time quantums on circular queues, and standard or non-standard subqueues are displayed. In addition, if only a subqueue is displayed, all processes in the specified subqueue will be displayed by number. An indication of whether the process is a TASK, SESSION, or JOB and whether it is a main process or not is also displayed.

=WHATJOB – allows the operator to specify a process number. The command then gives back a TASK number, JOB number, or SESSION number to which this process belongs as well as the numbers of all other processes belonging to the job.

=BREAKJOB – stops the specified job and saves the substate of all its processes.

=PUTJOB – allows the operator to place the specified job in a new subqueue and if the subqueue is linear, to place the job at a particular rank within that subqueue.

=RESUMEJOB – restores the substate of all processes for a "broken" job.

=ABORTJOB – aborts the specified job which must have previously been stopped using the BREAKJOB command.

=SHOWJOB – displays the status of a particular specified job.

=DISPLAYJOB – displays the entire process tree structure and process status for a particular job.

=TELL – allows the operator to send a message to a particular job. The message will be printed as soon as possible, but it will not interrupt a write, a read in process or an unreceptive job (by use of the :SETMSG command).

=WARN – equivalent to TELL, but causes the message to be printed immediately, no matter what the status of the user job is.

=SHUTDOWN – takes the system down in an orderly manner. All JOBS and SESSIONS are terminated. The TASK environment, however, is preserved so that it may be restarted when the WARMSTART option of the Configurator is used.

=CONSOLE – allows a new console to be defined by device number. This command must be used with care, since it disables all console activities for the terminal which was previously the console.

=DOWN – specifies that a particular device will be no longer available for general use.

=UP – brings a DOWNed device on line again for general use.

=SESSION – allows the console to be used as a normal time-sharing terminal. Immediately after typing this command, MPE will prompt with a ":" in anticipation of a legitimate HELLO log-on sequence. The terminal may now be used as a normal SESSION terminal, however, it still functions as the console and the "Console Interrupt" button will still result in a "=" prompt and console command request. Messages to the console from MPE will still appear interspersed between the SESSION lines.

SYSTEM MANAGER COMMANDS

When a System Manager has logged onto the system in SESSION mode, he may create and monitor accounts using the commands previously described in this section. In addition, he may monitor console activity with:

:SEE – This command duplicates all traffic at the system console on this terminal. This monitoring may be turned off with:

:BLIND

OPERATOR COMMANDS

A user, under MPE, can have operator capability. This capability enables a remote user to display system information (like subqueues) and exercise a total control over the programs he submits to the system. Users with Operator Capability implicitly have Class 5 capability (Real-Time).

The commands available to operators include:

:SHOWJOB
:SHOWQ
:CONSOLE
:TELL
:WARN

described in Section 4 – Console Commands

:SEE

described in Section 4 – System Manager Commands

:ALLOCATE
:DEALLOCATE

described in Section 4 – Library Management

:SYSDUMP

described in Section 4 – System Back-Up and Recovery.

Other commands available to the user with Operator Capability:

By using the Class 5 capabilities of the RUN command users with operator capability can schedule programs specifying a rank within a subqueue. Processes riding a given linear subqueue are linked together, defining an implicit order (1, 2, 3 . . .). The rank defines the *relative* location of a process in the chain and is not kept as a fixed attribute of the process.

A user with operator capability can "refine" the scheduling of the jobs *he* has submitted to the system through the command:

:PUTJOB (described in Section 4 – Console Commands)

An operator can turn the System Monitor (see Section 4 – System Monitoring Facilities) on and off with:

:MONITOR [ON/OFF], intime

where "intime" (in milliseconds) defines the Time Interval.

The information that is actually LOGGED (see Section 4 – Logging Facilities) is specified with:

:LOG [rectype [,rectype [,rectype. . .]]]

where "rectype" is the number of the log record type. Only those record types specified will appear in the log.

The Operator Capability enables a user to prevent the access of certain subqueues:

:BAR subqueuename

A "barred" subqueue does not accept any processes on it. Processes remaining on that subqueue at BAR time are not removed. Processes trying to access a barred subqueue are automatically scheduled on the ES (idle) subqueue. The ES subqueue cannot be barred.

:ALLOW subqueuename

This command enables a BARred subqueue to be accessed again by processes.

It should be noted that if an interactive job tries to access the CS (time-sharing) subqueue when it is barred, the SESSION will not be accepted.

:QUANTUM subqueuename, time

This command enables an operator to modify the Time quantum of any circular subqueue. The time specified must be expressed in milliseconds.

APPENDIX A

LIST OF USER COMMANDS

CAPABILITY CLASS	COMMAND NAME	DESCRIPTION	EXECUTABLE DURING JOBS SESSIONS BREAK*
0	:HELLO	Initiate a SESSION.	S
	:BYE	End a SESSION.	S
	:JOB	Initiate a JOB.	JS
	:EOJ	End a JOB.	JS
	:DISCONNECT	Suspend a SESSION	S
	:FILE	Define or redefine programmatic file specifications	JS
	:RESET	Permit new programmatic file specifications to take effect	JS
	:EOD	End of data indicator	JS
	:BUILD	Create a file	JSB
	:PURGE	Delete a file	JSB
	:SAVE	Make a temporary file permanent	JSB
	:LISTD	List descriptions of a set of files	JSB
	:STORE	Store a set of files	JSB
	:GET	Get one file from a :STOREd set	JSB
	:RESTORE	Restore an entire :STOREd set	JSB
	:RENAME	Rename a file	JSB
	:GIVE	Change ownership of a file	JSB
	:ALTSEC	Alter security of a file	JSB
	:RELEASE	Release access of file to any user	JSB
	:SECURE	Restore file access to previous modes	JSB
	:SETLOCK	Set a file lockword	JSB
	:PREP	Prepare a USL into a program file	JSB
	:RUN	Load and execute a program	JS
	:PREPRUN	Prepare, load, and execute	JS
	:SEGMENTER	Call the loader subsystem	JS
	:BASIC	Call the BASIC Subsystem	JS
	:SPL	SPL compile	JS
	:SPLPREP	SPL compile and :PREP	JS
	:SPLGO	SPL compile, :PREP and :RUN	JS
	:FORTRAN	FORTRAN compile	JS
	:FORTPREP	FORTRAN compile and :PREP	JS
	:FORTGO	FORTRAN compile, :PREP and :RUN	JS
	:CHART	Call the flowCHARTer	JS
	:EDITOR	Call the EDITOR	JS
	:TELL	Send a message to another job	JSB
	:TELLOP	Send a message to the operator console	JSB
	:SETMSG	Enable/disable message reception	JSB
	:SHOWTIME	Show the date and time	JSB
	:SHOWCHARGES	Show the resources used	JSB
	:SHOWJOB	Show JOB status information	JSB
	:ABORTJOB	Abort a JOB	JSB
	:ABORT	Abort the program currently running	JSB

CAPABILITY CLASS	COMMAND NAME	DESCRIPTION	EXECUTABLE DURING JOBS SESSIONS BREAK*
	:RESUME	Resume a "broken" program	JSB
	:XEQ	Switch job input to a file	JSB
	:GETRIN	Allocate a global RIN	JSB
	:FREERIN	Release a global RIN	JS
5	:GO	Starts a program and can make it a TASK	JS
7	:LOADINT	Load in interrupt routine to assigned DRT	JSB
	:UNLOADINT	Unload an interrupt routine	JSB

*Commands that can be invoked programmatically are those that can be issued during BREAK.

APPENDIX B

LIST OF MPE INTRINSICS

CAPABILITY CLASS	INTRINSIC NAME	DESCRIPTION
0	FOPEN	Initiate access to a file
	FREAD	Read sequential from file
	FWRITE	Write sequential to file
	FUPDATE	Update last-referenced record
	FSPACE	Space forward/backward in file
	FPOINT	Position sequential access in file
	FREADDIR	Read-direct from file
	FWRITEDIR	Write-direct to file
	FCLOSE	Terminate access to a file
	FCHECK	Request file-error information
	FGETINFO	Request general file information
	FREADSEEK	Signal anticipated direct file access
	FCONTROL	Perform file control action
	FSETMODE	Set file-access mode
	FLOCK	Lock file internal resource
	FUNLOCK	Unlock file internal resource
	FRENAME	Change the name of a file
	FRELATE	Return relationship between two files
	LOADPROC	Dynamic load library procedure
	UNLOADPROC	Dynamic unload library procedure
	TERMINATE	Normal terminate of program
	QUIT	Abort termination of job
	BINARY	ASCII to Binary number conversion
	ASCII	Binary to ASCII number conversion
	READ	Read from job input device
	PRINT	Print on job list device
	PRINTOP	Print message on system operator's console
	TIMER	Returns the content of the time counter
	CHRONOS	Returns the actual time
	WHO	Returns user attributes
	SEARCH	Search dictionary of byte arrays
	MYCOMMAND	Format a caller's command image
	COMMAND	Execute a system command
	GETLOCRIN	Allocate local RIN's
	FREELOCRIN	De-allocate local RIN's
	LOCKLOCRIN	Locks local RIN
	UNLOCKLOCRIN	Unlock local RIN
	LOCKGLORIN	Lock Global RIN
	UNLOCKGLORIN	Unlock Global RIN
	XARITRAP	Arithmetic trap recovery
	ARITRAP	Enable/Disable arithmetic traps
	XLIBTRAP	Library trap recovery
	XSYSTRAP	System trap recovery
	XCONTRAP	"Control Y" trap recovery
	SETJCW	Put word in JCW
	GETJCW	Get word from JCW
	RESETCONTROL	Reset "Control Y"
	CAUSEBREAK	Break key simulation

CAPABILITY CLASS	INTRINSIC NAME	DESCRIPTION
1	CREATE	Create Process (Son)
	KILL	Delete Process (Son)
	SUSPEND	Suspend Process (caller)
	ACTIVATE	Activate Process (Son or Father)
	GETORIGIN	Request Activation Source
	MAIL	Test Mailbox Status (Son or Father)
	SENDMAIL	Send mail (Son or Father)
	RECEIVEMAIL	Receive mail (Son or Father)
	GETPRIORITY	Request scheduling priority
	FATHER	Request PIN of Father
	GETPROCINFO	Request Process information (Son or Father)
	PROCTIME	Request Process CPU time (caller)
	GETPROCID	Request PIN of any Son
2	GETDSEG	Get data segment
	FREEDSEG	Release data segment
	DMOVIN	Data segment to stack move
	DMOVEOUT	Stack to data segment move
	DLSIZE	Expand/Contract (DL-DB) area
	ZSIZE	Expand/Contract (Z-DB) area
	RESETDBI	Releases the data segment used as DBI area
	MOVEFROMDBI	Moves data from DBI area to stack
	MOVETODBI	Moves data from stack to DBI area
	GETASKINFO	Check existence of Task in system
5	CREATETASK	Create Task
	SETTASK	Establishes a Task from existing process(es)
	DELETETASK	Takes a Task back to the calling JOB
	GETREACT	Request pending activation count
	RUNIN	Schedules a process to be run in a given time
	RUNAT	Schedules a process to be run at a given time
	JUMP	Scheduling queue jump
	CONNECT	Connect process to interrupt routine
	DISCONNECT	Disconnect process from interrupt routine
	ARMINT	ARM interrupt routine
	DISARMINT	Disarm interrupt routine
	LENGTHDBI	Gets the DBI data segment type and length
	SETDBI	Set up a data segment as DBI area
	CLEARTIMER	Clears the requests to Timer
	RESETDBI	Releases the data segment used as DBI area
	MOVEFROMDBI	Moves data from DBI area to stack
	MOVETODBI	Moves data from stack to DBI area
	GETASKINFO	Check existence of Task in system
6	FREEZEDSEG	Freeze data segment in core
	UNFREEZEDSEG	Release data segment from core
	FREEZEP	Freeze code/stack segments in core
	UNFREEZEP	Release code/stack segments for core
7	GETPRIVMODE	Get privileged mode
	GETUSERMODE	Get non-privileged mode
	SWITCHDB	Switch DB to data segment*
	LOADINT	Load interrupt routine
	UNLOADINT	Unload interrupt routine

*Actually an UNCALLABLE intrinsic; must be called from privileged mode.

Printed in U.S.A.

5952-4469(22)