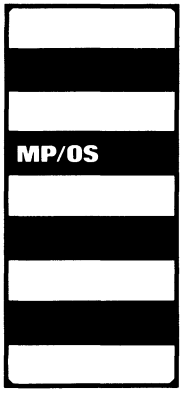
Chapter 5

THE MP/OS OPERATING SYSTEM

An Advanced Operating System for Real-Time, Concurrent Multitasking Execution and Development (And It's PROMable)

Jim Isaak
Data General Corporation



The MP/OS operating system is a member of Data General Corporation's Advanced Operating Systems family. The system provides a single process - multitask environment for both execution and program development on microcomputers and low-end minicomputers. MP/OS is compatible with Data General's Advanced Operating System, which provides a multiprocess-multiprogramming environment for general purpose timesharing applications (see Figure 5-1).

Overview

MP/OS runs on a wide range of processors: the microNOVA MN602 chip, the MBC/2 and MBC/3 single board computers, the MP/100 and MP/200 board and box-level microcomputer systems, the MPT family of intelligent workstations, and the NOVA 4 line of minicomputer systems. Programs developed for the MP/OS environment can also execute under the AOS system on the Data General ECLIPSE line of mid-to-large-size computers; MP/OS programs can be developed on both MP/OS and AOS as illustrated in Figure 5-2.

MP/OS can be configured to match three different execution environments:

1) Program development. This environment, requiring a diskette or disk storage unit, provides a sophisticated set of disk management facilities in addition to the basic MP/OS facilities.

2) Stand-alone operation. In this environment, no mass storage units are required. (Support for disk I/O is available as an option and treats the complete disk surface as a single file.) This environment is used for configurations in which system memory is downline loaded from another system or is loaded from a peripheral device.

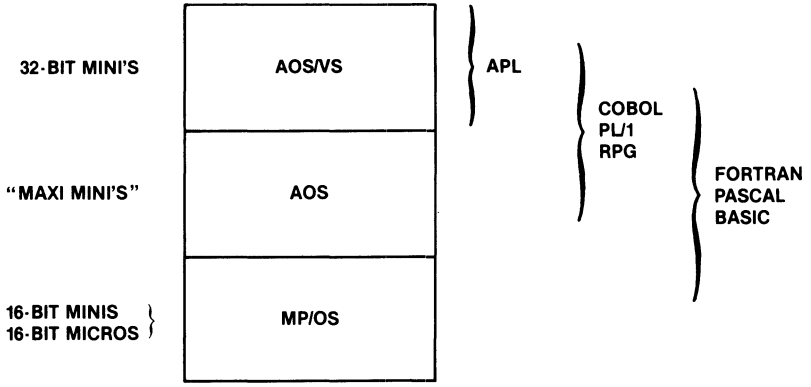


Figure 5-1 Data General's Advanced Family of Operating Systems.

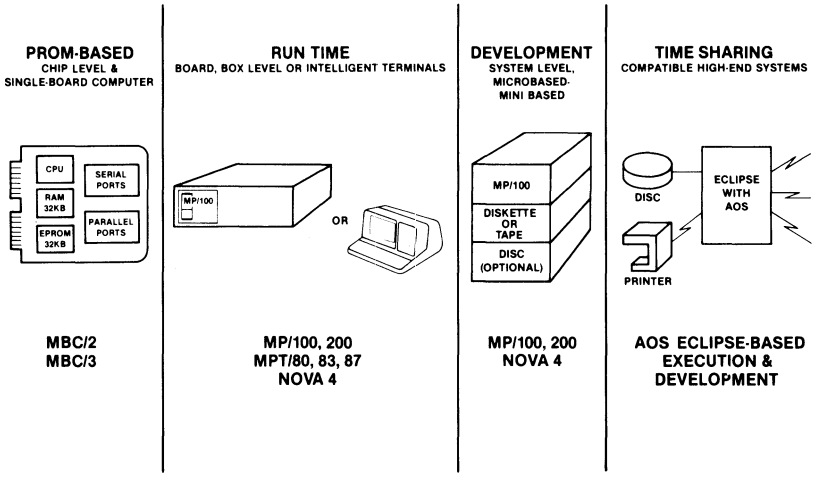


Figure 5-2 MP/OS Execution Environments.

3) PROM-based operation. In this environment, the pure system memory and pure user application programs are linked together for storage in PROM/ROM. This option supports dedicated applications that require high reliability in the face of hostile environments or unsophisticated end users. MP/OS utilities are provided to simplify this program generation procedure so the programmer need not be concerned with the segmentation of memory, the initialization of "impure" memory, or the other location/relocation problems associated with PROM-based execution.

MP/OS supports a wide range of system sizes, from a "bare bones," 5KB stand-alone system (with real-time clock and one console device) to a 64KB program development system (with 8 disk controllers, 8 terminals, 8 line printers, etc.). The program development system requires a diskette or disk to support the program development utilities (macro assembly, relocating binder, sysgen utility, Pascal & FORTRAN compilers, and BASIC interpreter).

MP/OS Hardware

MP/OS supports a variety of peripheral devices (as shown in Figure 5-3), including disk drives, serial ports, serial multiplexers, and printers. Up to eight disk controllers can be configured in an MP/OS system, interfacing to any mix of the following drives:

1) 12.5MB and 25MB Winchester disk drives.

2) 1.26MB diskettes – one or two per controller (diskettes can also share a controller with the Winchesters).

3) 315KB diskettes (one or two per controller).

4) 10MB removable/5MB fixed cartridge disk (may be mixed on the same controller as the 315KB diskettes on the NOVA 4).

5) 20MB removable/10MB fixed cartridge on the NOVA 4.

6) One or two 350KB 5¼" diskettes in the MPT configuration.

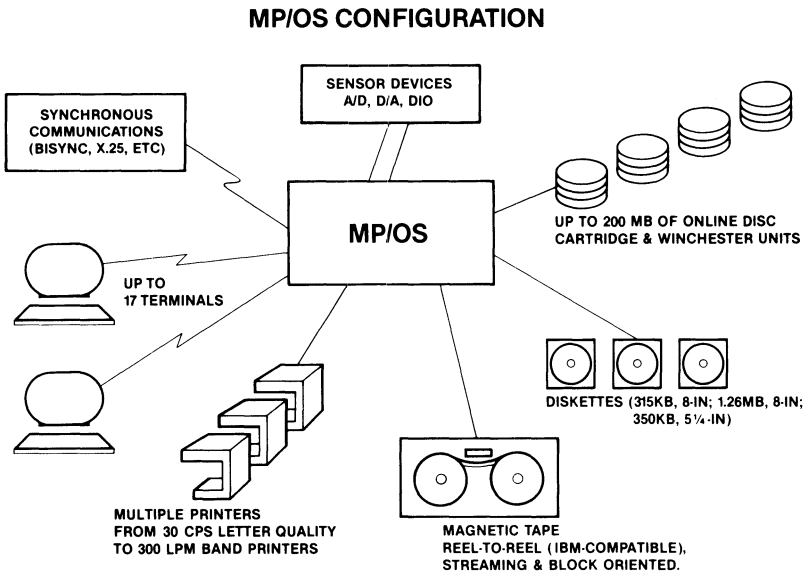


Figure 5-3 MP/OS Users Can Select from a Wide Range of Configuration Options.

In addition, the system supports a maximum of eight parallel printer interfaces and six single line serial ports. (Sixteen serial lines can be supported in an MP/OS system using multiplexer boards.) MP/OS serial interface drivers support:

1) Dual asynchronous/synchronous ports on the MBC and MPT systems.

2) Standard serial asynchronous devices such as video display terminals and printers.

3) Xon/Xoff handshaking protocols.

Finally, library support is available for:

1) Magnetic tape (both 800 and 1600 BPI).

2) IEEE 488 Bus.

3) Synchronous communications (including bisync, 3780/2780, and X.25 networking protocols).

The concept of library support allows programs to use these facilities without incurring ongoing memory overhead for the device drivers that are not in use. This capability is based on a system "IDEF" facility for dynamically declaring user-defined or library-defined device drivers. With this facility, a user program can assume control of interrupts from a peripheral device and initiate the appropriate processing task. MP/OS also provides a "line IDEF" facility that permits a user to assume control of a single I/O device on a system multiplexer unit. In this manner, one port of the 4-line async/sync mux can be used for synchronous communications while three are used for asynchronous. The IDEF concept supports a wide range of control functions – from unique user-defined device interfaces to standard A/D and D/A interfaces.

MP/OS Software

The following languages and language utilities (see Tables 5-1 and 5-2) are supported under MP/OS:

1) MP/Pascal - A threaded compiler implementation with conditional loading for minimal runtime overhead. MP/Pascal contains substantial extensions for systems programming. (Most system utilities are written in Pascal.) These extensions include multitasking support, high level access to all system facilities, random file I/O, include files, separate compilation modules (external procedures and variables), string and double integer data types, and assembly language interface capability. With these extensions, MP/Pascal greatly simplifies the software management task and provides a structured environment for the development of reliable software.

2) MP/FORTRAN IV - An optimizing, multitasking, and reentrant ANSI '66 FORTRAN implementation. MP/FORTRAN provides full access to the system call facilities and includes an assembly language interface capability (useful for interfacing to I/O devices and hardware control facilities).

3) MP/BASIC - A full implementation of minimal ANSI BASIC with integers, reals, strings (up to 32,000 bytes), and a number of the level one extensions such as exception processing, numerical and string functions,

	MP/FORTRAN	MP/Pascal	MP/BASIC
Compiler	yes-optional	yes-optional	interpreter
Integers	yes	yes	yes
Reals	yes	yes	yes
Strings	no	yes (also CHAR)	yes
Arrays	yes	yes	yes
Records	no	yes	no
Separate Modules	yes	yes	"CHAIN"
Overlays	yes	yes	no
Access to System Functions	yes	yes	(via ASM)
ASM Language Interface	yes	yes	yes
Random File I/O	yes	yes	yes
Trigonometric Functions	yes	yes	yes
String Functions	no	yes	yes
Complex Numbers	yes	no	no
Debugging Facilities	symbolic debugger	symbolic debugger and trace back	interpretive interaction
Block Structured	no	yes	no
Run Only	yes	yes	no
PROMable	yes	yes	no

Table 5-1 MP/OS Language Support

and chains. Also, extensions for file I/O and a parametrical assembly language interface are provided.

4) Macro Assembly - Provides machine code translation with a highly advanced macro capability. Supports relocatable code (both pure and impure).

5) Relocating Linking Binder - This utility automatically relocates assembler object files and Pascal or FORTRAN compilation modules. The binder resolves external linkage, conditionally loads the proper system- and language-library routines, and structures and builds the appropriate overlay files. The utility also provides a set

of useful tables and maps detailing the structure and location of the resulting modules. For stand-alone operations, the system is also bound with the user program into a single load module. For PROM-based operations, the code needed to initialize impure memory is automatically added to the pure segment.

CLI	Command Line Interpreter
Speed	Character-Oriented Editor
Bind	Relocating/Linking Loader
Fedit	Binary File Editor
Filcom	File Compare (binary)
Scom	Source Code Comparison (can produce a "differential" file)
Sysgen	Interactive System Configuration
Dinit	Disk Verification/Initialization Utility

Table 5-2 MP/OS Utilities

Both MP/Pascal and MP/FORTRAN compile to produce PROMable code. This code is bound with the required modules from system, language, and user libraries to produce a minimally sized run-time module. The conditional loading of the required modules is a function of the binder, and requires no programmer intervention. Both languages can be loaded with the symbolic debugger for interactive debugging.

The multitasking capability of MP/OS allows a single program to support multiple concurrent tasks. This facility is available in assembly language, MP/Pascal and MP/FORTRAN IV. Both MP/Pascal and MP/FORTRAN are stack driven with re-entrant code. This technique allows multiple tasks to execute procedures or subroutines in a completely asynchronous manner. (This capability is essential to most real-time applications.) The ability to share the heap or common area of memory (and global variables) is also a key to high performance application success.

Security

Security in MP/OS is provided as an element of system management. When bootstrapped, the system automatically executes a program named "CLI." This program is normally the system "command line interpreter." The standard system CLI program automatically executes a user-defined "logon" macro. The CLI program may optionally be a user-generated password/security checking program that ensures authorized system use. If the standard CLI is used, the logon macro can initialize the user environment (where appropriate) and initiate required application program sequences. In the stand-alone and PROM-based environments, a user program is executed immediately after system initialization – no user intervention is required.

In addition to the language libraries and direct system calls, a number of other useful library functions are provided. These functions include magnetic tape support (mentioned previously) and routines for a number of system related functions (e.g., to convert the system clock, stored as seconds since 1900, to Year/Month/Day/Hour/Minute/Second time).

Diagnostics

A number of diagnostic features are built into the operating system. All I/O operations are timed-out and error status is reported when I/O errors are detected. Soft disk errors are also recorded and hard failures cause the appropriate blocks to be allocated to the "bad block" pool. All system calls are checked for valid parameters to prohibit user program errors from destroying the system memory space. Also, the pure segment of the system is continuously checksummed (by the lowest priority system task) to detect any number of possible hardware or software failures.

In addition to these and other integral checks, a set of diagnostics for the CPU and for all peripherals is provided as part of the computer system documentation package. These diagnostics are useful for installation and preventive maintenance functions as well as for remedial service situations. (Both diagnostic tests and reliability tests are provided.)

System Generation

MP/OS system generation is a simple interactive procedure. The modular nature of MP/OS allows selection of global options such as the system execution environment, the processor type, and the number and type of disk drives. In addition, the system generation process permits system tuning (e.g., omitting system calls that are not required – IDEF, killtask). While the system generation process is short, requiring five to fifteen minutes (longer on diskette), it is often unneeded. Most development configurations are supported by the default system and no sysgen is required.

User Interface

The Command Line Interpreter (CLI) provides the most immediately visible user interface. The standard CLI is an English language free-form interpreter that allows abbreviation to the shortest unique command subset. For example, "Filestatus" can be abbreviated to "F," "FI," "FILES," etc. In addition, a "help" feature is provided for on-line access to general documentation. Standard macros (with parameters) are provided to simplify and customize the operator interface for a specific environment. These macros aid in the creation of program sequence **pipelines** (such as "compile-load-and-go") and the macros also simplify operations in the application environment. CLI-level functions provide control of I/O and disk functions (type, copy, locate, delete, mount, dismount) and support full access to the disk file structure (disk status, file status, directory access). Many commands allow switches and/or parameters for sorting, generic file searches, and other options. A number of additional "toolkit" utilities are also available such as binary and symbolic disk file editors and source code and binary file comparison programs.

This same operating environment, including additional tools, is available under the AOS system. In this manner, multi-user development may be performed with the same operating and development environment as that available on low-end microprocessor systems.

MP/OS Real-time Concepts

MP/OS is built around a simple central multitasking scheduling concept. The system contains two predefined tasks – a low priority checksum task (mentioned previously) and a lower priority idle-loop

task. All other system capabilities utilize user program tasks and therefore operate at the user-assigned priorities. Scheduling is performed on a strict priority basis: first for interrupt service software, second for user tasks, and finally for the two low-priority system tasks. For example, if two tasks request disk service and the disk is in use, both **pend** awaiting the **event** associated with disk service completion. The task with the shortest time out is unpended first, providing allocation of resources (CPU and peripherals) based on programmer-controlled priorities. Task priorities can be changed dynamically, providing a technique for critical path control. Also, since each pended task has its own timeout, a mechanism is provided to break deadlocks and to ensure that error conditions are detected within critical time boundaries.

Interrupts

The real-time characteristics of MP/OS depend, in part, on the priority resource allocation previously outlined, and also on the worst case "interrupt/reschedule" disable path in the code. (While two forms of disabling are available to user tasks, only the system "disable" path is documented here.) The worst case disable path occurs when updating the timeout queue. This worst case occurs only when the timeout factors need to be updated after a timeout of the first queue item, or after the addition of a new queue item. The disabled timeout is therefore related to the number of tasks pended at a given time and follows the formula:

200 instructions + (25 instructions x # tasks pended)

For example, with two tasks pended, this disable path has a worst case latency of 250 microseconds on an MN602 processor, or 150 microseconds on an MP/200 computer. Added to the 300 microseconds interrupt latency and the 800 microseconds re-scheduling latency, the total worst case scheduling latency is approximately 1350 microseconds on an MP/200. (These numbers are subject to change based on the revision of the operating system in use, on the hardware in use, and on the effects of user program interrupt disabling.)

The NOVA architecture interrupt structure provides 16 masking levels. These levels allow up to 16 interrupts to be nested "in service". Once a mask bit is set, devices sharing that bit cannot interrupt until the bit is cleared, so that the interrupt service entry code is re-entrant. Both the system and user device drivers (IDEF) use the same system primitive in order to pass control to

an interrupt service task: **interrupt service unpend.** An interrupt service routine may store data directly into a target buffer, or the service routine may pass the data as a message to the unpended task.

Task **unpending** can be performed by priority (the system default), by broadcast (all tasks awaiting an event), or by task ID (requiring the least overhead). Once all pending interrupts are serviced, scheduling of the highest priority "ready" task occurs. When a higher interrupt throughout rate is required, a program may assume control of the interrupt service location and service a high priority device directly. In this case, control is passed to the driver only when multiple interrupts have occurred. Using this technique, which must be programmed carefully, the maximum number of interrupts that can be processed per second on an MN602 unit is increased from 4,000 to 40,000.

Tasks

The basic element (or **thread**) in program control is a **task.** A task has a unique ID, a priority, a designated stack area, and a "kill processing" routine associated with it. Tasks can be created and killed dynamically; the maximum number of tasks (up to 256) is specified at link-time. Since tasks all reside within a user program/process, they can share code and data and communicate in very efficient ways. The stack area ensures that a task's context can be preserved when the task is interrupted (to allow execution of some higher priority task, of an interrupt service routine, or of another user task). A task can change its own priority or the priority of other tasks. A task may kill itself or kill other tasks. When killed, a task is given a chance to run its "kill processing" routine as a method of ensuring that all resources are released (any outstanding system calls for the killed task are aborted).

Inter-task communication is accomplished by means of a pend/unpend mechanism (described earlier). The concept/algorithm used for this "system" pend/unpend function is the same as the concept/algorithm previously described for the "interrupt" pend/unpend mechanism. When pending, a task specifies an event (which may be a system or user event) and a timeout in milliseconds.

A task initiating an **unpend** has three options. The task can unpend all tasks awaiting an event, a specific task (by ID), or the task that has the shortest timeout. This technique allows a direct relation between a task's urgency (expressed as its timeout) and its

opportunity to obtain resources. For direct control over the priority allocation of resources, the task ID variation allows explicit task selection and involves the least overhead. The task initiating the **unpend** receives (from the system) a value indicating the number of tasks unpended.

Each system call has a normal and error return. This feature eliminates on-line testing for error conditions, while still flagging exception conditions. (The **unpend** call allows a task to be unpended at its error return.) On return from a system call (both normal and error returns) a message is passed to the calling task in the form of a 16-bit register value.

Memory Management

The impure (allocatable) area of memory is managed from both ends towards the middle. The system allocates and defines required space from one end while the user program allocates space from the opposite end. A system call to allocate and to free memory is provided to ensure that the space is uniquely allocated. Impure memory is used for both user data storage and program segmentation. Program modules (one or more subroutines or procedures) can be loaded into, and executed from, overlay areas in impure memory.

A set of library routines is provided to manage overlay operations. An overlay segment is automatically initialized to the first overlay module for that segment at load-time. In addition, if the overlay is resident, no I/O is required and the overlay operations file is not opened. These features allow an overlay segment to be used for initialization and permit subsequent recovery of the overlay space (for either data storage or further overlay operations). Also, **overlay load** and **release** functions are provided to maintain a use count for each resident segment. This feature prevents overlaying an area until all the tasks using that area are finished.

I/O Devices and the File System

The MP/OS system provides device independent I/O operations. The set of I/O system commands includes **open, close, read, write,** and **position.** Additional commands exist for specific devices such as disk and character I/O devices. The **open** command associates one of (up to) 64 logical channels with a physical device or

disk file. Further read/write commands simply reference these logical channels. Open channels can be passed between processes during a **swap** or **chain** initiation of another program. In this manner, files can be passed between processes in a transparent fashion. This concept is used for the default input and output channels (typically the console); redirecting default channel I/O to alternate devices is simply a case of initiating a program with the proper channel assignments. (A message area of up to 2047 bytes can also be passed between programs to allow further exchange/sharing of data.)

A good example of I/O system flexibility is evident in the options available in the **open** system call. When opening a temporary file, it is quite easy to overlook creating the file. Also, when a file is created, it is easy to overlook deleting an existing file with the same name. These situations often lead to time-consuming iterations in program debugging. With the MP/OS open call, a number of options are available:

1) EX - Open for exclusive access.

2) CR - Create the file if it doesn't exist.

3) DE - Delete the file if it exists and create a new file.

4) UC - Create the file and return an error if it already exists.

5) AP - Append to the end of the file (position to end).

6) NZ - Do not zero file blocks as they are allocated.

These options (all documented with the call) allow a programmer to easily select the correct call variation.

File Structure

The MP/OS I/O and file structure consist of a hierarchy of directories. The root node of this directory tree is the table of system devices (the device directory). Included in the device directory are disk and diskette devices that may themselves contain further directory structures. Disks may also be "opened" without "mounting", which signifies direct access to all sectors on the disk (with no assumed directory structure). There is no limit to the nesting of directory structures on a disk. Each directory contains files that may be either data files or program files. The full name of a specific

file is called a **pathname;** a pathname always starts at the device directory and proceeds down the directory tree.

This directory structure supports separate development and operations directories on the same disk. Two versions (revisions) of a program (with identical names) can coexist in the system (one in each directory). Because of the separate directories, the production versions are protected from the versions under test. During execution, a directory may be selected as the working directory. With this technique, file names specified by untested programs access the "experimental" directory; files with the same names in the "production" directory are protected since only production software can access them.

A search list of directories is also provided to allow automatic searching for files located elsewhere in the directory tree. The "logon" facility of the CLI and CLI commands (to set the search list and to select a working directory) provide a quick and easy way to establish the operating environment. Often, the search list is set to include directories of utility programs and/or useful macros. In this way, users need not be concerned about where these common facilities reside.

All files are organized in a simple, but powerful hierarchial structure as shown in Figure 5-4. The primitive allocation unit is an **element** that ranges from 512 bytes to 16 megabytes in size. An element is a contiguous segment of disk space. A file may consist of multiple elements, with an **index** block of element pointers (allowing these elements to be allocated anywhere on disk). Moreover, the index blocks can themselves be indexed so that file structures as large as 2048 megabytes can be managed. While, in practice, file size is limited by the physical size of the disk drive, there is virtually no limit to the size of disks that can be supported using this file structure. This file structure provides a substantial amount of flexibility in file allocation, since files can be extended as needed with minimal dependence on locating sufficient contiguous disk sectors. In addition, this technique permits very fast random access as discussed in the following paragraph.

The advantage of contiguous file structures, besides simplicity, is the ability to access a random position in the file with a single seek. A contiguous file has no index to search and no linked list to traverse. With MP/OS files, the file descriptor maintains a set of file element pointers that point to data elements or to index blocks. This descriptor can be kept in memory, if desired, to eliminate a seek operation. This technique allows access to a maximum

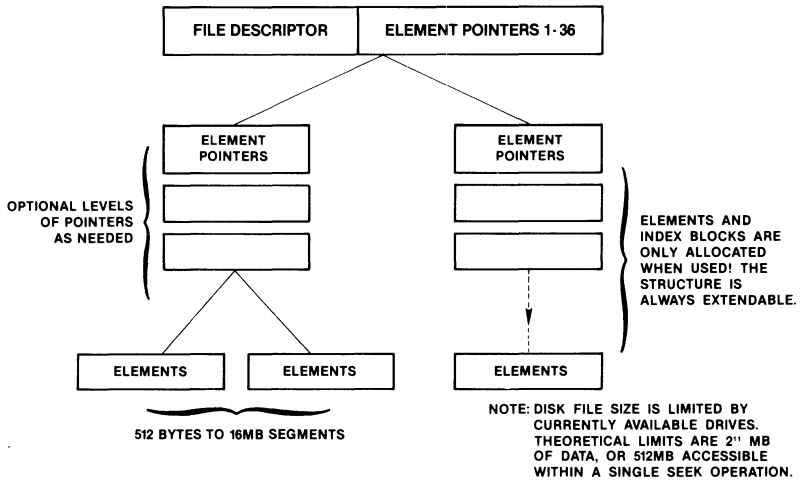


Figure 5-4 An MP/OS File Statement.

of 512 megabytes of data in a single seek. If index blocks are used, these blocks are maintained in a buffer pool along with data elements that require deblocking. (The buffer pool is managed by an LRU algorithm.) The result is a highly flexible and efficient management scheme for even the largest files.

File access is controlled by means of the standard **read** call and a random file **position** call. The file **position** call allows positioning to any byte within the file using a 32-bit pointer. The **read** call allows a fixed-size block read or a data-sensitive read. The data-sensitive read option is particularly useful for text files and/or terminal I/O. If a fixed-sized block read requests the transfer of an even number of 512-byte disk blocks to a word aligned buffer, a direct transfer is initiated from the disk to the specified buffer. Otherwise, deblocking is required and the requested block is read into a system buffer.

An ISAM file library provides a mechanism for managing multiple key indices as well as for managing a data record file. The significant facilities include multiple key access, logical and physical record deletion, keys that need not appear in records, fixed-length key fields, variable- or fixed-length records, partial key indexing (first key with a partial match is returned or first key greater than the specified key is returned), and duplicate keys. Primitive access commands are provided so that many index use variations are possible. When a keyed access is made, the entire

record may be read or the four-byte indexed value may be obtained. For some applications, this indexed value may be the only data needed. (The index value may be something quite different from a simple record pointer.)

File Security

All files (including directories) have **access control attributes.** These attributes include read-only, write-only, permanent, and attribute protect. Attributes prevent inadvertent loss of data and/or deletion of important files. By use of the logon control and the file attributes, the security of file system data can be controlled.

Beyond the organization of the file structure, a number of substantial reliability concepts have been built into MP/OS. First, the key disk structures are located through indirect label vectors to ensure minimal dependence on the error-free quality of specific physical disk sectors. Moreover, these structures are duplicated and checksummed. In this manner, errors can be detected and corrected. Second, the directory structure is a logical association, rather than a physical tree. If a directory index is destroyed, the referenced files are not lost. Moreover, the directory can be rebuilt from information contained in the file descriptors. Third, the allocation of sectors is based on a bit map of free sectors. "Bad sectors" are detected at initialization. These sectors are "allocated" and listed on a bad block list. Finally, the allocation of free blocks and release of used blocks is managed so that a system failure during allocation cannot result in the duplicate allocation of a block. If a system failure occurs, this event is detected when the system is re-booted. At that point a "fixup" program is automatically run, and the following procedures are initiated:

1) All key structures are validated; any damaged structures are rebuilt (all structures can be rebuilt even if both copies are damaged).

2) All directories are checked and rebuilt if needed.

3) All file index structures are checked to ensure that they point to valid (allocated) blocks and to ensure that these blocks are not allocated twice.

4) The bit map is rebuilt if required.

After the disk integrity is assured, the operating system continues into the CLI entry as previously described.

Summary

The MP/OS system provides a real-time, multitasking execution environment. MP/OS operates across a range of systems from microprocessor chips through minicomputer systems. This software is compatible at the object code level with the Advanced Operating System for software portability and cross development. MP/OS supports both stand-alone/PROM-based operations and full program development facilities on microprocessor-based systems. A highly efficient and reliable file system is provided, with support for a wide range of physical devices. MP/OS system design stresses simple operation, extensive system generation capabilities, system programming support, and user-defined device interface capabilities. A comprehensive toolkit for program development, software management, program debugging, and utility operations is also provided. MP/OS represents a system developed using modern operating system and software engineering concepts. MP/OS is designed as a foundation for the applications of the 1980s.

Jim Isaak is manager of Industrial Real-Time Marketing at Data General Corporation, and handles marketing strategy and product planning for industrial automation, communications markets, and energy/resource development. He is a member of the IEEE and ACM, and received his M.S.E.E. in Computer Engineering from Stanford University.