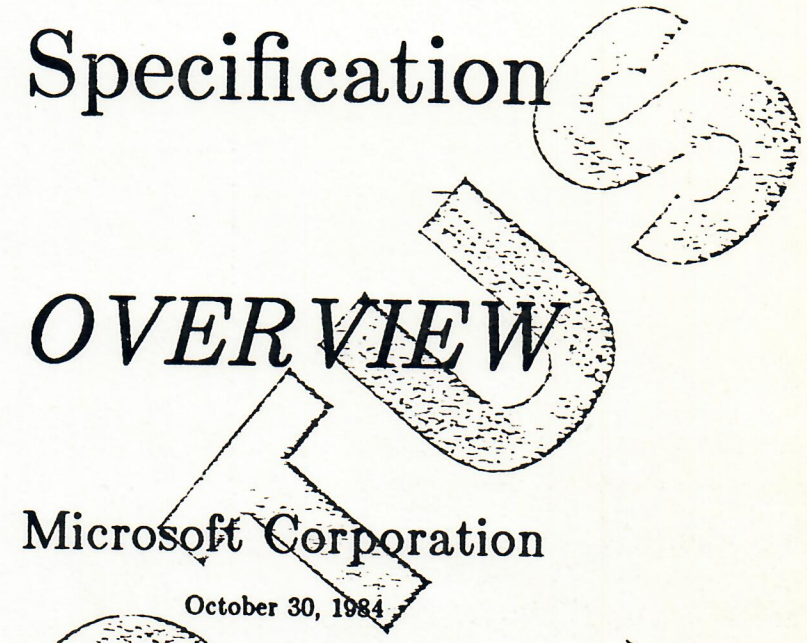
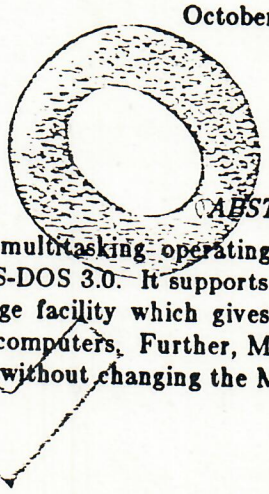
Microsoft Multitasking MS-DOS Product Specification



OVERVIEW

Microsoft Corporation

October 30, 1984



ABSTRACT

MS-DOS 4.0 is a multitasking operating system, developed from and downwardly compatible with MS-DOS 3.0. It supports true multitasking as well as a multiple concurrent screen image facility which gives the user the illusion of and benefits from many independent computers. Further, MS-DOS 4.0 allows most existing MS-DOS 2.0 applications to run without changing the MS-DOS 4.0 multitasking environment.

1. OVERVIEW

1.1. Introduction

As personal computer and office automation software becomes more sophisticated, multitasking capability becomes a requirement. Multitasking improves system performance by allowing the creation of background tasks such as print spooling. Further, it allows the machine to participate in a network environment by running network file servers, mail system daemons, and so forth. Finally, it allows the user to maintain several independent *threads* of activity, switching back and forth between them at will.

The purpose of this document is to give an overview of the features of Microsoft's multitasking operating system, MS-DOS 4.0. These features are discussed in more detail in the following specifications:

- *Microsoft Multitasking MS-DOS Product Specification OVERVIEW*
- *Microsoft Multitasking MS-DOS Product Specification DEVICE DRIVERS*
- *Microsoft Multitasking MS-DOS Product Specification SYSTEM CALLS*
- *286 and 8086 Compatibility*
- *Microsoft Multitasking MS-DOS Product Specification INTRODUCTION*
- *Microsoft Multitasking MS-DOS Product Specification MEMORY MANAGEMENT*
- *Microsoft Multitasking MS-DOS Product Specification DYNAMIC LINKING*
- *Microsoft Multitasking MS-DOS Product Specification SESSION MANAGER*

Requirements

MS-DOS 4.0 will run all *well-behaved* programs that run under MS-DOS 2.0. "Well-behaved" means those programs that use the DOS calls exclusively to interact with the machine and the user, that make no assumptions as to where the program will be loaded or the amount of memory available, and that otherwise use no machine-specific information.

Many popular programs do not meet this definition of good behavior, so MS-DOS 4.0 contains a variety of features to support the most common "ill-behaved" program actions, thus allowing most existing MS-DOS 2.0 programs to run unchanged.

MS-DOS 4.0 operates on the same type of machines that presently run MS-DOS 2.0. To maximize performance for MS-DOS 4.0, these machines should provide interrupt-driven I/O and dynamic memory allocation capability.

1.2. 8086 Architecture

The 8086 architecture (and consequently the 286 "8086-mode") lacks two features useful in a multitasking environment: it has no protection and offers no hardware relocation capability. While the 286 offers such features, it does so only in a mode which is binary-incompatible with the 8086. Therefore, it is not possible to pass these new 286 features to old applications. Nevertheless, there is a large existing software base for MS-DOS 2.0, so it is unacceptable to require all applications to run in 286 virtual mode (which would necessitate recompilation at a minimum, and restructuring at a maximum) to solve these problems.

Typical MS-DOS machines have not added any hardware memory protection external to the 8086 chip, so they have no protection against "buggy" or ill-behaved programs damaging other programs or the DOS itself. Consequently, unless circuitry is added (which is easy to do), MS-DOS 4.0 has no provisions for protection on 8086 machines or 286 machines when running old applications in 8086 mode. MS-DOS 4.0 cannot restore or protect tasks from damage from 8086-mode programs.

1.3. Definitions

The following definitions are used in this specification:

Task

An independent program execution. Many tasks may exist simultaneously, and all have their own copy of the DOS per task data structures.

Process

Used interchangeably with *task*.

Concurrency

The ability to run multiple screen-oriented applications at once by giving each its own logical (or virtual) full-sized screen. A series of keystrokes allows the user to switch between different logical screens. Only one screen may be visible at a given time (the physical screen), although all may be running.

Device Driver

Hardware-specific routine that interfaces the DOS to the hardware and is called by the DOS. Each device driver has a strategy entry point to begin an operation and an interrupt entry point which is called when the hardware has completed the operation.

Interrupt

An external event which requires immediate servicing, temporarily disrupting the 8086 from its normal execution. In some cases, the interrupt service routine will be exempt from time slicing, but these routines typically are optimized to let the 8086 return to normal business as rapidly as possible.

Foreground Screen Group

That group of tasks (often only one task) associated with the physical screen as seen by the user.

Background Screen Group

Groups of tasks associated with logical screens other than the physical screen. See *concurrency*.

Keyboard Focus

That task, in a given screen group, to which keystrokes are to be passed. Usually, but not always, this is also the only task doing screen output.

Detached Task

Those tasks in a given screen group that are not the keyboard focus.

Blocked Task

A task that has made a request to the DOS or to a device directly, and cannot continue running until that request has been satisfied.

Runnable Task

Opposite of a blocked task. A runnable task can be run should the scheduler so choose.

Task Switch

The DOS suspends execution of one task and resumes execution of another. The DOS must switch internal data structures. This may occur many times a second.

Screen Switch

The DOS interchanges the foreground screen group on the physical screen with a background screen group. This happens only at the instruction of the user or to handle errors.

Exec

The ability of one task (parent) to invoke another (child). The parent will be suspended until completion of the child. All applications are child processes of the Command Interpreter which must run to completion before control is relinquished back to the Command Interpreter.

Spawn

Similar to *Exec*, except that the parent is able to continue running without being dependent (at least initially) on completion of the child.

Shell

A command interpreter that controls execution of a set of programs.

2. THE DOS INTERFACE

Most MS-DOS functions are accessible through **INT 21H**. The DOS calls are used to control every aspect of the machine. Programs should use the DOS calls only to interact with the user, the devices, or other programs.

The dynamic linking facility provides another powerful method for applications to call on support services. This will be discussed in Section 5.

The DOS is reentrant from the applications developer's standpoint because if he desires to develop a "background" task, he need not be concerned with restrictions imposed by other tasks. This task would no longer have dependencies on the DOS data structures. However, the DOS is still synchronous from a given task's standpoint, meaning that if a task is given control while it is still technically within the DOS (such as receiving a hardware interrupt directly), that task may not call the DOS again. Similarly, this restriction prevents device drivers from calling the DOS except at initialization time.

3. PROCESS CONTROL

3.1. New System Calls

Several new or changed system calls are available to start, terminate, and control the execution of tasks or processes under MS-DOS 4.0.

EXEC

Used to start up new tasks. Tasks may be synchronous or asynchronous. Also used to load overlays.

WAIT/CWAIT

Used to wait for the termination of a task by the parent or initiator of the task, as well as to retrieve the return code from the child.

EXIT

Used to terminate a task and return a completion code to the parent.

KILL

Used to terminate an external task. This is actually a special form of SENDSIG.

3.2. New Capabilities

A major capability of MS-DOS 4.0 is multitasking. MS-DOS 2.0 programs could create child processes, but the child had to finish execution before the parent could resume. Under MS-DOS 4.0, the parent can wait for the child or continue executing.

Child processes still inherit their parent's environment – open file descriptors, and so forth. This capability, together with features such as pipes, allows programs and tasks to run any other program or task. For example, a program that sorts a file need not include a sort package; it can invoke the system sort utility as a child process. This design is called a *software tools architecture.*

A key consideration of this architecture is that most operations that affect processes actually affect the specified process and its descendents. For example, if a running program *X* has invoked SORT as a child and if *X* is killed, then SORT needs to be killed also, although the task or shell that created *X* is ignorant of SORT. In fact, neither the shell nor *X* knows that SORT may have invoked a child process to help out. Each process, together with all its descendents, is called a *command subtree.*

The use of a software tools architecture is a powerful programming technique and greatly improves programmer productivity as well as increases the functionality and compatibility of resultant applications programs. It is not, however, of great interest to the average end user, a non-programmer. Software tools and multitasking are important to the OEM and the ISV, and indirectly benefit the end user.

4. THREADS

MS-DOS 4.0 (and a future release of XENIX) will support a concept called *multiple threads*. A *thread* is a stream of control executing a process. Currently, when a DOS or XENIX process starts, it has only one thread – the program is being executed only in one spot with one CPU. MS-DOS 4.0 uses a mechanism whereby additional threads of control can be created, and code in several locations can be executed "simultaneously."

A thread is *not* another process. If a process is running with two threads they are almost identical: if one thread issues an OPEN system call and gets back handle #4, the other thread can do a READ on handle #4 to read that data. The only thread-specific information maintained are the register contents.

Threads are intended to be "cheap" so that processes can rapidly create and evaporate them. For example, a screen-interactive program might obey a command and then create a thread to do the screen update while the original thread looks for more commands. If a new command arrives, the update thread could be terminated or the new command could be processed in parallel. Threads will sometimes be used for CPU-bound operations because they simplify the design of a program. More often they will be used when there is I/O involved; it allows one thread to continue computing while another is blocked on I/O.

The operating system interface to threads is simple. A system call is performed that makes a new thread and stores the location of the new thread's stack. Another system call is available to destroy new threads.

5. MEMORY MANAGEMENT

5.1. New Features

The Microsoft memory manager provides:

- Automatic sharing of pure code segments.
- Maximum flexibility to relocate on a segment basis.
- Swappable segments. Swapping segments that belong solely to idle tasks is not sufficient. Segment sharing and task behavior may force swapping of some of an active task's segments, but will continue running that task until it references a swapped segment.
- Dynamic linking. This provides programs with access to other software services that are not necessarily built into the operating system. Links to the libraries are made through names.
- Transparency and compatibility. Existing high-level language applications have as much transparency and compatibility as possible passed on to them automatically through recompilation with new compilers.

5.1.1. Movable Memory Segments

MS-DOS 4.0 will allow allocated memory segments to be specified as being movable or fixed in memory. Movable memory segments permit more effective memory utilization because free memory blocks can be coalesced more readily. If a program requests a fixed segment, it is given a segment number as with earlier versions of MS-DOS. If a program requests a movable segment, it is returned a handle. A subroutine is called to reference the segment. The subroutine converts the handle into a long pointer and the segment is temporarily made fixed. When the program has completed the specific access, it calls another subroutine to unlock the segment. Movable segments may be swapped out; in this case, the "lock down" subroutine causes the segment to be swapped in first.

Finally, segments may be flagged "discardable;" if the system runs short of free memory, such segments will be discarded. In such cases a special code is returned from the "lock down" subroutine to indicate that the segment was discarded.

5.1.2. Shareable Code Segments

New programs will be able to share multiple instances of their pure code segments. This is transparent to the program; each instance of the program will have an independent data segment that the system will maintain for the benefit of the program. All pure code segments are automatically shared; they need not be movable.

5.1.3. Dynamic Linking

New programs will also be able to dynamically link to entry points in other segments. This will lead to additional sharing of code. For example, the run-time libraries of higher-level languages may be implemented as separate shared segments. This will reduce the size of executable files and reduce total memory usage. As an added benefit, programs will not have to be relinked to take advantage of revised versions of libraries. It will also make a richer environment available to the programmer. The built-in commands of COMMAND.COM will be available as subroutines.

5.1.4. Demand Loading

As an option, the segments that make up a program need not be loaded when the program is loaded. Instead, they may be loaded only when they are referenced. This is intended for infrequently used subroutines like error handlers. The set of routines that are pre-loaded may be specified by each program or library. The "on demand" loading feature may also be turned off in situations where it is inappropriate, such as floppy disk machines that require a lot of disk switching.

5.1.5. Swapping

To meet extraordinary memory demands, MS-DOS 4.0 may swap segments to disk. When a segment is swapped out, programs that use the segment will be permitted to run until they need to refer to the swapped-out segment. They will then trap to the system that will arrange for the segment to be reloaded. This virtual segment capability is transparent to the program.

5.2. New System Calls

See the product specification entitled *Microsoft Multitasking MS-DOS Product Specification MEMORY MANAGEMENT* for a detailed list.

5.3. Programming Considerations

Certain constraints must be placed upon user programs to allow these new features to operate. Suggested programming guidelines for new programs are:

1) Use small or medium memory models. The memory management features are designed to be most useful with small or medium (small data, large code) memory models. Long pointers should be avoided where possible.

2) Use the pre-defined subroutine linkage conventions. The memory management system assumes that certain subroutine linkage conventions are followed so that it may determine the state of the stack frame.

3) BP must always point to a valid stack frame. For the same reason as in the previous point, the value stored in BP is constrained.

4) Code segments must be pure to be shared. Storing into variables using CS overrides is not permitted.

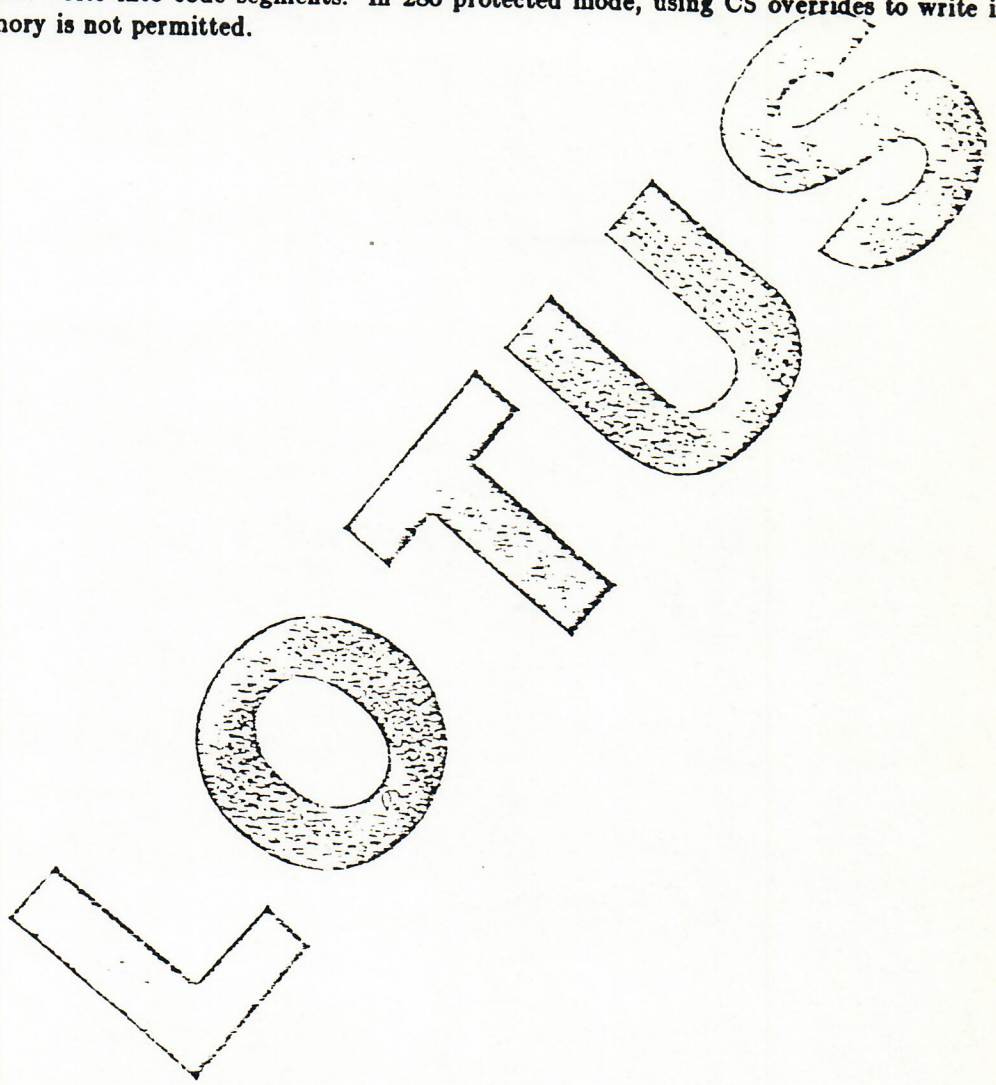
5) Far pointers must be dereferenced via indirect pointers. When a program would normally use a pointer, it must use a "memory handle." The memory management system must be called to turn the handle into an address and lock the segment against movement and swapping. When the program is done using the memory, it must call the system to unlock the memory.

The following programming conventions will allow for greater portability to the 286:

1) Avoid using segment registers as scratch registers. On the 286, segment registers must contain valid segment selectors.

2) Avoid segment arithmetic. In 286 protected mode, segment selectors may not necessarily be created by computation using shifts and adds.

3) Do not write into code segments. In 286 protected mode, using CS overrides to write into memory is not permitted.



6. CONCURRENCY SUPPORT

6.1. New System Calls

- FREEZE
  Used to stop the execution of a program for a time, at which point it can be restarted.
- RESUME
  Used to restart a frozen task.

6.2. New Capabilities

MS-DOS 4.0's multitasking capability is important to both the OEM and the ISV, and its concurrency features are key to the end user. "Concurrency" refers to the ability to perform any one of several things at one time, together with the ability to switch back and forth between them at will. A human being is a prime example of a concurrent processor. He can do only one thing at one time, but he can switch back and forth between several ongoing tasks.

MS-DOS 4.0 includes a utility named "Session Manager" that manages several threads of activity. Session Manager gives the user the illusion of having several independent processors. For example, the user might be running a copy of Multiplan when the phone rings. The user could activate Session Manager by striking a special key, then use Session Manager's menu interface to create a new *screen group*. This screen group appears to be a whole new PC. The screen now would display a COMMAND.COM prompt (or perhaps a Visual Shell menu). The user would now enter his calendar program.

When done reviewing his calendar, he could toggle back to his Multiplan screen with a couple of keystrokes. The screen contents and mode would be restored just as they were when Multiplan was last running. He would not need to exit the calendar program first; he could leave it running and switch away from it just as he had earlier switched away from Multiplan. More specifically:

- There can be as many different "virtual machines" or "screen groups" as memory allows, although these may be swapped to disk when inactive (as in the previous example when the tasks are suspended for keyboard input) on systems where this would be realistic (hard disk or network connection). Session Manager might enforce an upper limit of 16.
- This facility does not require cooperation on the part of the application. Control is taken from the application; when it returns the application cannot detect any change as the screen is fully restored.

Many older programs perform tasks that are hostile to the multitasking environment. For example, some write directly into physical screen memory, without going through the DOS (including the ANSI driver) or firmware. Although MS-DOS 4.0 is capable of continuing to run applications when their "screens" are not being displayed on the console, such ill-behaved programs must be FROZEN so that they will not write to screen memory when some other program is using it. In general, when Session Manager switches the physical screen to another image, there are three possibilities for programs that were using the old screen image:

1) FREEZE the program. RESUME it only when its screen is the active one. This is the default action and must be taken for older, ill-behaved programs.
2) Let the program continue to run, but suspend it when it tries to do any console output. This mode prevents the scrolling of information that will never be seen by the user.
3) Let the program continue to run, even if it writes to the screen (through a software interface).

Session Manager decides which of these alternatives is appropriate for each program. Session Manager determines program behavior via a series of defined *behavior bits*. These are discussed in more detail in Section 12.4.

7. SCHEDULER

7.1. New System Calls

- SLEEP
  Used to delay the execution of a task for a specified amount of time.
- PRIORITY
  Gets/sets a task's priority.

7.2. New Features

The two overriding principles applied to the MS-DOS 4.0 scheduling algorithm were to keep it simple and to minimize perceived performance degradation (from the standpoint that an application no longer receives 100% of the CPU resource). Since there aren't multiple "users" who will notice inconsistencies in the frequency and duration of time slices, it is possible to give the foreground screen group a disproportionately high percentage of the CPU resource to minimize the user's belief that other tasks are interfering with his primary task.

The scheduler essentially runs the highest priority task in the run queue. A task leaves the run queue by being blocked on I/O or voluntarily going to sleep. When a task leaves the run queue (which can only happen when it is actively running or through user intervention), the highest priority task (that is runnable) is invoked. An exception to this is when a scan of all runnable tasks is made at a certain interval, currently three seconds. Any task that has not run during that interval will then be run for one slice, to the exclusion of higher priority tasks (except when the active task has declared itself time-critical). This technique will help to minimize lockout in this otherwise simple scheduling environment.

The user may be thought of as being able to select from a limited set of priorities for any given task. A default constant will be assigned to all tasks for which no specification is given. The scheduler then adjusts for each of the following events:

- Member of the foreground screen group.
- I/O completion such as a disk read request or keyboard input.
- If a foreground screen task has done screen output (character or graphics) through a detectable software interface (so as to minimize the jerkiness associated with time slicing during screen output).

The priorities 0 and 255 have special meaning. Priority 0 is only assigned voluntarily by a task to itself. This means to run the task only when no other is available, excluding it from the three-second scan. A print spooler might run at this priority. Priority 255 is reserved for time-critical tasks that will run to the exclusion of all others, disabling the three-second scan. It is anticipated that such tasks will run for very short periods of time. For example, a downloader running in the background might run at this priority so that as it is receiving data at 9600 baud. When its sector has filled, it can be dumped to the disk before its ring buffer overflows. It will continue to be interrupted as characters come in while this DOS request is taking place. Other programs that might run at this priority include a network server, real time process control, Session Manager, and BASIC (while it is playing music).

The following examples illustrate a typical multitasking environment:

1) Two tasks with different priorities. The higher priority task will always run, with the lower priority task only running during I/O operations of the other. If there is any three second period where the higher priority task was not blocked for I/O, the lower priority task will get a slice.

2) Two tasks of equal priority, one disk-intensive and one screen-intensive. This disk-intensive application often will not use up its slice, but instead will "block" on I/O. Its priority will be boosted upon completion. The screen-oriented task will also have its priority boosted, but not as much as the disk-intensive task, after every slice during which it did screen I/O that the DOS could detect. Consequently, if the disk-intensive application does not

complete its slice before being blocked, it will have higher priority and will run whenever ready. (The screen-oriented application will still get most of the machine because the I/O will take longer than a slice to complete.) If, however, the disk-intensive application uses up most of its slices (in which case it is more CPU-intensive than disk-intensive), it will run once every three seconds.

The scheduler manages the execution of all processes. It decides which tasks run, and for how long. In general, a process runs until it has exhausted its time slice, or it must block on some I/O operation, or it voluntarily surrenders the CPU.

If a task has to wait for some event, the scheduler takes the task off the run queue until that event happens, at which time the waiting tasks are placed back in the run queue. Tasks are automatically scheduled to suspend and resume during I/O operations; tasks can also request a voluntary suspension for a specified interval of time.

No polling for events should be done. If some task is waiting for a particular event, it is not placed in the run queue until the event happens. This reduces the overhead of switching tasks by avoiding checks to see if any of the waiting tasks are runnable.

A mechanism exists to specify time-critical tasks. Such a tasks is immediately run at the earliest opportunity. It runs until it relinquishes control or goes to an I/O wait. In this way, a bounded response time that depends only on scheduler clock resolution (a tick) is provided for tasks that have to respond to real time events. If more than one time-critical task becomes ready, then the second will not run until the first completes. The only thing that may interfere with a time-critical task is an interrupt service routine. Note that this is not provided to service hardware interrupts (these are handled by the device drivers), but to complement the device drivers and allow task level processing to occur quickly in response to a critical event.

When a task issues an I/O request, the request is queued, the task is taken out of the run queue, other tasks are run, the I/O completes (the device driver notifies the scheduler), and the task is placed back in the run queue.

Several applications have timing-dependent copy protection schemes using IBM **INT 13H**. In this case, that task's priority will be increased to 255 for several seconds so that time slicing will not confuse the copy protection scheme.

8. INTERPROCESS COMMUNICATION

8.1. New System Calls

- PIPE
  Used to create named or anonymous pipes.
- CREATMEM
  Used to create a shared memory area.
- GETMEM
  Used to get access to an existing shared memory area.
- RELEASEMEM
  Used to release access to a shared memory area.

8.2. New Features

MS-DOS 4.0 provides two forms of Interprocess Communication (IPC): *pipes* and *shared memory*.

8.2.1. Pipes

A *pipe* is a serial communications channel connecting two or more processes. Just like its physical namesake, the writer tasks put data into the pipe and the reader tasks remove it. Pipe I/O is done via the standard system **Read/Write** calls; pipe I/O is indistinguishable from normal file I/O. This is deliberate. It allows a task to communicate interchangeably with consoles, disk files, and other tasks, including those over the network.

There are two subclasses of pipes, *named pipes* and *anonymous pipes*. The distinction is in how the pipes are established; once opened they are identical.

Anonymous pipes are created via the **PIPE** system call. The **PIPE** call returns both ends of a pipe. No other task can also open that pipe since it has no name, but any child tasks that the current task creates will inherit whichever end of the pipe the parent chooses.

This allows a task to run a child task and read/write data from or to the child. The parent task typically arranges for the pipe channels to be the STDIN and STDOUT channels so that the child task is unaware that it is being invoked by some other task. In this manner *any* program (such as SORT, for example) can be invoked to do work for any other program.

Named pipes are created via the **Create** and **Open** system calls, and have pathnames associated with them. All named pipe pathnames must start with the string:

\PIPE\

This restriction is made to ensure compatibility with future enhancements to pipes. Since these pipes have names, they can be explicitly opened by other unrelated processes. Named pipes are typically employed by service tasks and their clients: the service task opens a pipe of a known name and issues a read. Client tasks open the same name and write their requests on the pipe. Arbitrarily sophisticated architectures can be created: for example, the server task takes the client "off channel" by specifying a new pipe name. The client and the server (or a child task created by the server) could then communicate privately over the new pipe.

Either type of pipe exists only while some process (or processes) have it open. The pipe is deleted when all processes close their handles to the pipe, and the region of memory occupied by the pipe is deallocated. Pipes are not buffered on disk.

8.2.2. Shared memory

MS-DOS 4.0 supports a shared memory facility. The **CREATMEM** system call allocates a segment of memory of the requested length and records its name. Like named pipes, above, shared memory names use the file system name space but are currently restricted to names that begin with:

\SHAREMEM\

This restriction is made to ensure compatibility with future enhancements to shared memory. Once created, an area of shared memory can be accessed by other processes via the **GETMEM** system call. Shared memory is typically employed to allow two or more closely cooperating processes to share access to common tables and buffers. Pipes and named pipes should be used if the processes only wish to transfer stream data; on some systems pipes may be more efficient than shared memory.

9. SIGNALS

9.1. New System Calls

- SETSIGNAL
  Controls the task's sensitivity to a SIGNAL. Specifies an address to call when the signal is delivered.
- SENDSIGNAL
  Sends a signal to another task or an entire command subtree.

9.2. New Features

The signal mechanism allows a program to receive asynchronous events — a kind of software interrupt. Programs use the **SIGNAL** system call to specify a service address for the signal(s) in which they are interested.

When a signal arrives at a process, the system examines the process signal table. If the process has specified an action (either accept or ignore), the system does as instructed. If no explicit action has been specified or inherited from the parent, the default action is taken.

The following signals and default actions have been defined:

SIGNAL Description		
SIGNAL	*Function*	*Default Action*
SIGDIVZ	Generated when program divides by 0.	Abort Program
SIGFPE	Generated by floating point exception.	Abort Program
SIGTERM	Sent when the process is to terminate immediately.	Abort Program
SIGABT	Sent when the process is to terminate immediately. This signal cannot be disabled or intercepted.	Always Aborts Program
SIGINTR	Sent when the special *INT* key is typed at the console. This is normally CTRL-C.	Abort Program
SIGKB1	Sent when the special *KB1* key is typed at the console.	Ignored
SIGKB2	Sent when the special *KB2* key is typed at the console.	Ignored
SIGUSER1	Sent by some other process. Used for interprocess communication.	Ignored
SIGUSER2	Sent by some other process. Used for interprocess communication.	Ignored

Signals fall into one of three categories:

1) Program Faults
   These signals, **SIGDIVZ** and **SIGFPE**, represent program errors.
2) Keyboard Events
   These signals, **SIGINTR**, **SIGKB1**, and **SIGKB2** represent asynchronous keyboard events.
3) IPC Events
   These signals, **SIGTERM**, **SIGINTR**, **SIGUSER1** and **SIGUSER2** are Interprocess Communication signals and are sent via **SENDSIG** from some other process.

This document groups the three classes of signals together because they are treated similarly by the DOS. These classes will be treated separately in the *MS-DOS Programmer's Reference Manual* since the origins and uses of the three kinds of signals differ considerably.

10. KEYBOARD INTERCEPTS

A *keyboard intercept* is a mechanism whereby a program can be asynchronously notified of a keyboard event, such as a CTRL-C. Keyboard intercepts are handled somewhat differently in MS-DOS 4.0 than they were in MS-DOS 2.0.

MS-DOS 2.0 defined only one keyboard intercept: CTRL-C. The device driver treated CTRL-C as a normal character; if a non-raw **READ** attempted to read a CTRL-C character from the queue, the DOS would call the **INT 23H** vector instead. Programs that hung up in loops or that did not read from the console would not respond to a CTRL-C character.

Under MS-DOS 4.0, the keyboard device driver presents every incoming character to the DOS. The DOS maintains a flag indicating the status of the keyboard device: *cooked* or *raw*. If the device is in raw mode, all characters are treated as data. If the device is in cooked mode, characters that match CHAR_INT, CHAR_KB1 or CHAR_KB2 (each is settable via **IOCTL**) generate **SIGINTR**, **SIGKB1** and **SIGKB2**, respectively.

In normal operation, the current command interpreter (often called a "shell," usually COMMAND.COM) has enabled the **SIGINTR** signal. If the user types CTRL-C, the shell receives the **SIGINTR** signal and issues (via a system call) a **SIGTERM** to the currently running process. Actually, it issues a **SIGTERM** to the entire command subtree of the currently running process, in case it also has created child processes.

Processes can circumvent this normal sequence in three ways. First, they may not wish to be killed unexpectedly, perhaps because they need to clean up some locked files, restore a database, or whatever. In such a case the program intercepts the **SIGTERM** signal; when the signal is received, the service code would do the cleanup and then exit the process. Note that the command shell will not reprompt the user until all the processes in the command subtree have terminated.

The second way that programs can affect **SIGINTR** (i.e., CTRL-C) processing is to set the keyboard device into raw mode. This is done by programs, like screen editors, that treat a CTRL-C as a data character. The program would restore the keyboard to cooked mode before it exits.

The third possibility is that the running program is itself a shell. Examples of this might be editors, Multiplan, etc. These programs do want to be alerted to a CTRL-C (or whatever character they prefer) but don't want to be killed because of it. They would issue the **SETSIGNAL** system call to intercept **SIGINTR**; this automatically suspends the parent shell's sensitivity to **SIGINTR**, **SIGKB1** and **SIGKB2**. When this new shell exits to its parent, the parent is automatically resensitized to those signals, as appropriate.

Note that existing programs believe that issuing a special "raw-mode" **READ** (subfunction 6) protects them from being killed by a CTRL-C. This will not be the case in MS-DOS 4.0; a special behavior bit is set so that running such programs automatically puts the console in raw mode.

11. FILE SYSTEM

11.1. New System Calls

- LOCK
  Used to lock a range in a file. This prevents other tasks from reading or writing that range until it is unlocked. A task may lock multiple segments within one or more files.
- UNLOCK
  Releases a file LOCK.
- OPEN
  Additional arguments to allow programs to restrict *any* read and/or write access to the file by other tasks.

11.2. New Features

Since multiple processes may contend for files, MS-DOS 4.0 provides for controlling access to entire files and/or selected fields within files. These access rights can be used to control concurrent access to a file or totally exclude access to a file. Provisions are made for locking ranges in a file.

Further, the *Mode* field of the Open system call has been extended to allow the program to restrict any further read and/or write access to the file. This differs from the range locking described above because it affects the whole file and remains in effect until the file is closed.

Programs presenting an unextended ("old style") mode to Open will open the file in *compatibility mode*. This mode allows one task to open the file as many times as it wishes, in any MS-DOS 2.0 mode. All other tasks are refused access to a file open in compatibility mode.

These extensions to the file system calls are exactly the same as are found in MS-DOS 3.0.

11.3. Networking

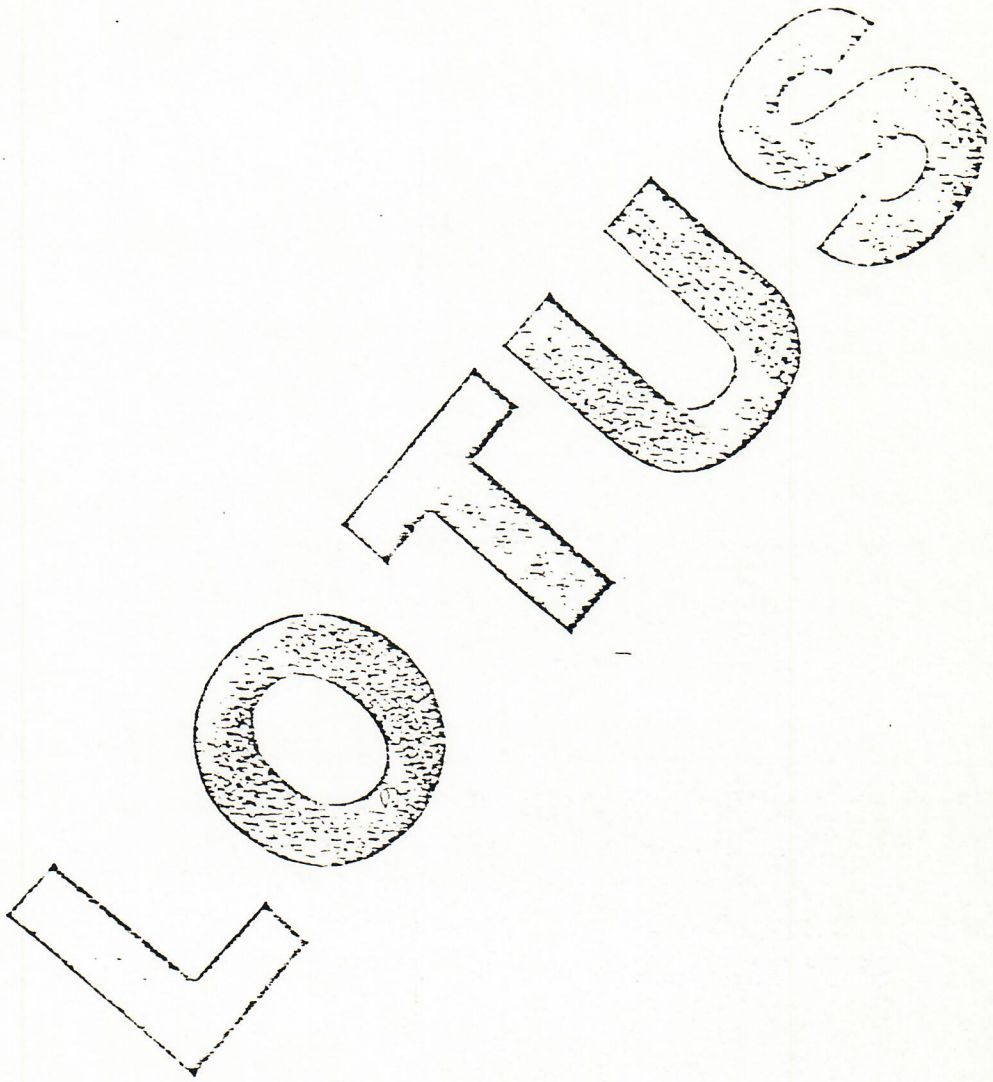
Networking will be supported under MS-DOS 4.0 in much the same manner as it is under MS-DOS 3.0, except that it will take full advantage of the multitasking capabilities. Specifically, the server will become just another MS-DOS application. The redirector will function identically to its current state, although since it is tightly coupled with the specific MS-DOS version, a new version will be required. The existing transport layer device driver should work, although the ability to use DMA will become much more important in the multitasking environment.

11.4. Omitted Features

Microsoft recognizes the need to extend the file system in a number of different areas. This has been deliberately omitted from this release and is slated for a future release. Specifically these capabilities include:

- File system protection and permissions.
- Increased performance on large disks. The current file system was designed and optimized for floppy disk devices. Running such a file system on a 32 Meg hard disk is stretching it beyond anything that was imagined when it was designed. Significant performance gains are possible with a file system optimized for large non-removable devices.
- Installable file system. Similar to the needs of the large hard disk being different from those of the floppy, there are other storage devices, such as laser disks, that might require a customized file system to maximize performance. This would allow a specific file system to be installed for specific devices, much as a device driver is today. The new hard disk file system mentioned above would be written to this installable interface.
- Networking. The network code would presumably be rewritten to the same installable file system interface.

- Symbolic Links. Makes files over the network appear in the local name space. The file contains the name of the synonym file.
- Undelete. Currently the FAT entries and the first byte of the directory entry are destroyed on file delete. These could instead be saved and dynamically expunged by the file system when the disk filled up. This would allow the user to undelete his files so long as they have not been written over because the disk is full.
- More information in the directory entry. Longer names, lowercase names, access and creation date/time, name of the application that created the file, revision number, etc.



12. COMPATIBILITY ISSUES

12.1. Ill-Behaved Programs

Many existing application programs assume they are the sole program on the machine. Programs that write directly into screen memory or play with the interrupt vectors, for example, fall into this class. These programs are called *ill-behaved*—they will not run, as is, in a multitasking environment.

Although they may be ill-behaved, they are also quite popular. MS-DOS 4.0 contains facilities that provide special support for these programs to allow them to run unchanged. Not all kinds of ill-behavior can be supported, though. The following sections describe "recoverable" ill-behavior, the recovery techniques, and irrecoverable ill-behavior.

12.2. Recoverable Ill-Behavior

- Direct Screen Manipulation. Many programs write directly to the screen memory, circumventing the DOS. These programs have the appropriate behavior bit set (see Section 12.4) so that they are frozen whenever their screen is not the active one.
- System Peripheral Manipulation. Some programs manipulate system peripherals by means of the ROM firmware or by intercepting the interrupt vectors. MS-DOS 4.0 notices their manipulation of the system interrupt vectors and remembers the task's new vectors. Interrupts to those vectors while the task is active are sent to the user task.
- Interrupt Vector Manipulation. Some programs adjust the interrupt vectors to control private hardware or as a form of cross-module calling. The DOS records these vectors on a per-process basis so that vectors set up to allow interrupt calling within a task will work, but vectors set up to allow interrupt calling to some other task (a terminate-and-stay-resident routine, for example) will fail.
- **INT 24H**. Many programs intercept the **INT 24H** (hardware error) interrupt so that they can control and restore their screen format. MS-DOS 4.0 does *not* support user interception of **INT 24H**; instead, Session Manager will receive the **INT 24H**, save the screen, put up a display, get instructions from the user, and restore the original screen.

  Programs whose screens might be damaged will run normally under MS-DOS 4.0; programs that "get fancy" will run if the user successfully *Retries* **INT 24H** errors, but may be incorrect for the *Abort* or *Ignore* options, because the application may not be expecting these responses if it had previously disabled them.
- Polling. Some applications poll the keyboard through the ROM (which is detectable by the DOS) and others poll I/O ports directly (which is not detectable). Whenever the DOS detects polling, it will relinquish the remainder of that task's time slice. The task will be placed back in the run queue.

12.3. Non-Recoverable Ill-Behavior

- System Peripheral Manipulation. Depending upon the degree and kind of manipulation, these programs may not work in a multitasking environment. For example, programs that reprogram the video controller without going through a software interface must run to completion before a screen switch can take place, since the screen device driver will not know what state the screen is in.
- Interrupt Vector Use. As discussed above, programs that use the INT instructions to communicate with private "terminate and stay resident" programs will fail.
- **INT 24H**. Programs will no longer receive **INT 24H** calls. Programs that require these calls to do special error processing may fail.

- Memory Utilization. Programs that size memory themselves or otherwise ignore DOS memory management in favor of self-help will fail.
- Files and I/O. Programs that write scratch files with fixed names will fail if a second copy is run in the same directory.
- CTRL-C handling. Programs that make assumptions about the register contents and stack format when the CTRL-C vector is taken will fail.
- Multiple invocations of certain existing applications may fail. This is due to some applications storing values in fixed locations or trying to open for output a file with a fixed name.
- Keyboard. Any application that steals the hardware keyboard interrupt (e.g., IBM INT 9H) and replaces this service routine without later chaining to the routine it replaced will disable screen switching.
- Timer. The scheduler requires a timer interrupt (e.g., IBM INT 8H or INT1CH). Running programs that trap these and fail to chain can be fatal.

12.4. Behavior Bits

The binary .EXE file header defines a series of *behavior bits* that define the behavior of the program. The DOS and Session Manager examine these values to provide the right amount of compatibility support without wasting effort on well-behaved programs. These bits include:

- Program writes directly to screen memory.
- Program affects interrupt vectors.
- Program intercepts the keyboard vector.
- Program expects old "read raw keyboard" semantics.

A utility will be distributed that recognizes the most common MS-DOS applications. During installation, the user can use this utility to set the correct behavior bits on his existing applications. The default value for the bits (and for .EXE files that have no bits) is ill-behaved, so use of the utility is optional. New programs would have their correct behavior bits set by the publishers before distribution.

In general, maximally ill-behaved programs (no behavior bits set), will run. They will be frozen in the background and their task switch overhead will be larger than well-behaved programs, because the DOS must scan interrupt vectors. Setting behavior bits can only improve performance for partially ill-behaved programs.

12.5. Memory Utilization

Old programs cannot be made movable and/or swappable since they do not conform to the new memory management guidelines. Non-movable segments (including all old applications) will be allocated contiguously to provide as much contiguous memory as possible for movable segments. It becomes increasingly important for any old "terminate and stay resident" programs to be locked at initialization time to minimize memory fragmentation.

13. INTERNALS

From the standpoint of most of the MS-DOS 4.0 kernel, there is only one major concern. Each task consists of a Per Task Data Area (PTDA) and a program segment. This design is called *multiple logical kernel activations*. This technique (which is also used by XENIX) significantly decreases the complexity of the kernel and helps to maintains compatibility with previous versions. For integrity and simplicity, the entire MS-DOS kernel is considered a critical section of code that may not be interrupted. This by itself would make for a non-optimal environment if only one task at a time could be active within the kernel. By exiting the critical section down inside the device driver (which requires that all DOS data structures be appropriately updated prior to calling the device driver), multiple tasks can be active within the kernel at once if all but one are actually in the device drivers. The majority of any I/O operation is spent in the device drivers, and the time from entering the kernel to reaching the device driver is relatively small, so this restriction is a small price to pay for the simplicity of design.

The program segment has the same format as any program under MS-DOS 2.0 (100H header plus actual program). The PTDA is the data area for the DOS and contains all task specific information. This area is not available to the task, and may be only manipulated by the DOS. Each task has a separate PTDA.

Communication between a task and MS-DOS is still done via **INT 21H**. Interrupt vectors are now managed by the DOS through the **SetVector** and **GetVector** calls. A task is not allowed to place vectors directly in the table because several tasks may want to set vectors for the same interrupt. To prevent such conflicts, the DOS intercepts all interrupts, dispatching to the appropriate foreground process.

MS-DOS 4.0 provides limited facilities for polling. It is designed to run in an interrupt-driven environment. Polling is accomplished by tying in to the scheduler clock through the **TimerService** call. In this way, checking can be done for a non-interrupt-driven event at every scheduler tick. This facility is also available for device drivers; its use introduces overhead but allows machines to use polling, without penalizing those other machines that do not require it.

Support for a 8087 coprocessor will be given if the processor is installed in the machine at boot time. This will guarantee that the state of the 8087 will be saved on task switches. A facility to mark programs not using the 8087 will be provided to minimize overhead.

14. HARDWARE CONSIDERATIONS

The minimal system for supporting MS-DOS 4.0 is an 8086/88 machine (i.e., processor, memory, video I/O, etc.) with a timer. The minimal realistic system should also have interrupt-driven I/O and DMA capability for the disk.

MS-DOS 4.0 is designed to fully support concurrent I/O. Concurrent I/O refers to the ability to be working on several different I/O requests simultaneously. Although each process can have only one outstanding I/O request at a time, several tasks may each have a pending request.

Concurrent I/O is especially beneficial for disk devices. They can sort their pending requests based upon their position on the disk, thus reducing seek time and increasing total I/O throughput.

To take maximum advantage of this capability, the hardware should allow for concurrent I/O to all its devices. This includes having a rational interrupt scheme, separate DMA channels for each device, concurrent disk seeks, and so forth.

All hardware interrupts are handled by the appropriate device driver. It is the responsibility of that device driver to inform the DOS at interrupt time when an I/O operation has completed.

15. DEVICE DRIVERS

15.1. Multitasking Device Drivers

Although most existing MS-DOS 2.0 device drivers will run under MS-DOS 4.0, they will slow system performance. For best results, all device drivers should be rewritten to use the features provided in MS-DOS 4.0. These features include:

- Blocked task. MS-DOS 4.0 will remove the task from the run queue when it is waiting for an I/O operation to complete. Examples would be waiting for keyboard input and waiting for a DMA disk operation to complete.
- Overlapped I/O. When this type of disk operation is taking place, other runnable tasks will be started so that the CPU may be kept busy. Since multiple tasks may be active within the file system at once, these other tasks may even make disk requests.
- Queued requests. Since multiple tasks may be active within the file system, the device driver may have multiple requests queued up. The device driver is free to reorder these in the most efficient manner.
- Write behind. The device driver may also queue up write requests, letting the requesting task continue processing. The only type of error that can occur under such a circumstance is a device (INT 24H) error which may be handled asynchronously with respect to the request.

15.2. Console Driver

The screen and keyboard portions of the console device driver will be split into two drivers; as these drivers get more complex, it becomes more difficult to replace one of the screen or keyboard independently of the other.

15.3. Asynchronous Driver

The current polled asynchronous communications interface on the IBM PC is inadequate for a multitasking environment. Specifically, asynchronous interrupts have the potential to come in at such a rapid rate, that it is impossible to relinquish the remainder of the time slice upon detection of polling as will be done with the keyboard, and it may be impossible to time slice at all without causing the application to lose characters.

The new driver will operate in an interrupt-driven mode. It will pass characters to a ring buffer established by the calling application. All communications applications, including BASIC, should be rewritten to use this interface. A compatible driver could be written for MS-DOS 2.0 so that these new applications would continue to run on older versions of the DOS.

15.4. Large Disks

MS-DOS 3.0 was extended to include 16-bit FAT entries which helped to reduce wasted space on large disks. Using 16-bit sector numbers limits expansion beyond 32 megabytes without increasing the sector size. 32-bit sector numbers will be added to facilitate larger hard disks. See section 11.4, "Omitted Features," in this specification.

15.5. Firmware

Many aspects of the IBM PC firmware (ROM BIOS) are inadequate for a multitasking environment. The most glaring deficiency lies in the lack of a jump out of the ROM disk code while DMA is taking place (label WAIT_INT). Another problem is the non-reentrant nature of this disk code which precludes queuing of multiple requests at one time over different DMA channels (a floppy and hard disk request going on at the same time for instance).

Other problems include the ROM loop for control NumLock (label K40) and video firmware's assumption that the logical screen is in fact the physical screen (label M3). Other problems such as the **INT 16H** ROM loop (label K1) are easily circumventable at the device driver level.

Some of these problems were rectified on the PC AT through the **INT 15H** Wait and Post capability.

16. UTILITIES

16.1. COMMAND.COM

COMMAND.COM will be made sharable and swappable. In addition, many of its commands such as COPY will be made accessible to applications through the dynamic linking facility.

Both functions of the resident portion of COMMAND are subsumed by other parts of the operating system and hence the resident portion is no longer needed. **INT 24H** errors are handled by Session Manager and the reloading of the transient code is handled by the DOS swapping facility.

Batch processing will be enhanced to allow recursive batch processing and redirection of I/O on an entire batch file.

16.2. Session Manager

The Session Manager manages multiple processes running on different screens. It uses initialization file SM.INI. This program includes the hard error catcher, therefore, HE_DAEM will not run with the Session Manager.

Normally Session Manager will invoke a copy of the command interpreter for each screen group. It could, however, invoke applications directly if so instructed.

See the specification *Microsoft Multitasking MS-DOS Product Specification SESSION MANAGER* for details.

16.3. Other Utilities

KILL.EXE

Sends signal *nn* with disposition *mm* to processes mentioned in PID. Defaults to SIGTERM and process tree disposition.

HE_DAEM.EXE

Intercepts hard errors, reports them and requests user action. This should always be run in the background for now. The above command line should be placed in AUTOEXEC.BAT if SM is not used.

BBSET.EXE

BBSET sets or clears behavior bits in an .EXE file header. MS-DOS uses the behavior bits to determine the level of special compatibility support needed to run the application.

EXEFIX.EXE

EXEFIX writes into the min-BSS and max-BSS fields of the executable header. These fields are given in hex paragraphs and control the initial memory allocation of the program. To avoid excessive use of memory, all executable programs should be run through EXEFIX. By default, C programs will release memory allocated to them in excess of 64K and use the remainder for their stack and allocatable (via **malloc**) memory. The program will also convert .COM files to .EXE files. This will allow them to be BBSET and have their memory allocation adjusted.

PRINT.EXE

The print spooler will be redone to be just another application. It will relinquish its scheduling to the DOS and applications.