CHAPTER 3: VIRTUAL MACHINE SUPPORT

The Virtual Machine Concept

A significant number of modern day PC applications were designed with the perspective of total hardware ownership. While this approach can enhance a program's operation in a stand alone environment it is seldom very compatible with a multitasking environment. Unlike a single tasking operating system, which merely provides services to the currently running application, a multitasking operating system must take more of a supervisory role to provide isolation and access control between each task. Controlling the flow of data to and from devices such as video displays, keyboards, disk drives and serial and parallel ports is vitally important to maintain reliability and prevent data corruption.

The PC-MOS operating system maintains a virtual machine environment, supporting multiple PC-DOS style tasks (as well as 386 native mode tasks). PC-MOS is not a generic multitasking kernel that provides a virtual PC in which you can 'boot' any operating system. Each task shares the MOS kernel and the services it provides. For the most part, this sharing of the kernel and the associated memory management operations are transparent to the application programs running in each task.

In designing and setting up a MOS system it is important to understand how this virtual machine environment is maintained; how the devices and resources within one physical machine can be shared in a coordinated manner. PC-MOS can run on different types of CPU's (e.g. 80386, 80286, 8088, 8086) and use different memory management hardware and software drivers. These aspects of a computer system will affect how well this virtual machine approach can be maintained.

No machine, virtual or actual, would be worth anything without task memory within which to run applications. As explained in Chapters 1 and 2, through the support of a paging-capable memory manager, multiple tasks can be supported which all share the same 640K base system memory area. This means that each task can have a "full size" TPA rather than suffer the minimal TPA sizes which result from having to split the physical 640K system memory between all tasks. Therefore, a basic prerequisite in the maintenance of multiple tasks (multiple virtual machines) of any practical size, is to have paging-capable memory management support.

In addition to providing the capability to map extended memory into the base memory area for each task's TPA, and allowing the use of FREEMEM and unused video memory (VTYPE), a memory management driver also provides support for "misbehaved" applications. For example, programs which write directly to the video display memory can be brought under control by using remapping to trick them into writing into a special video buffer when they "think" they're writing to the actual display adapter's memory. In this way, the video output of unwatched background tasks or tasks being watched by serial terminal workstations can be captured and channeled in an appropriate manner.

Another memory management capability known as I/O trapping also plays an important role in taming the activities of ill behaved applications. The term "I/O" stands for Input/Output - referring to the CPU's interaction with its peripheral devices such as video display control registers, communications ports, keyboard controllers, etc.

I/O trapping makes it possible for the processor to selectively divert attempts by applications to directly access the hardware, when this could be disruptive to a multitasking environment.

The I/O trapping feature is only available in 80386- based systems where the $386.SYS memory management driver is being used. Systems using an 80286 or 8088 type of CPU cannot provide this level of control. Were this isolation measure not available, an ill behaved application running at a serial workstation could affect the operation of another task or the entire system.

When an application attempts an operation that is protected through I/O trapping, PC-MOS interdicts and provides a simulation or buffering of the I/O event instead of allowing direct access to the device involved. This "fools" the application into believing that it is interacting with the actual I/O device and prevents system corruption.

Actually, the trapping feature of the 80386 hardware does not, of itself, provide complete control. A portion of the operating system software must manage the trapping, deciding which I/O accesses to act upon and what action to take in each case. The $$MOS.SYS kernel and the $386.SYS memory management driver work together to support this operation.

I/O trapping is not the only type of exception processing that an 80386 chip supports. Invalid conditions such as trying to execute undefined instruction codes and generating improper addresses are other exceptions that will cause the processor to interrupt itself and branch to an exception handling routine. This allows the operating system to take immediate action to terminate the current process when an application goes astray.

Another aspect of a computer system which MOS must maintain control over is the interrupt vector table. This is necessary in order to insure system stability given the behavior of certain applications and TSR utilities which manipulate interrupt vectors. The optimum control method is known as LIDT control and is available on 80386 and 80286 machines where a memory management driver is utilized. A somewhat less effective method known as REGRAB control must be used on 8088 and 8086 machines and in the case where a memory management driver is not specified for higher class CPU's.

The extra processing involved in supporting a virtual machine does have its consequences since extra overhead time is incurred. There will also probably always be some extremely ill behaved applications and general compatibility conflicts that will cause interference. However, given the utility of multitasking, it can be worth the compromises in all but the most stringent situations.

The next section describes which devices and system entities are global in nature and which are specific to a task. Following this is a discussion of the different CPU types and memory management levels which are supported by MOS.

Global and Private System Resources

The CONFIG.SYS File

MOS's CONFIG.SYS file is fairly similar to that of PC-DOS. It is used to define characteristics and resources of the system which are generally global in nature. Device drivers loaded by the DEVICE= statement are available to all tasks, although certain drivers could act in a task specific manner depending on their design. The MEMDEV driver, FREEMEM declarations, and configuration of the caching subsystem are common to all tasks.

Regarding numeric co-processor support, when MOS finds the CONFIG.SYS declaration: "8087=yes", it will allocate a small save buffer from the SMP for each task and use it to preserve the 8087's status between each task's time slice. Many math intensive applications will automatically make use of a math coprocessor when one is installed. If more than one application attempts to make use of the math chip at the same time and there is no CONFIG.SYS declaration, calculations will be corrupted. If your system has an 80287 co-processor chip, use "80287=yes" with "80387=yes" being used for that chip.

One CONFIG.SYS statement which is not directly task specific or global is VTYPE. The use of the "F" option will only affect the foreground task, whereas the VTYPE number affects the potential size of all background tasks.

AUTOEXEC.BAT and other Startup Batch Files

When a PC-MOS system is first booted up, the foreground task checks for an AUTOEXEC.BAT batch file in the root directory of the boot drive. If one is found, it is executed. When the ADDTASK command is used to start additional tasks, the name of a startup batch file can be specified. A unique file may be used for each task or a group of tasks may share one common file. It is also possible to make one batch file behave differently depending on which task it is executing in. See Chapter 4 for more details.

Device Drivers

Loading a device driver through the use of a DEVICE= statement in CONFIG.SYS will result in an entity which is globally accessible. Any interrupt vectors set by the driver will be reflected in the interrupt vector table of all tasks. The code and data for such drivers is placed within the SMP which is a non-switched type of memory.

When the ADDDEV command is used to install a device driver at run time (after the boot up process has completed), there are two options - global and task specific. A driver installed in a global manner with ADDDEV will produce the same result as a CONFIG.SYS loaded driver with one important exception. While references to the driver's device name can be made from any task, any interrupt vectors set by such a driver will only take effect in the task within which the ADDDEV command was invoked. To avoid conflicts, do not use ADDDEV with any drivers which set interrupt vectors. Add these types of drivers through the CONFIG.SYS file.

Using ADDDEV to install a driver in a task specific manner differs from the global case in that task memory is used instead of SMP memory. The code and data area required by the driver are allocated from the task in which ADDDEV is invoked in the same manner as a TSR utility. In addition, the driver will not be accessible to any other tasks through either the device name access method or through any interrupt vectors which may be set. There should generally be no problem installing a driver in this manner when it sets interrupt vectors.

Special Hardware

Add-on hardware such as network interface cards and intelligent serial communications boards typically work in conjunction with a device driver. The driver's design will determine whether these devices can be accessed simultaneously by multiple tasks with proper coordination. Mass storage peripherals such as tape drives, especially those used for backup and archive purposes, are usually not designed to support multiple accesses. An adapter card which does real world I/O, such as an A/D converter, would be another example of equipment which should only be accessed by one task. Such equipment is often used to interface laboratory instrumentation or a process control system.

Disk Files

Disk files are generally treated as globally accessible entities. MOS does not do any automatic file locking where a file's use by one task prevents simultaneous use by other tasks. MOS does provide file and record locking services to applications which are designed to use these features. However, this is strictly up to the application's involved. If you use a single user type of database in more than one task at the same time you can corrupt your data files. MOS's security feature can also provide selective access to files and directories.

Printers

Many different configurations are possible when it comes to printers. If no re-routing directives are used to customize your system's printing configuration, all tasks will see the first parallel port as the LPT1 device, with a second parallel port comprising LPT2, etc. If two tasks print to the same logical device at the same time, the result will not be usable. See Chapter 7 for a detailed discussion of printer routing.

Serial Ports

Serial ports are accessed by their specific port address by most telecommunications programs. Serial ports can also be accessed by a logical port number when a serial communications device driver is being used. In either case, serial ports are global items. What one task sees as COM1 is seen as COM1 by all other tasks.

Since most telecommunication packages are hard-coded to work with a port at address 03F8 or 02F8, only one task can run a program configured for the same port at the same time. Likewise, when accessing serial ports through the COMx device name method or through the INT14 system call interface, each task must be working with its own port or group of ports. Serial ports are covered in Chapter 8.

IRQ Lines

The use of IRQ signal lines in a computer system often relates to the use of serial ports. It can also be involved when interfacing tape drive units. In certain cases, the directive MOS USEIRQ n (where n is an IRQ number) must be issued to specify that a certain IRQ is associated with a certain task. No emulation is possible to virtualize IRQ's such that each task can have its own full set of IRQ signals. See Chapter 9 for more details.

Video Display Memory

When a paging-capable form of memory management is available, video memory is virtualized. This means that applications which attempt to write directly to the display RAM from a task other than the one the master console is viewing will not bleed through. If paging-capable memory management is not available, running many applications from a serial terminal workstation or in a background task will result in garbage on the master console's display. The only way to get around this is to work only with "safe" applications, of which there are few, or patch your present packages - a tedious prospect.

Standard System Hardware

When I/O trapping support does not exist in a multitasking environment, any of the standard I/O devices in a system are global in that they are fair game for any task which decides to lay claim to them. This is rarely a desirable condition. If an application which is supposed to be running at a serial terminal workstation attempts to move the cursor by directly programming the video controller hardware, the user at the master console will find their cursor darting about in ways they'd rather it not. Likewise, if a workstation application issues a beep in response to an invalid menu choice, the workstation user will only hear the beep if they are within earshot of the master console.

Direct access to the timer/counter chip, DMA chip, etc. will also produce improper results when made from more than one task at the same time. The keyboard controller can also be a source of confusion. TSR style utilities and other programs which respond to hot keys typically do so by direct I/O reads of the keyboard controller. If such a program is being run from a workstation other than the master console when I/O trapping is not supported, which keyboard will these programs respond to? When these programs directly read the keyboard controller, they're doing so with the address of the master console's hardware! Given the amount of hardware dependence of many applications, the I/O trapping feature of the 80386 chip is a valuable part of a multitasking environment.

How MOS Works With Different CPU Types

80386

When the 80386 memory management driver, $386.SYS, is used MOS initially operates in what is known as Virtual 8086 mode (or VM86 mode). This provides for the support of multiple virtual machines of the 8086 class as well as multiple tasks or virtual machines which run in 80386 protected mode. Capabilities for paging, I/O trapping and other types of exception processing are built into the 80386 chip. No external hardware is needed.

When the $386.SYS memory management driver is not installed, the system will run in the default 8086 real mode. This is the mode used when DOS is booted on an 80386 system and the paging, I/O trapping and exception handling capabilities of the CPU are not utilized.

While the 80386 CPU can run one 80286 protected mode task, PC-MOS does not presently support this operating mode. Applications which are designed to boot under PC-DOS on an AT and then put the CPU in 80286 protected mode will not run under PC-MOS.

80386 SX

As adapters become more available, this chip should provide an easy means to upgrade existing 80286 based AT's. Full 80386 paging and trapping features will be available but there will be some performance tradeoffs due to the use of narrower data and address busses.

80286

The default 8086 real mode is used when MOS is run on an 80286 system. there is currently no support in MOS for 80286 protected mode tasks. Applications which are designed to boot under PC-DOS on an AT and then put the CPU in 80286 protected mode will not run under PC-MOS.

With the default 8086 real mode, there is no built-in paging capability. However, paging may be achieved through either one of two add-on hardware devices. One is known as the AT Gizmo the other is the ALL ChargeCard. The corresponding memory management driver, $GIZMO.SYS or $CHARGE.SYS must be used with these products. The I/O trapping and exception handling features of the 80386 chip are not available on the 80286 chip.

8088

The 8086 real mode operation is the only choice. There is no built in paging capability but paging may be achieved through an external hardware device known as the All Card. The memory management driver $ALL.SYS must be used. I/O trapping and exception handling are not available on the 8088 chip.

8086

The 8086 chip suffers the same restrictions as the 8088 with the added drawback that there is no external add-on hardware available at this time to provide paging support.

How MOS Works With Different Memory Management Drivers

$386.SYS

When used in an 80386-based system, this driver supports paging (remapping), LIDT style interrupt control, I/O trapping and other exception processing.

$GIZMO.SYS and $CHARGE.SYS

When the corresponding hardware device is installed in an 80286 machine, memory may be managed in an equivalent manner to the 80386's paging. LIDT style interrupt control is also supported.

$286N.SYS

This driver provides LIDT support on 80286 machines when no AT Gizmo or ALL ChargeCard is available.

$ALL.SYS

When an ALL Card is used in an 8088 machine, memory may be managed in an equivalent manner to the 80386's paging. LIDT support is not possible on an 8088 so the REGRAB method must be used.

No Memory Management Driver

With no memory management there is no ability to protect from direct video writing or I/O programming. You must use extremely well behaved applications or patch the ones you have.

Special Considerations for the Memory Management Driver

There are some special compatibility situations which may require that certain options be selected in the memory management driver. It will sometimes be necessary to include a /I option at the end of the MEMDEV statement in your CONFIG.SYS. If you wish to use certain types of software debugging tools this option changes the way the MOS controls interrupts such as INT8 and INT9. The type of control used when the /I is not specified, LIDT control, is more comprehensive. However, one consequence of this tighter control can be a deadlock when a debugger is used which must also achieve tight control of the machine.

Including the /C option at the end of a MEMDEV statement will suppress MOS's use of the extra 60K of FREEMEM which can be accessed just above the first megabyte boundary. If an application is also attempting to make use of this wrap-around effect to address certain lower memory areas there could be a conflict. This option prevents MOS from using the wrap-around FREEMEM and, consequently, reduces the FREEMEM available for relocation of the operating system's components.

Virtualization of Video

The topic of virtualization cannot be fully explained without covering the aspects related to video display management. In MOS, video management is directly related to the types of workstations which are supported and the combinations in which they may be used. Since this isn't the most appropriate place in this manual for a complete discussion of workstations, a brief introduction will be given here with more detail given in Chapter 6.

The "video mode" of a system is normally determined by the type of physical display adapter that is installed in a computer. The type of adapter used determines the address of the video display memory as well as other characteristics of its operation. For example, under PC-DOS if you wish to use an application which requires a CGA display adapter but your system has only a monochrome card installed then you must change the card and, of course, use a compatible monitor.

The video modes inherently supported by MOS are monochrome, Hercules, CGA, EGA, and VGA. Other special types of video display systems will operate under MOS with certain constraints. In addition, MOS is not limited to using only the video mode or modes supported by the primary physical display adapter. Tasks running in the background (un-watched by any workstation) or running with a serial terminal type of workstation attached must present an emulation of a video mode, which can differ from the installed video hardware for the master console. Through the remapping of a video buffer into the video memory area and manipulation of video information in the task's BIOS data area, applications can be led to believe that a different video mode exists than that of the native display hardware.

There is no way to display graphics on a system with a monochrome display card and monitor.

MOS can emulate a monochrome video mode on a machine that has a physical CGA card as the native display adapter. This is the case when a typical PCTERM type of serial terminal is used since it is a monochrome type of workstation.

The foreground task's initial video mode is determined by the physical display adapter, just as it is in a single tasking PC-DOS situation. If a CGA video card is installed then the machine will start up in a CGA text mode. If a monochrome card is found on boot up then MONO mode will be in effect. When an EGA adapter is installed, the system will typically operate in a CGA mode upon power up.

When a background task is added (a task that is not initially associated with a workstation) its initial video mode will be that of the task which spawned it. When a workstation task is added, the terminal driver specified in the ADDTASK command determines the new task's initial video mode. One of the most commonly used terminal drivers, PCTERM.SYS, will cause the new task to have a monochrome video mode -- to match the monochrome type of display used with a PCTERM type of workstation.

Through a feature known as PAMswitching the task that a console is currently viewing can be changed. The foreground task is initially viewed by the master console, the console that is comprised of the primary physical display adapter (and associated monitor) and the keyboard attached to the host system's motherboard.

When a background task is added, it will initially be unwatched - there being no workstation console directly associated with it. By entering a special set of keystrokes a console can be detached from its current task and reattached to a different task. For example, to make the master console switch to task #1, you would hold down the ALT key and simultaneously press the 1 on the numeric keypad.

In the case where a workstation task is added using the PCTERM terminal driver on a system with a CGA master console, the master console may be PAMswitched to watch the workstation task since the text mode of a CGA adapter is capable of displaying the text that appears on the monochrome console. Likewise, when a task is using a CGA text mode a monochrome workstation console could be PAMswitched to watch successfully. However, when a CGA graphics mode is in effect in a certain task, PAMswitching a monochrome workstation console to watch would not work properly. Similar compatibility considerations exist with other types of workstations. The "MOS MAP" utility command may be used to see the current video mode of each task.

Through the use of the "MOS VMODE" utility the video mode of a task may be changed within limits. Since video mode emulation relies on a memory manager with remapping capabilities, this is a primary requirement. Furthermore, when a video mode change will require a task's video buffers to be increased in size there must, of course, be enough extended memory to satisfy this need. In a task with a monochrome video mode, two video buffers of 4K each are required. Two 16K buffers are required in the CGA case.

In a system with Hercules graphics capability (VNA - Video Network Adaptor) the initial video mode will be monochrome and a VTYPE declaration of 5 or 5f must be used in CONFIG.SYS in order to utilize the Hercules mode. A VTYPE of 2 or 2f may be used if only the monochrome mode will be used. In addition, the command "MOS VMODE HG1" (or "HG2") must be issued to instruct MOS that you intend run applications which will use a Hercules mode. The use of HG1 (Hercules graphics page 1) will result in two 32K video buffers being allocated from extended memory for each task in which this video mode is used. When support for the second Hercules page is needed, the "MOS VMODE HG2" command must be used and enough extended memory for one 32K buffer and one 64K buffer must be available.

Due to limitations inherent in the hardware design of the EGA display adapter, some additional considerations must be made. The EGA card is largely controlled through a write-only type of I/O register interface; an approach which has made support of this graphics adapter in a multiuser environment difficult at best. In order to virtualize an EGA task to the point where PAMswitching can be properly supported, MOS must make use of its I/O trapping capability to track the I/O write activities done to the EGA card's registers. As a result, when it is desired to PAMswitch to and from a task in which an EGA specific graphics mode is being used, a "MOS VMODE EGA" command must be issued prior to running the application which will use the EGA mode. Unfortunately, the additional overhead involved in trapping each I/O write operation done to the EGA card can result in a significant degradation of video performance.

Another constraint involved in running a task in an EGA or VGA specific video mode is that such a task cannot continue to run when it is not attached to an actual EGA console. Whereas a MONO or CGA mode task can be fully virtualized through a software emulation and can, therefore, continue to execute when unwatched, the increased complexity involved in the EGA and VGA graphics systems makes a software emulation impractical. When a task which is in one of these graphics modes is unwatched, it will be automatically suspended. As soon as an appropriate console is re-attached to the task, it will continue to execute from the point where it was suspended. Additional constraints of an EGA or VGA system are that a single 256K video buffer is required and no VTYPE declaration may be used to increase the maximum task size.