CHAPTER 2: SYSTEM CONFIGURATION

The System Memory Pool (SMP)

The MOS kernel uses a buffer area known as the system memory pool (SMP) to hold numerous types of information related to its operations. The SMP is used to store data about each task, hold the operating code and data for device drivers, and hold other data structures the kernel needs to track locks on files and records, etc. This buffer area is most similar to DOS's area for loading device drivers.

Under DOS, since the only thing the device driver loading area is used for is device drivers there is no need for a pre-defined size. This area is automatically made just large enough for the drivers loaded. In PC-MOS, much of the data that the SMP holds is dynamic in nature, and device drivers can even be loaded at runtime using the ADDDEV command. Since all of the operating system's components must be packed tightly together in memory at boot time, the size of the SMP must be predefined. This is done within the CONFIG.SYS file with the statement:

```
SMPSIZE=nnnk
```

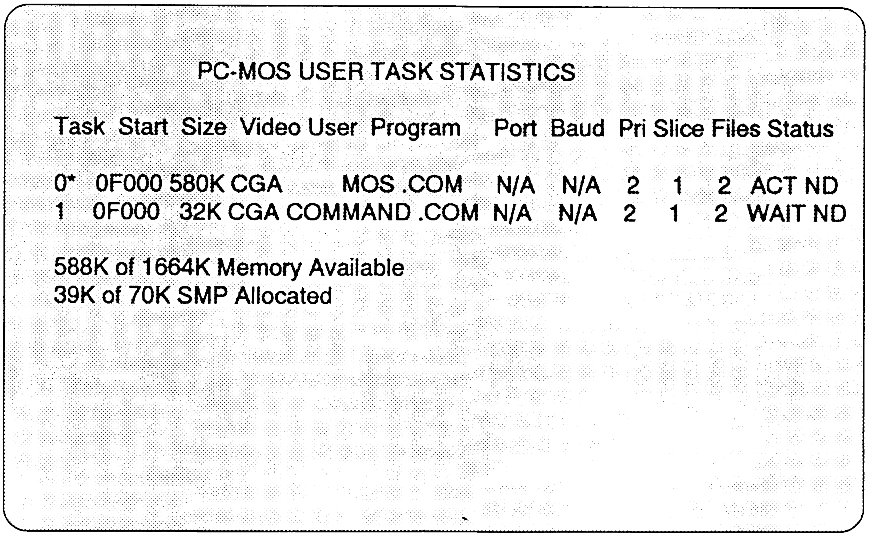
where nnn is the number of kilobytes of memory to allocate for the SMP.

Optimal sizing of the SMP is important. If the size chosen is too small your system will not be able to function properly. If too large an allocation is made, memory is wasted and the maximum possible task size may be reduced. The best approach is to start with a moderately large SMP, and then determine what is actually being used (explained below). Once some usage history has been collected, any extra can be trimmed back.

Regarding the SMP needs of device drivers, it is not always possible to determine the load requirements from the driver's file size. When a typical driver loads, there is some initialization code which will not be kept after the driver is installed. This means that the driver's file size could be larger than the actual need. Conversely, in some drivers, additional data buffer space is allocated after the driver's code so any estimates based on the file size will be very rough. If you cannot obtain clear specifications on the drivers you will be using, you'll need to start by allowing a healthy margin and monitor the actual usage. Note that unlike other uses of the SMP, once a device driver is loaded, it does not change the amount of memory it uses.

When MOS boots, some important statistics for the SMP are reported including the total SMP size, the percent currently in use and the percent used for device drivers. This information may be used to get an initial idea of what kind of margin you have. When first running a new installation through its paces, its also a good idea to monitor the SMP usage through some typical system operations. Once you've gotten to the point where your tasks are running and applications are loaded, measuring the SMP usage again is important to insure there is no waste.

This type of monitoring can be done through the use of the MOS MAP command. Note that you will need to keep a small utility task available in which MOS MAP may be run while your applications are loaded. Figure 2-1 shows an example of the information presented from this utility command. The bottom line is the one which reports SMP usage.



```
              PC-MOS USER TASK STATISTICS

Task  Start  Size  Video User  Program      Port  Baud  Pri Slice Files Status

0*  0F000 580K CGA        MOS .COM     N/A   N/A   2    1    2  ACT ND
1   0F000  32K CGA COMMAND .COM N/A   N/A   2    1    2  WAIT ND

588K of 1664K Memory Available
39K of 70K SMP Allocated
```

Figure 2 - 1

Information displayed by MOS MAP

There is one case where it actually makes sense to increase the size of the SMP. If there is a pocket of FREEMEM, 64K in size, and the SMP (sized at 61K), has been located within it, then the remaining 3K in that pocket will be unused. By increasing the SMP size to 64K, your margin is 3K larger and there has been no detrimental effect on memory allocation or the maximum task size.

However, if the SMP size were increased to 65K and the 64K pocket was the largest available FREEMEM area then the entire 65K of FREEMEM would have to be located within the base 640K of memory with the 64K pocket being used to hold other, smaller system components. When all components are finally located, it is likely that the maximum task size will be smaller than if a 64K SMP size was used. The utility command MOS INFO shows the FREEMEM list and the memory allocation used by each system component.

As stated above, in PC-DOS, the size of the area used to hold device drivers is automatically adjusted to fit the need at boot time. Since the size of MOS's SMP is predefined, if too small of a size is used (e.g. no margin) it is possible for a device driver to load itself over operating system code or data. At the point where the operating system can know this has happened, its ability to report the situation may be impaired. Leaving a margin is important.

Since the SMP is also used to hold data at run time, an insufficient SMP size can cause certain run time operations to fail. The most common example of this is when a program of the .EXE type is being loaded for execution. During this process, MOS must allocate a temporary buffer within the SMP. If there is not enough room for this allocation then a critical error message window will pop up to indicate the problem. When this happens you may need to increase the SMP's size and re-boot. If, however, the shortage occurred because of coincidental timing, trying again in a moment should get you going. It could be that several users were starting programs at the same time.

Relocation Strategy

In order to use the FREEMEM memory area to relocate MOS's system components out of the base 640K, MOS must first identify what FREEMEM areas are available. When MOS first boots it examines the memory region from C0000 to F0000. MOS attempts to determine which parts of this memory region are unused and which have memory devices allocated to them. The starting location and size of each free area is stored in a "FREEMEM list" with a maximum of five areas being cataloged.

Another way that FREEMEM can be identified is by including from one to five FREEMEM=nnnn,nnnn statements in the CONFIG.SYS file. The exact process of identifying these FREEMEM areas will be covered later in this section.

When there is enough free space, MOS will locate its system components within the FREEMEM area to the greatest extent possible. To make the best use of the available FREEMEM space, the largest components are relocated first. This is because any system components which cannot fit up in high memory must be located down in the base 640K area. If available FREEMEM should run out before space for all components can be allocated, it will be much better if the items remaining are the smaller ones.

Figure 1-6, in Chapter 1, showed the memory map which resulted when all system components were able to be located within the FREEMEM area. Figure 2-3 illustrates the consequences of not having enough FREEMEM to hold all components. Those which couldn't fit up high had to be put down low which reduces the maximum possible task size. A shortfall was made more likely in this case due to the 64K block from E0000 to F0000 being unavailable as FREEMEM.

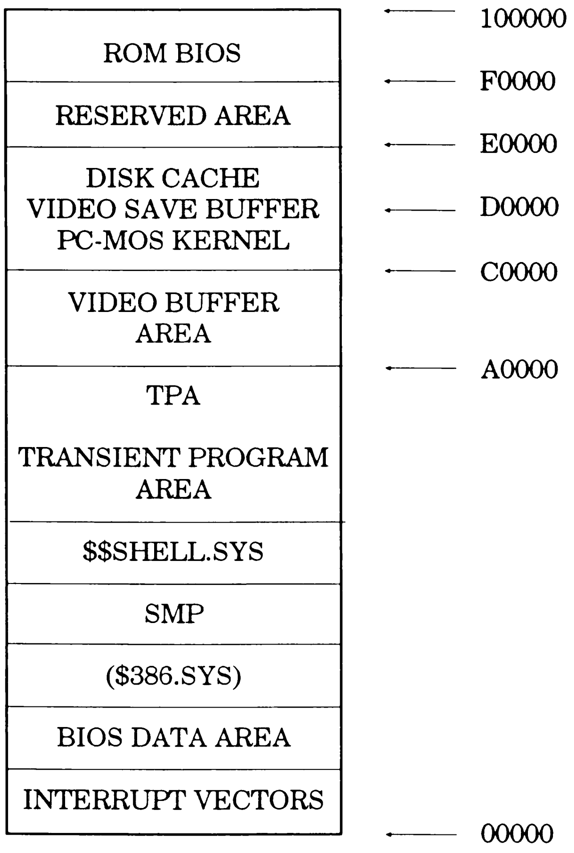


Figure 2 - 2

Memory map which results when there is not enough FREEMEM to locate all of the system components up high.

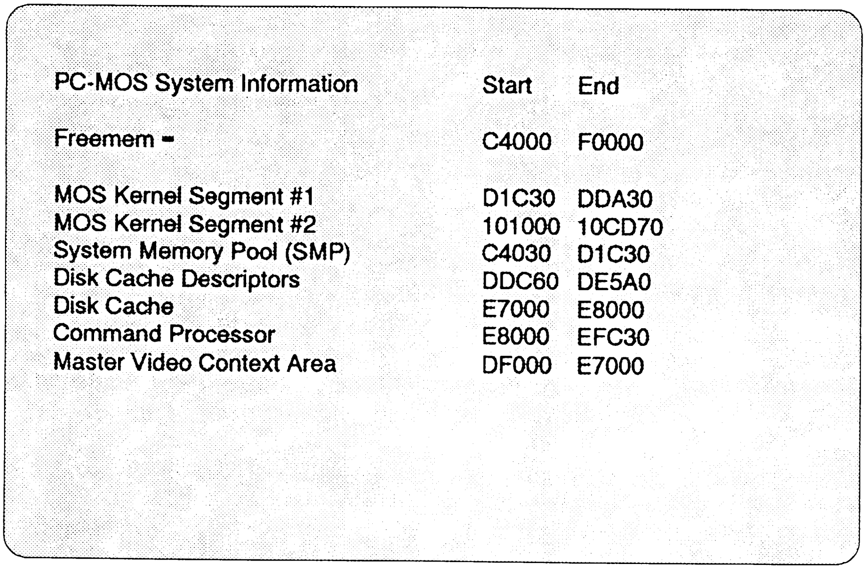
Beginning with release 2.30, the kernel can be relocated in two separate sections. The SMP and these two sections are sorted so the largest item is relocated first. With the kernel able to be divided into two sections, more of it is likely to fit up into a FREEMEM area.

FREEMEM Tuning

The MOS INFO command reports what FREEMEM is being used and how it is used to hold MOS's system components. If you did not specify any FREEMEM statements in your CONFIG.SYS file, then MOS will do an automatic search for FREEMEM and the results of this search will be what is reported by the MOS INFO command. Otherwise, the MOS INFO display will simply reflect the FREEMEM ranges you specified. If any FREEMEM statements are included in CONFIG.SYS the automatic search is not done.

Figure 2-3 shows a sample of the display output produced by MOS INFO. In figure 2-4, this data is presented in order of memory position. (Note that this is a manipulation done for clarity, the MOS INFO command displays its data as shown in figure 2-3). Due primarily to careful sizing of the SMP, all of MOS's system components are able to fit up into high memory. Notice how the ending address of the Command Processor, at EFC30, is quite near the upper FREEMEM limit of F0000.

To illustrate more clearly what happens when you run out of FREEMEM, another sample output from MOS INFO is provided in figure 2-5 with the corresponding sorted-by-memory view in figure 2-6. By increasing the SMPSIZE declaration in CONFIG.SYS by just enough to go over the threshold point, the Command Processor was forced to be located down in the base 640K memory area. This reduces the maximum possible task size by the size of the command processor - which is presently just under 32K. In the sample configuration, an SMP size increase from 55K to 60K caused this to happen.



```
PC-MOS System Information                  Start   End

Freemem -                                  C4000   F0000

MOS Kernel Segment #1                      D1C30   DDA30
MOS Kernel Segment #2                      101000  10CD70
System Memory Pool (SMP)                   C4030   D1C30
Disk Cache Descriptors                     DDC60   DE5A0
Disk Cache                                 E7000   E8000
Command Processor                          E8000   EFC30
Master Video Context Area                  DF000   E7000
```

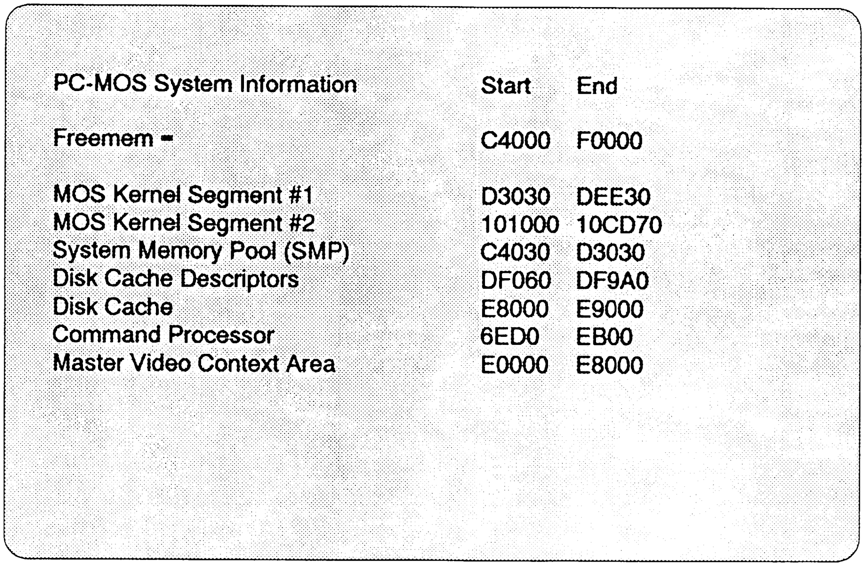
Figure 2 - 3

A sample of the MOS INFO display

System Memory Pool (SMP)	C4030	D1C30
MOS Kernel Segment #1	D1C30	DDA30
Disk Cache Descriptors	DDC60	DE5A0
Master Video Context Area	DF000	E7000
Disk Cache	E7000	E8000
Command Processor	E8000	EFC30
MOS Kernel Segment #2	101000	10CD70

Figure 2 - 4

The information from figure 2-3 re-ordered by memory address range



```
PC-MOS System Information              Start   End

Freemem -                              C4000   F0000

MOS Kernel Segment #1                  D3030   DEE30
MOS Kernel Segment #2                  101000  10CD70
System Memory Pool (SMP)               C4030   D3030
Disk Cache Descriptors                 DF060   DF9A0
Disk Cache                             E8000   E9000
Command Processor                      6ED0    EB00
Master Video Context Area              E0000   E8000
```

Figure 2 - 5

Another sample of the MOS INFO display. Note that a larger SMP has forced the Command Processor to be located down in the lower base 640k area.

Command Processor	6ED0	EB00
System Memory Pool (SMP)	C4030	D3030
MOS Kernel Segment #1	D3030	DEE30
Disk Cache Descriptors	DF060	DF9A0
Master Video Context Area	E0000	E8000
Disk Cache	E8000	E9000
MOS Kernel Segment #2	101000	10CD70

Figure 2 - 6

The information from figure 2-5 re-ordered by memory address range

The size of the kernel and command processor do not change within a given release of MOS and the size of the main disk cache window is fixed at 4K when memory management is available (which is, incidentally, required for any discussion of FREEMEM to be relevant). Since the size of the disk cache descriptor area is fairly small in the first place, the only two system components which can be varied are the Master Video Context Area and the SMP.

In the sample configurations shown in figures 2-3 through 2-6, the video context area was sized at 16K since a CGA video adapter was installed. If you could live with the restriction of using no other video mode but MONO in any task on your system, you could use a VTYPE of 2 or 2F and the video context area would be reduced to 4K. The VTYPE statement will be covered later in this chapter.

In most cases, the size of the SMP is the factor which most directly affects FREEMEM fallout. This is why using the information provided by MOS INFO and MOS MAP can be valuable in that it allows you to fine tune your SMPSIZE setting. Note that in the case illustrated in figure 2-5, if an SMPSIZE which forces the Command Processor out of FREEMEM is unavoidable, you might just as well fill the SMP out by another 28K. This lets you make use of the memory from the end of the disk cache area to the start of the ROM BIOS (from E9000 to F0000). A calculator capable of hexadecimal math will be very useful in this type of work.

If you look closely at the MOS INFO display you may notice some small waste. The video save area (also known as the Master Video Context Area) must start on a 4K boundary which can result in some unused space (a maximum of 4K-1 bytes). This is because the 80386's paging capability works with 4K blocks on 4K boundaries. The window for the caching buffer must also be 4K aligned.

When using any type of utility to determine which parts of your computer's FREEMEM areas are available and which parts are in use, you must bear in mind that, depending on what is presently located in the C0000 to F0000 range, the memory test that the utility performs may upset your system's operation. Such tools usually test for RAM by writing a data pattern to memory and then seeing if it can be read back. This type of tool must not be used when any parts of MOS are presently located into FREEMEM, since the tool's memory test will corrupt those parts of MOS. MOS does its testing during the bootup process when the initialization code is temporarily running in the base 640K memory area.

It is also possible to upset an adapter card by running a memory test if the card is using a block of RAM within the FREEMEM area. For example, if a "smart" communications card has a 4K block of RAM at address D0000 and the card's dedicated processor is programmed to react to certain values within that memory block, the test program could cause the dedicated processor to get an incorrect stimulus. In such cases you will have to consult the manufacturer's literature or technical support to find out what areas are used.

Note that prior to release 2.30, the only area tested for FREEMEM was from C0000 to F0000. Beginning with 2.30, an additional 60K area just above the 1 meg boundary is also available. This is possible through a special exception to the constraint that only memory within the 1st megabyte is directly addressable to a CPU running in an 8086 mode.

If the master console's video adapter is a simple CGA type card (not an EGA running in a CGA mode) and there will be no Hercules workstations in your system, you can use explicit FREEMEM statements to take advantage of the 16K memory region from BC000 to C0000 as well as the 16K region from B4000 to B8000. Thus you could use the following pair of FREEMEM statements:

```
FREEMEM=B4000,B8000
FREEMEM=BC000,F0000
```

The second statement shown presumes that no other gaps exist within the FREEMEM memory area. You must adjust accordingly if your system does have gaps. Also, a conflict with this special use of the B4000 to B8000 region can arise when a feature known as VTYPE is used to raise the upper boundary of the TPA. See the discussion of VTYPE=3 and VTYPE=4 later in this chapter.

Certain 80386-based systems have a special purpose block of RAM located within a part of the FREEMEM region. The contents of the ROM BIOS, which normally begins at address F0000, are copied into this high speed RAM to improve the performance of BIOS system call execution. This RAM, often referred to as "Shadow RAM", usually resides in the E0000 to F0000 memory region.

Since MOS's code is used in place of a majority of that of the ROM BIOS, it is more beneficial to defeat this copying of ROM into RAM and allow the area it occupies to be available as FREEMEM. In some machines, MOS can defeat this relocation scheme automatically and in others changing a CMOS setup parameter, motherboard jumper or DIP switch may be necessary. There also are some machines where it is not possible to defeat this relocation at all. Consult the documentation for your machine or the technical support department of its manufacturer.

FREEMEM Usage by Adapter Cards

Some add-in circuit cards will have on-board RAM or ROM memory allocated within part of the FREEMEM area. For example, many EGA display cards have an EGA ROM BIOS allocated from C0000 to C4000 (C0000 to C3FFF inclusive). Other types of adapters which typically use FREEMEM are intelligent serial I/O cards, special purpose video adapters and some hard disk controllers.

Some of these cards do not allow their memory allocation to be adjusted -- a constraint which can cause problems in any operating system environment (e.g. mixing two conflicting cards is prohibited). When the addressing of such memory is adjustable, it is best to work for contiguous remainder of FREEMEM since large FREEMEM blocks are more advantageous than smaller ones. The more of MOS's larger system components that can be located within the FREEMEM area, the larger the maximum task size can be. In other words, you should try to configure all such adapters so that their allocations are packed up just below the top limit (F0000) or just above the bottom limit (C0000).

A certain portion of the built-in ROM BIOS code is only used during the initial POST process (Power On Self Test). This code is responsible for the memory test and system's checkout that occurs when your computer is booted. Once these tests have been completed, the operating system code is loaded from the disk and given control. From this time on, the code for the POST process is no longer needed.

If it can be determined that a section of ROM code is only required during the POST process and, if the location of this code within the ROM BIOS can be precisely determined, it can be reclaimed as FREEMEM through the use of an explicit FREEMEM statement. Note that once any explicit FREEMEM statement is used, MOS does not do a FREEMEM search of the upper memory area from C0000 to F0000. You must, therefore, include explicit FREEMEM statements for all available areas. A further constraint is that a FREEMEM block must start and end on a 4K boundary.

FREEMEM Usage by Systems Device Drivers

Certain of the device drivers supplied with MOS require the reservation of a window within a portion of the FREEMEM area. The $EMS.SYS driver, which uses extended memory to emulate an expanded memory board, requires a 64K window to operate. The $RAMDISK.SYS driver requires a 16K window and, as with any use of page frames, the windows for both of these drivers must begin on a 4K boundary.

Just as with hardware adapter cards which use a portion of the FREEMEM area, it is best to adjust the area used by these drivers to leave one contiguous FREEMEM area for the relocation of system components. Also, be aware that each of these drivers assumes a certain standard address for the start of its memory window if none is explicitly specified in the CONFIG.SYS parameter line. $RAMDISK.SYS uses B4000 as its default while $EMS uses E0000.

Using Explicit FREEMEM Statements

When MOS's boot up FREEMEM search can properly detect which areas are free, this relocation process will be automatic unless FREEMEM statements are found in the CONFIG.SYS file. In certain situations, however, the hardware involved escapes detection. When this happens, if MOS presumes that it is safe to relocate its operating code and buffers into a part of this FREEMEM region, conflicts can occur. In such cases, one or more FREEMEM statements must be used in the CONFIG.SYS file to explicitly declare to MOS which memory regions are available (and, thus, which ones are not). When any FREEMEM statements are found in the CONFIG.SYS file, MOS does not search the upper memory and try to determine what is available, the memory ranges specified are taken as all there is.

It is generally wiser to use explicit FREEMEM statements. However, you must remember that they are in use. If a new adapter card is installed which has memory allocated within the FREEMEM space then you must remember to update your FREEMEM declarations to prevent conflicts. Likewise, if you remove such an adapter, you should update the FREEMEM declarations to allow MOS to use all available space for its system components.

To document which FREEMEM areas are in use by which devices you can put comment lines in between FREEMEM lines to identify what each gap is for. Starting a CONFIG.SYS line with a semicolon (;) makes it a comment. In addition, the REM statement can be used to create a comment line which will show on the display during the boot process. The following line could be entered into a CONFIG.SYS file to provide a reminder of the system's configuration at each boot up:

```
REM= FREEMEM gaps left for EGA ROM BIOS and Smart Serial card
```

Finding and Resolving FREEMEM Conflicts

When you are having trouble bringing up a system, you need to start out with the simplest possible configuration. One way to simplify the situation is to use the declaration "FREEMEM=n" (where "n" means none) to prevent MOS from locating any of its system components within the FREEMEM area. This means that the kernel, SMP, command processor and related buffers will all be located in the lower part of the base memory. No other FREEMEM declarations should be active when FREEMEM=n is being used. You should delete or comment out any you might have. Insert a semicolon (;) in front of a CONFIG.SYS line to temporarily turn it into a comment.

Once you get up and running without any FREEMEM usage you should introduce FREEMEM areas in stages so it can be determined which one is critical (presuming you can't find out from consulting the documentation for your machine).

This will mean editing your CONFIG.SYS file and rebooting for each trial but the benefits of increased maximum task size will make it worthwhile. If you can find the area which is causing the conflict, simply don't include it in your FREEMEM declarations. Be sure to document what you've done and why, by using comment lines directly within your CONFIG.SYS file.

Since you will be using explicit FREEMEM declarations for the areas you have found to be usable, you must be sure to declare all available FREEMEM areas with explicit statements. Remember, the existence of any explicit FREEMEM statements in the CONFIG.SYS file prevents MOS from doing an automatic FREEMEM search.

The VTYPE Directive - Reclaiming Unused Video Memory

When IBM laid out the memory map of the PC, it defined the address region from A0000 to C0000 as reserved for video adapters. This effectively put a cap on the top of the Transient Program Area. However, only the EGA and VGA actually make use of video memory starting at A0000. The monochrome display adapter uses only 4K of memory space from B0000 through B0FFF and the CGA adapter uses 16K from B8000 through BBFFF.

Being where it is, the video memory region is both a barrier and dividing line, imposing a ceiling on the TPA. Rather than let this space go to waste, the MOS kernel supports the remapping of extended memory into portions of the video memory area beginning at A0000. This allows the upper ceiling of the TPA to be raised, resulting in larger maximum task sizes. In the case of CGA, the 96K memory region from A0000 to B8000 can be made available for TPA use and on a monochrome system, the 64K memory range from A0000 to B0000 is available.

How does MOS know how much of this memory can be reclaimed? The variety of video modes possible in MOS means that there is no way for MOS to tell, by just testing the hardware at boot time, which video modes will require support. MOS could detect that there is a physical CGA video card installed and re-claim A0000 to B8000 for TPA -- but, this would cause a conflict if you tried to start a workstation task that required a monochrome mode since the B0000 to B4000 address range needed for the video buffer could also be used as TPA memory when the task size is large.

It is up to the system administrator to declare to MOS how much video memory can be reclaimed. This is done through a CONFIG.SYS statement known as VTYPE which is short for Video TYPE. The VTYPE declaration is basically a way of telling MOS: "I'll never use a video mode that needs memory in this area so you can reclaim it for task memory". The memory map in Figure 2-7 shows how the TPA memory region can extend up to B0000 when no EGA or VGA video equipment is installed and a VTYPE of 1 or 2 is used.

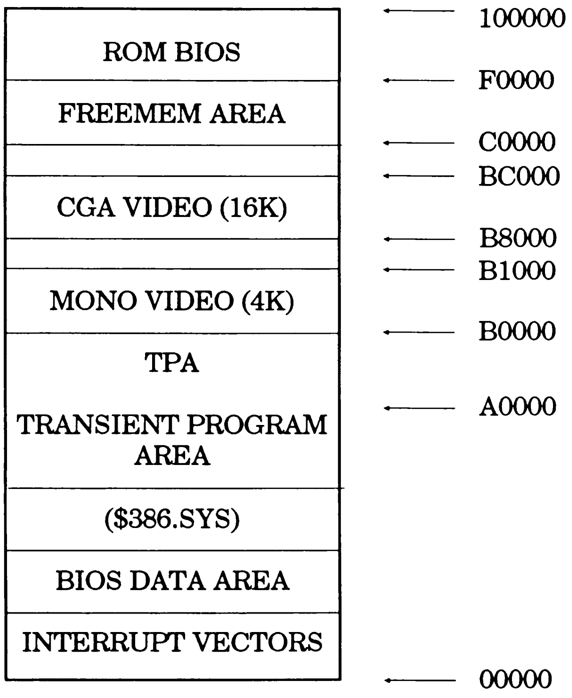


Figure 2 - 7

When a VTYPE of 1 or 2 can be used, the TPA can overtake the A0000 - B0000 memory range usually reserved for EGA/VGA video memory.

Note that when an EGA adapter is used, no extra space is available. The address range from A0000 to C0000 is allocated to the video RAM (with C0000 to C4000 typically being used for the EGA ROM BIOS). Also, if a paging-capable memory management driver is not being used then the VTYPE declaration can not have any effect on the TPA's upper ceiling. However, it does still affect the size of the video save area.

VTYPE Values

No VTYPE Specification

When no VTYPE declaration is made, MOS must presume that EGA or VGA video equipment may be installed. This prevents the A0000 to B0000 memory region from being available as additional TPA space since it must serve as the video memory area for the EGA/VGA video adapter. With the upper task boundary fixed at A0000, the largest possible task size will be in the neighborhood of 600K. This is presuming that all of MOS's system components can be relocated into the FREEMEM area rather than consuming any of the base 640K memory area.

In preparation for a CGA video mode, the master video context area is sized at 16K, which matches the size of the video buffers required to virtualize a CGA display system (virtualization of video is covered in Chapter 3).

If your system does not have EGA/VGA equipment installed, and paging-capable memory management support exists, there is basically no reason not to specify a VTYPE. Including a VTYPE declaration in your CONFIG.SYS keeps the option open for adding a task which can take advantage of at least 64K of additional head room.

VTYPE=1 [F]

Including the statement VTYPE=1 in your CONFIG.SYS file is a way of telling MOS that you definitely do not intend to use either an EGA or VGA type of video mode. Doing so would present a conflict between the use of the A0000 to B0000 memory range as task memory and video memory.

Although you certainly don't have to create tasks any larger 600K as long as a smaller size suits your needs. When the need for growth presents itself, using a VTYPE of 1 will allow an additional 64K of task memory. Given the 600K estimate made above, this translates into a task size of about 664K.

In this case, the master video context area occupies 16K so CGA video virtualization can be accommodated. The effects of including the "F" option after the VTYPE number will be covered in the next section.

VTYPE=2[F]

This version is basically a variation of VTYPE=1 which results in a master video context area of 4K in size rather than 16K. In systems where the master console's display adapter is of the monochrome type and no other video modes will be used, this reduction in the size of the master context area can have a beneficial effect on FREEMEM tuning. This configuration precludes the use of workstation tasks where 16K CGA video buffers are needed.

VTYPE=3[F]

VTYPE=3 results in an upper task ceiling of B4000. This configuration offers a balanced approach in the case where no monochrome tasks will be required. Raising the task ceiling to B4000 can yield a decent task size, as described by the following formula:

600K + 64K (for A0000 to B0000) + 16K (for B0000 to B4000) = 680K

In addition, The 16K memory area from B4000 to B8000 is still available for use as the memory window for $RAMDISK.SYS or as FREEMEM.

If you do plan to take advantage of this approach, understand that the memory area from B0000 to B1000 is no longer reserved for video memory when a monochrome video mode is to be used. This means the master console must use a CGA adapter card and no tasks can be added which would use a monochrome video mode. A PCTERM type of serial terminal workstation task would be an example of one which requires a monochrome video mode. In anticipation of CGA support, the master video context area is sized at 16K.

VTYPE=4[F]

This is the next progression from the VTYPE=3 discussion. As such, it carries the same restrictions on video modes (e.g. no MONO mode) as the in the previous section. With an upper task limit of B8000, expect a maximum task size of about 696K.

VTYPE=5[F]

A VTYPE declaration of 5 is intended for use with The Software Link's workstations known as VNA (Video Network Adaptor). These workstations support monochrome text mode as well as Hercules graphics. To handle the video buffers used in this environment, the size of the master video context area must be increased to a size of 32K and the upper task limit is set to B0000. This results in the same task size limit of 664K as in the case of VTYPE=1.

The Use of the F Option

Using VTYPE with an "F" after the number will cause the memory of the foreground task to be filled out based on the VTYPE value chosen. This is done by remapping extended memory into the unused video memory area during the foreground's time slice. If the foreground task will be large enough for your purposes without using the F option, not using it will leave more extended memory available for other things such as task RAM for background tasks, RAMdisk, EMS, cache, etc.

When the F option is being used, the foreground task size, as reported by MOS MAP, represents the maximum possible task size which may be spawned with the ADDTASK command (providing there is enough extended memory available to make another task of that size). When the F option is not used, the MOS MAP display will show the present size of the foreground - but background tasks could be larger. For example, in a monochrome system, using VTYPE=1 might yield a foreground task size of 608K but the maximum size for a background task would be 672K (608K + 64K).

Task Size Considerations

First, a definition of terms is in order. The term "task" refers to the group of entities which comprise a virtual machine. The TPA (transient program area) is one of these entities. Some other memory entities involved in the support of a virtual machine are video buffers, context buffers and buffers to preserve the state of a numeric co-processor and the EMS system (when these options are used). You will also find that the term "partition" is sometimes used to mean the same thing as "task".

When an application states that it needs a certain minimum amount of memory to operate, the overhead of PC-DOS is figured in. For example, if XYZ accounting package is documented as requiring 640K, the interrupt vectors, BIOS data area, device drivers, disk buffers, the PC-DOS operating system kernel and the command processor must also reside within this 640K so the actual memory available for the accounting package is not likely to be much larger than 570K.

Under PC-MOS only about 8K of each task's TPA is taken up by operating system overhead (a data area for the command processor). As a result, a PC-MOS task size of approximately 580K should be enough to satisfy the minimum requirements of XYZ accounting.

Being one contiguous block of memory, a task's TPA size is governed by the positions of its lower and upper boundaries. In the interest of achieving the largest possible size for the TPA, we have seen how the lower boundary can be kept as low as possible by taking advantage of FREEMEM areas to provide an alternate location for MOS's system components. Locating the operating system overhead in FREEMEM areas keeps these entities from using up memory from the lower part of the base 640K area -- the area from which a task's TPA must come.

Through the use of the VTYPE declaration, it is possible to raise a task's upper TPA boundary given an agreeable type of display adapter. For example, on a CGA system MOS can remap extended memory pages into the area from A0000 to B8000, providing an additional 96K of address space for each task that needs it. In the case where VTYPE's "F" option is not used, the limiting factor for the foreground task is the amount of actual physical system RAM installed within the base memory area of the machine.

Therefore, on a system with a monochrome display adapter and 512K of physical base memory installed, using a VTYPE=1 declaration will raise the upper limit of all tasks except the foreground up to B0000. The TPA of the foreground task will stop at 7FFFF since that is the highest usable base memory address in a 512K system.

These factors are important in determining the largest potential task size which can be supported on a particular machine. It should be noted that just because a VTYPE is used which allows a task's TPA to top out at B0000 (or B8000, etc.) the user always has the option to choose a smaller task size as is appropriate. In addition, having the potential to create a large task (a task with a large TPA) is only of use when there is enough extended memory and SMP buffer space available to fill the task's requirements.

To this point, it has been assumed that paging- capable memory management support is available. When paging, or remapping, is not possible then there is no support for the VTYPE option, no support for the relocation of the operating system into FREEMEM and no support for the re-using of the same address space for each task. After the operating system's overhead has taken what it must from the lower portion of the base memory area, what little remains must be split up to comprise each task's TPA.

Also, when remapping support is not available, an additional amount of system overhead is placed within each task's TPA for what is called the video save buffer, thus reducing the TPA's usable size. The amount of memory required for the this buffer is based on the video mode of the task. If a monochrome video mode is suitable for your installation, the resulting 4K encroachment on each task's TPA is much better that the 16K buffer size required to support a CGA system.

Estimating Extended Memory Needs

For MOS to be able to use extended memory, hardware which supports memory management must be available and the corresponding memory management device driver must be installed. This driver provides services to support the allocation and re-mapping of extended memory. During the boot up process MOS calls upon this driver to allocate and map extended memory into FREEMEM areas to make these areas usable. This use of extended memory is a fixed type of overhead since it does not change once established. Another type of fixed allocation of extended memory comes from the use of systems device drivers such as $RAMDISK.SYS and $EMS.SYS and from the disk caching sub-system.

If a VTYPE declaration is used which includes the "F" option, the kernel calls upon the memory management driver to allocate and map extended memory into the otherwise unused area between the top of the base memory and the start of the actual video buffer (e.g. B0000 for MONO, B8000 for CGA). The use of the "F" option causes this memory filling to be done only for the foreground task. This would also be considered a fixed use of extended memory.

When the ADDTASK command is invoked, extended memory is used to make up the TPA and the context and video buffers which comprise the new task. The allocation required for the TPA corresponds directly to the task size specified with the ADDTASK command. In the case of the foreground task, while no extended memory is needed for the TPA (unless VTYPE's "F" option is involved), extended memory is required for its video buffers.

In order to account for the buffer known as the context buffer in an estimate, the machine type must be considered. In an 80386-based system or an 8088 system with an All Card™ type of add in memory management device, a context buffer of 4K must be allocated from extended memory for each task except the foreground. In 80286-based systems that use an AT Gizmo™ or a ChargeCard™ type of add in device the memory for this context buffer comes from the SMP rather than from extended memory. Note that the AT Gizmo and ChargeCard do support paging but other factors preclude the use of that feature for the context buffer. More details on the All Card, ChargeCard and AT Gizmo devices can be found in Chapter 3.

Estimating the extended memory requirements for the video buffers is a bit more involved. Two buffers are required -- a video save buffer and a VIDRAM buffer. MOS uses these buffers to accomplish the virtualization of a physical video display adapter for a task. This is required to support tasks which will operate without a workstation being attached to them at all times and for workstations of the serial variety (often referred to as 'dumb' terminals or, in PC-MOS, PCTERM type terminals).

In conjunction with other operating system features, MOS uses these buffers to, in essence, "fool" an application into perceiving that it has a physical display adapter at its disposal when it actually does not. Without this capability, the display output of an application running in a task other than the one the master console is currently viewing would end up appearing on the master console's monitor. The master console is the keyboard, display adapter and monitor which is initially associated with the foreground task.

In addition to the VIDRAM buffer and the video save buffer, an entity referred to as the video save area is involved in this display adapter virtualization technique. The position of this entity in memory can be reviewed with the MOS INFO command. This area is where each task's video save buffer is mapped into when that task is receiving its time slice of execution. All tasks can share the same area since only one task will need to use it at a time.

When the video save area can be located within the FREEMEM region, there is initially no allocation of extended memory associated with it -- it is just a mapping window which is reserved for future use. Since the size of a task's video save buffer will vary depending on the task's current video mode the video save area must be large enough to accommodate the largest possible need by any task within the system. When no VTYPE declaration is used the size of the video save area is based on the initial video mode of the physical display adapter as detected at boot time. For a CGA system, the video save buffer would be 16K with 4K being used for a monochrome system. The section on VTYPE shows the size of the video save area for each VTYPE value when a VTYPE declaration is used.

When the location of the video save area falls within the FREEMEM region then the memory for the foreground's video save buffer must come from an allocation of extended memory. In the case where there is too little FREEMEM for the video save area to find its home there, it will be located down in the base memory area and the actual physical base memory within the video save area is used for the foreground's video save buffer. Therefore, no extended memory is required.

The memory for the VIDRAM buffer must always be allocated from the extended memory pool and the sizes of the video save and VIDRAM buffers will be the same in all but a couple of special cases. As with the video save area, the primary factor in determining the size of these buffers is the task's initial video mode. In the case of the foreground task, this depends upon the type of physical display adapter installed in the machine. It should be noted that when an EGA adapter is installed, which starts up in a CGA mode by default, MOS will treat this as a simple CGA case, with the peculiarities of EGA coming into play later on, when specific EGA graphics modes are used.

When the ADDTASK command is used to spawn a new task the initial video mode of this new task depends on the type of workstation, if any, which is initially associated with the task. The initial video mode of a background task is taken from the video mode of the task from which the spawning was done. A background task is one which is started without any workstation initially involved. However, a compatible workstation can "switch in" to watch this new task later on.

Providing that the workstations involved are compatible, a command called MOS VMODE may be used to change a task's current video mode. This will result in a change in the extended memory allocation. The allocation may go up or down depending on the transition so this must also be figured into an estimate of extended memory needs. The transition from CGA to MONO involves the release of two 16K video buffers and the acquisition of two 4K buffers.

In a system containing an EGA or VGA master console, a CGA video mode will typically be the default mode that the display adapter powers up in. In this case, or in the case where a simple CGA video adapter is installed, each task will have two 16K video buffers (a video save buffer and a vidram buffer). The use of any of the EGA or VGA specific video modes involves the allocation of one 256K video save buffer. This includes the following:

```
MOS VMODE EGA
MOS VMODE E40
MOS VMODE E43
MOS VMODE VGA
MOS VMODE V40
```

The use of one of these video modes is required when you need to be able to PAMswitch in and out of a task which is running in an EGA or VGA graphics mode. If your EGA/VGA display will only use its CGA graphics mode or text mode, or if you will not need to PAMswitch while running in a graphics mode, the use of this VMODE is not necessary. Paging-capable memory management support is required for any of these video modes and only a video save buffer is used, the second video buffer, known as a VIDRAM buffer, is not needed.

When workstations only require monochrome text mode support, a VMODE of MONO will be in effect by default and two video buffers of 4K each are required per task. If no other mode but MONO will be used within a system then you may want to use a VTYPE of 2 or 2F to reduce the size of the master video context area to 4K.

To support Hercules graphics on VNA workstations, a MOS VMODE HG1 or MOS VMODE HG2 must be issued. The HG1 version results in a 32K video save buffer and a 32K vidram buffer. This mode is suitable for applications which only use the first Hercules graphics page. When support for the second page is required, HG2 must be used. This means a 32K video save buffer and a 64K vidram buffer. Unfortunately, whether or not an application uses the second page is often a matter which can only be determined by experimentation. If you can get away with HG1's smaller memory requirements and your application runs with a proper display then you are probably safe.

In order to tie all this information together in a usable fashion, an estimate for a sample system configuration has been worked out below:

ONE TIME FIXED OVERHEAD:

192K	to fill out the C0000 to F0000 FREEMEM range.
64K	to fill out the TPA for the foreground since a VTYPE=1F will be used.
32K	for the two video buffers for the foreground task (CGA).
256K	for the disk cache.
256K	for a RAMdisk.

Total extended memory due to fixed allocations = 800K

MEMORY NEEDS PER BACKGROUND TASK:

Two small background tasks with a CGA video mode will be used as utility tasks. In addition, one full size PCTERM style workstation task exists.

200K	for the TPA's of the two background tasks (100K each)
64K	for video buffers for the two background tasks
8K	for both of their context buffers
600K	for the workstation task's TPA
8K	for the workstation's video buffers
4K	for its context buffer
	Total allocation per task for the 3 tasks = 886K

Total extended memory requirements = 886 + 800 = 1686k or roughly 1.5 Megabytes. In such a case, it would be wise to allow a margin and install 2 Megabytes.