OCTOBER 1992

NOVELL® RESEARCH

Managing Memory in a DOS Workstation: Using MS-DOS 5.0 and Windows 3.1

Edward A. Liebing
Senior Technical Editor
Systems Engineering Division

This AppNote is the second in an ongoing series that explains how to use various memory managers to maximize the amount of memory available for running applications. This AppNote looks at the EMM386.EXE and HIMEM.SYS memory managers that are included with Microsoft's MS-DOS 5.0 and Windows 3.1.

Contents

Acknowledgements

Thanks to Earle Wells, Tom Mattingly, and Sandra Duncan of Novell for their help in ensuring that this information is technically accurate.

Introduction

In the first AppNote of this series, Managing Memory in a DOS Workstation: Part 1 in the August 1992 NetWare Application Notes, we covered the basics for understanding the types of memory available in a PC. This Application Note continues from there and looks at two Microsoft products--MS-DOS 5.0 and Windows 3.1--that come with memory management software. Both of these include essentially the same memory managers: EMM386.EXE and HIMEM.SYS.

This AppNote looks at various ways of using Microsoft's memory managers to increase the amount of conventional memory available in a DOS workstation. It discusses how to order the loading of drivers and shows a number of CONFIG.SYS and AUTOEXEC.BAT file examples. It looks at how to best load drivers for NetWare v3.x and for NetWare Lite.

Memory Managers: MS-DOS 5.0 and Windows 3.1

Microsoft includes two memory managers with both MS-DOS 5.0 and Windows 3.1:EMM386.EXE and HIMEM.SYS. The EMM386.EXE program simulates expanded memory in 386- or 486-based computers with extended memory. It is also used to create upper memory blocks (UMBs) in the available area between A000 to FFFF (640KB to 1024KB). With DOS 5.0, this is where programs are placed when you load them high.

HIMEM.SYS is an extended memory manager for programs that can run in extended memory, such as Windows. It is also used to access the 64KB of high memory area (HMA) that lies above the 1024KB boundary.

(For an explanation of expanded and extended memory, UMBs, and the HMA, see Managing Memory in a DOS Workstation: Part 1 in the August 1992 NetWare Application Notes.)

Along with EMM386.EXE and HIMEM.SYS, you'll be using other DOS 5.0 commands, including:

- LOADHIGH and DEVICEHIGH to place programs into upper memory blocks
- DOS=HIGH, UMB to place DOS into HMA and establish a link between DOS and the UMB area
- "MEM /C" to track where your programs end up in memory

The "MEM /C" command doesn't show you whether there is more than one item loaded into HMA (the 64KB high memory area above 1MB). Currently, only those items specified by the operating system (such as DOS and buffers) can load into HMA. With MS-DOS 5.0, only one application can be loaded into HMA at a time. (An example of using "MEM /C" is given later in this AppNote.)

DOS 5.0 and Windows 3.1 EMM386 Drivers

There are some differences between DOS 5.0's EMM386.EXE and Windows 3.1's newer version of the program. For example, when I loaded the DOS version of EMM386.EXE, I started with 159KB of UMBs. However, DOS's EMM386.EXE takes 9,424 bytes of conventional memory.

Loading the Windows version of EMM386.EXE produced 156KB of UMBs, but the memory manager only takes 4,256 bytes of conventional memory. After weighing the differences, this amounts to only 2KB. But there are times when an additional 2KB is all you need to load a larger program into UMBs rather than conventional memory.

Windows 3.1's EMM386 is newer than the EMM386 from the first release of DOS 5.0, and that EMM386 is newer than Window 3.0's EMM386. (The DOS 5.0 update dated 11-11-91 contains the newer EMM386 released with Windows 3.1.) For the most part, you're better off using the newest version of EMM386. However, the original DOS 5.0's memory managers work a little better with NetWare Lite v1.0 and v1.1. The examples in this AppNote demonstrate the use of both EMM386 driver versions.

Sample CONFIG.SYS Files and Explanations

The following two examples of CONFIG.SYS files use MS-DOS 5.0. The first example uses the EMM386.EXE driver to access UMBs, while the second example uses the EMM386.EXE driver for expanded memory.

These examples show actual working files, but they don't cover all the possible parameters that you can use with the entries listed--that's what DOS manuals are for. Also, due to the wide variety of hardware configurations that are possible with PCs, the memory numbers you see on your workstations probably won't match these examples exactly. But the basic concepts behind the examples should help you in setting up your own machines.

Example 1: Using EMM386 to Access UMBs. Figure 1 is a listing of the first example CONFIG.SYS file. A brief explanation of each line is given after the listing.

Figure 1: This example CONFIG.SYS file places DOS into HMA and activates UMBs.

```
SHELL=C:\COMMAND.COM /E:300 /P
        FILES=45
        BUFFERS=30
        STACKS=9,128
        DEVICE=C:\DOS\HIMEM.SYS
        DEVICE=C:\DOS\EMM386.EXE /NOEMS
        DEVICEHIGH=C:\DOS\ANSI.SYS
        DOS=HIGH, UMB

SHELL=C:\COMMAND.COM /E:300 /P
```

This first line in the example CONFIG.SYS file tells DOS which command processor file to use (usually COMMAND.COM) and where to find it (the root directory of drive C in this example).

The /E:nnnn parameter allows you to set aside more space in memory for the DOS environment. The size of environment you'll need depends on factors such as whether you use color in your DOS prompt or whether you use Novell's MENU utility (both need more than the default of 256 bytes to work properly). You can also have environment space problems if you map a lot of search drives or run programs that use environment variables.

The environment can be expanded to a maximum of 32,768 bytes if necessary. You should use this option sparingly, because it does expand the size of the portion of DOS which remains in conventional memory. If you load DOS into HMA, you can expand your environment up to 24KB without increasing DOS in conventional memory. But that 24KB must include the BUFFERS entry as well (see explanation below).

The /P parameter causes COMMAND.COM to be loaded permanent (you can't type EXIT to close it).

FILES=45

The FILES entry is the maximum number of files (file handles) that can be open concurrently on the workstation. The number can range from 8 to 255. If the workstation runs programs that extend DOS's functionality, such as Windows 3.x or DESQview 386, you might need to increase this number to 55 or 60.

The FILES entry doesn't affect the number of network file handles that you set through the "FILE HANDLES = nn" command in the NET.CFG or SHELL.CFG file, except that the total of the two cannot exceed 253. If you are using the network to load Windows or DESQview 386, you'll need to leave room in the shell for the same amount of file handles. You can avoid a number of printing problems by matching the FILE HANDLES setting in the workstation's NET.CFG or SHELL.CFG file to the FILES entry in CONFIG.SYS (set them both to 55 or more).

BUFFERS=30

A buffer is a block of memory (cache) that temporarily holds disk reads and writes. The BUFFERS entry can be adjusted according to the amount of memory the workstation has to work from. Each buffer takes 528 bytes of memory. Setting the number of buffers between 15 and 25 (or higher) speeds up directory and subdirectory access. Unless the workstation uses the local hard disk a lot, you won't need to increase this entry.

If you load DOS into HMA, DOS will automatically take its buffers into HMA with it. If you keep the number of buffers set below 48, both DOS and buffers will fill HMA. There are situtations when you'll need to set the BUFFERS entry higher. For example, if you load Windows from the workstation's hard disk drive, run other applications, or access local data a lot, set the BUFFERS entry higher than the default (usually 15). The default varies according to how much memory the workstation contains--stations with 512K or greater default to 15 buffers.

One last note on buffers: If you are also using a disk caching program, such as MS DOS's SMARTDRV.SYS or Super PC-Kwik, you can get away with fewer buffers because disk caching performs the same kinds of disk read and write buffering functions. Also, the write-deferred capabilities of disk caching programs can increase performance by waiting until idle time to write to disk, as well as by being a cache store for disk reads.

STACKS=9,128

The STACKS entry is used (mainly by Windows) to allocate memory for processing hardware interrupts. The first number (9 in the example) displays how many stacks are available. This can be 0 or a number between 8 and 64. The second number (128 in the example) shows how many bytes of memory are allocated to each stack.

If the workstation is an IBM PC, PC-XT, or IBM PC-Portable, installing Windows will result in the Stacks entry showing 0, 0, which means MS-DOS allocates no stacks on the computer. On all other computers, the default setting is 9, 128. To save memory, you can try setting STACKS to 0, 0; if you find that certain programs tend to hang more often, you can reset STACKS to 9, 128. (You'll also see stack sizes of 256 in some of the examples shown here.)

DEVICE=C:\DOS\HIMEM.SYS

This entry and the following entry set up the extended memory manager that includes HMA (HIMEM.SYS) and access to UMBs as well as expanded memory conversion (EMM386). For DOS 5.0 and Windows 3.x, you must load HIMEM.SYS before EMM386.EXE. Other memory managers combine the functionality of HIMEM.SYS into their version of EMM386.EXE.

HIMEM.SYS and Windows' version of EMM386.EXE together take about 5KB of conventional memory

(HIMEM takes 1.2KB and EMM386 takes 4.1KB). DOS's earlier version of EMM386 version takes 9KB. Note that you don't use DEVICEHIGH= to load these drivers. You load them using DEVICE= and let them take care of their own placement in memory.

When you use HIMEM.SYS, you can later add the line "DOS=HIGH, UMB" to place DOS in the high memory area above 1MB. The UMB part of the command links DOS with the upper memory manager and ensures UMBs are accessible (see explanation below).

DEVICE=C:\DOS\EMM386.EXE /NOEMS

When you run EMM386.EXE, DOS displays on the screen how much upper memory is available for program placement. Depending on the type of BIOS, you may see as much as 159KB of upper memory blocks. For example, the BIOS in COMPAQ 386 and 486 computers offers up to 159KB, while American Megatrends Inc. (AMI) BIOS only offers up to 95KB of upper memory blocks. This is because AMI BIOS has a more extensive configuration capability, which takes away certain UMB regions.

If you use the upper memory block area for loading programs, either the "/RAM" or "/NOEMS" parameter is necessary:

- Use "/RAM" if you're running applications that use expanded memory
- Use "/NOEMS" if you want to access UMBs but not expanded memory

In our example, the "/NOEMS" parameter is used to prevent EMM386 from converting extended memory to expanded memory while still using upper memory blocks for TSRs.

EMM386.EXE comes with a number of other switches you can use with expanded memory. We will examine some of them in our next example. For the others, consult your DOS manual.

DEVICEHIGH=C:\DOS\ANSI.SYS

Once EMM386.EXE is in place, you can place other device drivers into upper memory rather than using conventional memory. Our example loads the ANSI.SYS file, which is used to add color to the DOS prompt. You can try loading other device drivers into UMBs, but you may have to experiment to determine if they remain stable enough to use.

DOS=HIGH, UMB

DOS can be placed either in upper memory or in the high memory area. To place DOS in HMA, you must load HIMEM.SYS and use DOS=HIGH in the CONFIG.SYS file. To use UMBs for holding TSRs, you must include "UMB" in the statement.

To place TSRs in UMBs, you must also have EMM386.EXE loaded using either the "/RAM" or "/NOEMS" parameter. You'll get more upper memory blocks with "/NOEMS" than with "/RAM" because the expanded memory conversion takes upper memory to create a 64KB page frame. This reduces the amount of upper memory left for relocating TSRs.

Example 2: Using EMM386 for Expanded Memory. The example CONFIG.SYS file listed in Figure 2 demonstrates the use of expanded memory. The workstation in this example has 12MB of memory and contains an Ethernet NIC, so no memory is lost to the network board (as is the case with Token-Ring adapters). **Figure 2: CONFIG.SYS file showing the use of EMM386 as an extended-to-expanded memory converter.**

```
FILES=35
        BUFFERS=15
        SHELL=C:\DOS\COMMAND.COM /P /E:200
        DEVICE=C:\DOS\HIMEM.SYS
        DEVICE=C:\DOS\EMM386.EXE 1184 /RAM i=C800-C9FF
```

x=CDFF-CFFF i=D000-EFFF
DEVICE=C:\DOS\RAMDRIVE.SYS 512 /A
INSTALL=C:\DOS\FASTOPEN.EXE C:=50 /X
DOS=UMB, HIGH

Note: The EMM386.EXE parameters should all be on one line.

For this example, we'll cover only those items that have changed from the first example.

DEVICE=C:\DOS\EMM386.EXE 1184 /RAM i=C800-C9FF
x=CDFF-CFFF i=D000-EFFF

For this implementation of the EMM386.EXE driver, the workstation is using expanded memory. The number 1184 indicates how much of the workstation's extended memory is converted to expanded memory. We included the /RAM parameter in order to use the upper memory blocks. That gives this particular workstation 92KB of available upper memory.

If we had used the /NOEMS parameter, it would have left 156KB of upper memory available. In the case of expanded memory, the difference includes the memory needed for setting up a 64KB page frame.

The i=C800-C9FF and "i=D000-EFFF" parameters tell EMM386 to include these two memory areas when creating UMBs. Be sure that any memory you include is not being used by video adapters or other programs. The "x=" parameter excludes portions of memory, such as CDFF-CFFF, that will be used by another program. If you have an area of memory which you designate in both include and exclue parameters, the exclude parameter overrides the include parameter.

Note: Areas excluded on the EMM386.EXE line in CONFIG.SYS do not need to be excluded again in the EMMEXCLUDE line of the Windows SYSTEM.INI file.

The EMM386.EXE driver has many other options for modifiying it. Figure 3 summarizes some of the more popular settings for EMM386.EXE.

Figure 3: Some of EMM386.EXE's configuration parameters.

Parameter	Description
on/off/auto	activates and deactivates the EMM386 device driver
w=on/off	activates and deactivates the Weitek coprocessor
/ram	gives access to expanded memory and UMBs
/noems	prevents access to expanded memory but allows access to UMBs
i=(range)	specifies segment address range for driver or RAM
x=(range)	prevents driver from using specified segment addresses
mx	specifies a page frame address--default page frame is D000-DFFF (see DOS manual for other predetermined ranges)
frame=address	directly specifies a page frame segment
/p(range)	specifies a page frame address

Read the DOS manual carefully when setting up expanded memory capabilities.

DEVICE=C:\DOS\RAMDRIVE.SYS 512 /A

In older versions of DOS, RAM disks were called virtual disks and you configured them through the

VDISK.SYS driver. RAMDRIVE.SYS works the same way as VDISK.SYS, with a few different parameters which you can find in the DOS manual.

For this example, the RAM drive is 512KB in size, and the "/A" parameter creates it in expanded memory. If you are using extended memory, use the "/E" parameter. If you don't use either "/A" or "/E," the RAM drive will come out of conventional memory, which defeats your purpose in optimizing memory usage (unless you need to juggle the order to get a larger program into UMBs). Other things you can set include the sector size and the number of file entries. The RAM drive will be assigned the next available drive letter.

INSTALL=C:\DOS\FASTOPEN.EXE C:=50 /X

The FASTOPEN utility caches frequently-opened local files by storing their names and starting sectors in memory. You use the INSTALL command instead of DEVICE= in CONFIG.SYS to run FASTOPEN.

Note: DOS lets you have up to 24 local (non-networked) partitions.

You designate the drive letter for each drive on which you want to run FASTOPEN, followed by the number of files you want it to keep track of. In this example, the workstation is keeping track of 50 files on drive C. Each file designation takes 48 bytes. The "/X" parameter loads the FASTOPEN cache into expanded memory, which also frees up conventional memory.

Again, there are a number of stipulations for using this utility (such as not running it with disk-compaction programs). Consult your DOS manual for details.

Sample AUTOEXEC.BAT Files and Explanations

The following are two AUTOEXEC.BAT files that work with the above CONFIG.SYS files. We'll continue with the same approach by explaining the first example, then covering only those entries in the second example that differ from the first.

Example 1: Loading NetWare TSRs High. Figure 4 is a listing of an AUTOEXEC.BAT file for a user who wants to load IPX and the NetWare shell (NETX) into upper memory blocks. **Figure 4: Example of an AUTOEXEC.BAT file that loads network TSRs into UMBs.**

```
ECHO OFF
        LH C:\WIN31\SMARTDRV C 1024
        PATH=C:\WIN31;C:\UTIL;C:\DOS;C:\
        SET TEMP=C:\WIN31\TEMP
        PROMPT $P$G
        LH DOSEDIT
        LH C:\MOUSE\MOUSE
        LH C:\IPX
        LH C:\NETX
        F:
        LOGIN SRD/ELIEBING

LH C:\WIN31\SMARTDRV C 1024
```

This file uses the LOADHIGH (LH) command to load TSRs into upper memory blocks. You can either write out LOADHIGH or use the abbreviated LH to invoke the command. In this case, the user is loading Microsoft's SMARTDRV disk caching program. Disk caching programs help Windows run a little better and SMARTDRV has been optimized for Windows.

The C parameter turns off deferred writes to disk, which is important only if you are running SMARTDRV for the C drive along with NetWare Lite. (See the section on "NetWare Lite and NetWare" for more information.)

The number 1024 designates the total size allocated to disk caching. Without such a specification,

SMARTDRV automatically takes up to half your extended memory for disk caching. You can also control how much upper memory SMARTDRV driver takes by setting the driver size. At the 1024 setting, SMARTDRV is 26.7KB in size; by controlling SMARTDRV's size, you may be able to place more TSRs into UMBs.

LH DOSEDIT through LH C:\NETX

Because the workstation in CONFIG.SYS Example 1 had 156KB of UMBs to use, all five of these programs preceded by LH loaded into upper memory blocks. To confirm that they did indeed load high, you can type "MEM /C" to see the final outcome of where everything went.

Once everything was loaded high, the user had 620KB of conventional memory to use for applications.

Example 2: Using ODI Drivers. For the second example, we'll look at an AUTOEXEC.BAT using Novell's ODI drivers instead of a dedicated IPX driver for network communications (see Figure 5). **Figure 5: This AUTOEXEC.BAT file loads ODI drivers for network communications.**

```
ECHO OFF
        PATH=C:\UTIL;C:\DOS;C:\WIN31;C:\
        SET TEMP=C:\WIN31\TEMP
        PROMPT $P$G
        LH C:\LSL
        LH C:\NE1000
        LH C:\IPXODI /A
        LH NETX
        F:
        LOGIN SE/BLOGGINS
```

LH C:\LSL through LH C:\IPXODI /A

In this example, user Bill is loading the Open Data Link (ODI) drivers, including the LSL (Link Support Layer), the network driver (in this case, for the NE1000 card), and IPXODI.

The /A parameter cuts down on the size of IPXODI by stripping out both the SPX capabilities and the network management capabilities of IPX. Be sure that you don't need either of these functions before using the /A parameter. Otherwise, you may not be able to run applications that use SPX for guaranteed communications.

You can also use the /D parameter, which allows for SPX, but takes out the network management capabilities.

Loading IPX high gives you about 17.5KB more conventional memory, and loading NETX high gives you about 48.8KB more conventional memory (depending on the version of NETX you load). Loading the ODI drivers high can result in a comparable savings, but can shrink IPXODI to 12KB if you use certain network board drivers (such as the NE1000) and the "/A" parameter.

The overall size of the ODI drivers depends on whether you fully load IPXODI or if you use network board drivers from third-party vendors, which can be larger than those released by Novell. Some third-party board drivers don't function well through a memory manager and must remain in conventional memory.

You can, however, load LSL and IPXODI into UMBs and then load the network driver into conventional memory, or any combination of UMB and conventional memory loading (depending on the network board dirver's capability). You may have to experiment to see if your third-party driver will work with a memory manager or not.

Using MEM /C to Display UMBs

You can use the MEM command with the /C option to see how your memory efforts are going. Figure 6 is a

sample printout of memory usage resulting from the CONFIG.SYS and AUTOEXEC.BAT files for Example 1. Adding "| MORE to the command lets you see the information a screenful at a time.

The MEM /C command shows three basic kinds of information. The Conventional Memory section shows which programs are currently running in conventional memory, while the Upper Memory" section shows how the workstation's upper memory blocks are being utilized.

The first number under "Size in Decimal" (for example, 15040 on the MSDOS line) is the size of the program in bytes. The number in parentheses (for example, 14.7K) is a calculated value in kilobytes. This second number is the result of dividing the first number by 1,024 (1KB), and then rounding to the nearest tenth. Dividing 15,040 bytes by 1,024 gives you 14.6875, or roughly 14.7KB. Using KB values makes it easier to deal with large sizes in the megabyte range.

The information at the bottom of the "MEM /C" display shows the combination of conventional and UMB memory, how large an executable program can load in conventional memory, and the largest block available in the UMB area. It also shows how much memory is extended or expanded, or both, depending on how you are using EMM386.EXE.

Figure 6: By using "MEM /C," you can see how successful your memory management efforts have been.

Conventional Memory :

Name	Size in Decimal		Size in Hex
MSDOS	15040	(14.7K)	3AC0
HIMEM	1072	(1.0K)	430
EMM386	4256	(4.2K)	10A0
COMMAND	2672	(2.6K)	A70
FREE	64	(0.1K)	40
FREE	632048	(617.2K)	9A4F0

Total FREE : 632112 (617.3K)

Upper Memory :

Name	Size in Decimal		Size in Hex
SYSTEM	167472	(163.5K)	28E30
ANSI	4192	(4.1K)	1060
SMARTDRV	26768	(26.1K)	6890
IPX	17936	(17.5K)	4610
NETX	49984	(48.8K)	C340
DOSEDIT	3088	(3.0K)	C10
MOUSE	17072	(16.7K)	42B0
FREE	64	(0.1K)	40
FREE	144	(0.1K)	90
FREE	1120	(1.1K)	460
FREE	39632	(38.7K)	9AD0

Total FREE : 40960 (40.0K)

Total bytes available to programs
(Conventional+Upper): 673072 (657.3K)
Largest executable program size : 630752 (616.0K)
Largest available upper memory block :39632 (38.7K)

```
11534336 bytes total contiguous extended memory
      0 bytes available contiguous extended memory
  10412032 bytes available XMS memory
MS-DOS resident in High Memory Area
```

In the "Upper Memory" section, you can see that DEVICEHIGH and LOADHIGH placed ANSI.SYS, SMARTDRV, IPX, NETX, DOSEDIT, and MOUSE into the upper memory blocks. By putting "DOS=HIGH" in the CONFIG.SYS file, you see the very last line stating "MS-DOS resident in High Memory Area." Since only one program can access the high memory area, DOS is a good candidate. As mentioned earlier, DOS also brings its buffers with it.

Sometimes EMM386.EXE places programs in a different order than how they were initially loaded. This is the memory manager's way of filling free space which is available because another program only partially filled an upper memory block area. In our example, both DOSEDIT and MOUSE were moved to a different block area.

This moving around doesn't affect TSRs that can be unloaded. You unload TSRs in the same order in which you load them, not as they appear when viewing them with the "MEM /C" command. For example, NETX can be unloaded by using the "U" switch after the program name, as in "NETX U." You can similarly unload IPXODI, but not dedicated IPX.COM.

However, if other TSRs are loaded after you load NETX, you'll have to unload them first before unloading the NetWare shell. Otherwise, you'll receive a message stating that other TSRs (or IPXODI, network drivers, or the LSL) are loaded above NETX, or that another program is sharing an interrupt with the program you are trying to unload. In either instance, you won't be able to unload the TSR.

286 Workstations--Get What You Can

Since MS-DOS's EMM386.EXE program is designed to work with 80386 microprocessors, most 286-based workstations aren't able to use it. Without using the HIMEM.SYS memory manager, a 286-based workstation can end up with 505KB of conventional memory after loading IPX and NETX, or as little as 448KB after loading NetWare Lite software. You can use HIMEM.SYS to load DOS into HMA (provided the workstation has at least 1MB of memory). By loading the minimum networking TSRs, you can have over 550KB of conventional memory for running applications in native NetWare, and over 530KB for NetWare Lite (without NLCACHE).

By adding a RAM disk, you can increase the workstation's overall performance by placing the most used programs into the RAM drive.

The CONFIG.SYS example in Figure 7 is for those 286-based computers that don't have a lot of extended memory, but incorporate the two ideas explained above.

Figure 7: For 286-based computers, you can use some simple memory management steps.

```
SHELL=C:\COMMAND.COM /P /E:300
        BUFFERS=30
        FILES=35
        DEVICE=C:\DOS5\HIMEM.SYS /MACHINE:8
        DEVICE=C:\DOS5\RAMDRIVE.SYS 384 /E
        DOS=HIGH
```

For 286-based workstations with only 1MB of memory, you can load HIMEM.SYS, then use the DOS=HIGH command to place DOS into high memory. This leaves you with an additional 48KB of conventional memory (depoending on how you have DOS configured).

If the 286 computer hangs when loading HIMEM.SYS, try adding the /MACHINE parameter after the

command. Consult your DOS manual for the available /MACHINE options to try.

286-based computers with 1MB of memory can designate the 384KB of extended memory to use for either a virtual RAM disk or cache memory. The above example creates a RAM disk using RAMDRIVE.SYS. This driver turns the 384KB of extended memory into a virtual memory disk drive and assigns the next available drive letter to the RAM drive. You can then copy the most frequently-used applications and utilities to the RAM drive, because accessing memory is much faster than accessing the hard disk drive or the network.

On a NetWare Lite installation, for example, you could copy COMMAND.COM into the RAM drive, then set the command specification (COMSPEC) to find the file at that drive letter. If the RAM disk is D, you would place the following commands in the AUTOEXEC.BAT file:

```
COPY COMMAND.COM D:
        SET COMSPEC=D:\COMMAND.COM
        SET PATH=D:\;%PATH%
```

The last line places the RAM disk first in the PATH statement (you can only use the %PATH% statement in a batch file—it doesn't work from the command line).

You can also place other applications and utilities that you use most often in the RAM drive. However, don't place data files in a RAM disk. In the event of a power failure, you will lose all your work (it's not saved to disk, only to memory). Instead, place applications in a RAM drive and leave data on a local disk or on a network drive.

Also, set applications to perform periodic backups of the data files you are working on. Then, if there is a power failure, simply copy the applications back to the RAM drive, pull up the data files, and you're back in business.

NetWare Lite and NetWare

Some sites run NetWare Lite to provide peer-to-peer connections under native NetWare. This setup can leave precious little overall RAM for running applications. Following is a sample CONFIG.SYS and two sample AUTOEXEC.BAT files that use this same CONFIG.SYS:

```
FILES=45
        BUFFERS=30
        SHELL=C:\COMMAND.COM /E:300 /P
        DEVICE=C:\DOS\HIMEM.SYS
        DEVICE=C:\DOS\EMM386.EXE /NOEMS
        DOS=HIGH, UMB
        STACKS=9,128
```

The first AUTOEXEC.BAT file in Figure 8 gives the user 592KB out of 640KB, with 156KB of UMBs.

Figure 8: Sample AUTOEXEC.BAT for loading both NetWare Lite and NetWare drivers.

```
PATH=C:\WINDOWS;C:\UTIL;C:\DOS;C:\HJWIN;C:\NWLITE
        SET COMSPEC=C:\COMMAND.COM
        SET TEMP=C:\WINDOWS\TEMP
        PROMPT $P$G
        C:\DOS\SHARE
        C:\IPX
        LOADHIGH SERVER
        LOADHIGH CLIENT
        LOADHIGH NETX
        LOADHIGH NLCACHEX.EXE  1729
        LOADHIGH DOSEDIT
```

```
M:
LOGIN SERVER1/DSMITH
```

In this example, the user has dedicated IPX instead of the ODI drivers. You can use either IPX or ODI with NetWare Lite, as well as with NetWare. Notice that both SHARE and IPX are loaded into conventional memory rather than in UMBs.

After a lot of experimenting and changing of LOADHIGH order, the above file allowed for SERVER (44KB), NETX (44KB), and NLCACHE (41KB) to load high, leaving only 1280 bytes left in the UMB area, and giving the user 592KB in which to run applications.

The size you make NLCACHE will also affect how much memory you have in the end. NetWare Lite's NLCACHE is a disk caching utility that lets you defer writes. (NLCACHE's write-deferred option has been thoroughly tested to run with NetWare Lite, in much the same way SMARTDRV has been optimized to run with Windows.)

As they are loading, certain programs tell the memory manager how much memory they need in order to load high. For example, NetWare Lite's SERVER.EXE program has EMM386.EXE look for a 64KB block of memory before loading high, even though the program itself only takes about 46KB to begin with. If EMM386.EXE can't find that large of a memory block, SERVER loads into conventional memory.

Because of this, it's sometimes best to load SERVER.EXE first. This is why NLCACHE and DOSEDIT are loaded last instead of first. The only problem with loading programs like DOSEDIT last is that you can't unload the TSR with an unload command. You would have to reboot the workstation if you wanted to unload another TSR that was loaded before DOSEDIT.

The second AUTOEXEC.BAT example in Figure 9 is for a more generic 386 computer with AMI BIOS. This workstation has 542KB of conventional memory, with 92KB of UMBs for loading NetWare Lite and NetWare high.

Figure 9: Sample AUTOEXEC.BAT for a workstation using AMI BIOS.

```
ECHO OFF
    PATH=C:\DOS5;C:\UTIL;C:\WIN31;C:\NWLITE;C:\
    SET TEMP=C:\WIN31\TEMP
    PROMPT $P$G
    SET COMSPEC=C:\COMMAND.COM
    LOADHIGH SHARE
    CLS
    CD \NWLITE
    ECHO NETWARE LITE AND NETWARE BOOT
    LSL
    NE2000.COM
    IPXODI /A
    LOADHIGH SERVER
    CLIENT
    LOADHIGH NLCACHE.EXE 1024 /B=9 /T=2 /O /X /Z /S
    NETX
    M:
    LOGIN SERVER2/NHAYES
```

Because this workstation has only 92KB of UMBs to work with, the ordering of TSRs becomes more important. You can sometimes load certain smaller TSRs in conventional memory to get a larger TSR into UMBs, resulting in more conventional memory in the end.

In this example, the user loads the mouse driver, the SHARE program, SERVER.EXE, and NLCACHE (caching 1MB of RAM) high. Everything else is loaded into conventional memory. This ordering leaves 576

bytes in UMB and gives the user 542KB of conventional memory to run applications.

A Working Example

To get a better feel for how to optimize conventional memory with MS-DOS 5.0, here's a before and after scenario. The workstation uses two batch files: one to load a normal NetWare connection, the other to load TCP/IP for a large backbone connection. The workstation is a Compaq 386 SystemPro with 8MB of RAM. The original CONFIG.SYS file is listed below.

```
DEVICE=C:\DOS\SETVER.EXE
        SHELL=C:\DOS\COMMAND.COM C:\DOS\ /E:2000 /P
        BUFFERS = 10
        FILES=30
        DEVICE=C:\DOS\HIMEM.SYS
        DEVICE = C:\DOS\EMM386.EXE NOEMS
        DOS=HIGH, UMB
        DEVICE=C:\WINDOWS\MOUSE.SYS /Y
        STACKS=9,256
```

This workstation's AUTOEXEC.BAT is also straightforward, loading simple commands and a command editor:

```
PATH C:\WINDOWS;C:\DOS
        SET TEMP=C:\WINDOWS\TEMP
        SET PCPLUS=C:\PCPLUS
        PROMPT $_$P$_$_$N$G
        CED -B2048,128,512,256,128,128
```

The network batch file looks like this:

```
IPX
        NETX
        F:
        LOGIN
```

The TCP/IP batch file contains the following commands:

```
CD TCP
        LSL
        NE2000
        IPXODI
        NETX
        TCPIP
        TELAPI
        CD ..
        F:
        LOGIN
```

The initial setup does not utilize the EMM386.EXE memory manager, only HIMEM.SYS to load DOS into HMA. Using a shareware BOOT program (discussed later), we reordered the CONFIG.SYS file as shown in Figure 10.

Figure 10: The reordered CONFIG.SYS file.

```
SHELL=C:\DOS\COMMAND.COM C:\DOS\ /E:1000 /P
        BUFFERS = 10
        FILES=35
```

```
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\EMM386.EXE /NOEMS
DOS=HIGH, UMB
DEVICEHIGH=C:\WINDOWS\MOUSE.SYS /Y
DEVICEHIGH=C:\DOS\SETVER.EXE
STACKS=9,256
```

The CONFIG.SYS was reordered to allow the mouse driver and SETVER to be loaded high. Again, you can use DEVICEHIGH after the EMM386.EXE driver has been loaded. The environment size was adjusted down to 1,000 bytes. Few other changes were really necessary.

In the new AUTOEXEC.BAT, we simply placed all commands in UMBs via the LOADHIGH command (see Figure 11).

Figure 11: Modified AUTOEXEC.BAT for the normal network connection.

```
PATH C:\WINDOWS;C:\DOS;C:\UTIL
    SET TEMP=C:\WINDOWS\TEMP
    SET PCPLUS=C:\PCPLUS
    PROMPT $_$P$_$_$N$G
    LH CED -B2048,128,512,256,128,128
    LH IPX
    LH NETX
    F:
    LOGIN
```

This file gave the workstation 615KB of conventional memory to work with, and 58KB of UMBs for loading other TSRs should the need arise. Compare this to the 512KB of conventional memory available before using the EMM386 memory manager and UMBs.

Figure 12 shows the AUTOEXEC.BAT as modified for loading the TCP/IP-related software.

Figure 12: The modified AUTOEXEC.BAT for the TCP/IP connection.

```
PATH C:\WINDOWS;C:\DOS;C:\UTIL
    SET TEMP=C:\WINDOWS\TEMP
    SET PCPLUS=C:\PCPLUS
    PROMPT $_$P$_$_$N$G
    LH CED -B2048,128,512,256,128,128
    CD TCP
    LH LSL
    LH NE2000
    LH IPXODI
    LH NETX
    TCPIP
    LH TELAPI
    CD ..
    F:
    LOGIN
```

In this AUTOEXEC.BAT, everything was loaded high except the TCPIP command. Like NetWare Lite's SERVER.EXE program, the TCPIP.EXE program also has code in it to have the memory manager look for space that is greater than its actual size. In repeated tries, the TCPIP program wouldn't load high, but the TELAPI program would, even though the TELAPI program is slightly larger in memory than the TCPIP program. (Neither TCPIP nor TELAPI has a /U unload parameter, so the workstation must be rebooted to unload TCP/IP support.)

Even with TCPIP loading in conventional memory, the workstation had 595KB of conventional memory to work with. When compared to 439KB of conventional memory before utilizing the memory manager and UMBs, the extra memory can be very beneficial.

Tips Along the Way

In working to get the most mileage out of MS-DOS's memory managers, I've found a few pointers that might be helpful:

- Before you begin, create a working boot diskette of your present workstation environment.
- Obtain a boot file organizer.
- Determine the minimum amount of memory your applications need.
- Experiment with the load order.
- Use configuration parameters to optimize memory use.

This section also tells how to resolve the Packed file is corrupt error message, which is related to memory optimization.

Create a Bootable Diskette

Before changing any settings, be sure to create a bootable DOS diskette containing your present DOS environment settings, CONFIG.SYS, and AUTOEXEC.BAT so you can always return to the original setup and fix any mis-steps.

Once you format a diskette with the FORMAT /S command, use XCOPY to copy over all files in your boot directory. If you call programs from other directories, include those directories and programs as well.

Obtain a Boot File Organizer

The best sources for boot file organizers are shareware providers such as The Software Labs and reputable bulletin board services. BOOT.SYS by Hans Salvisberg and BOOT.EXE by Stephen C. Kick are two good examples of shareware programs that allow you to have multiple CONFIG.SYS and AUTOEXEC.BAT files in a BOOT.CFG file. You can then label each type of configuration (such as Local, NetWare with Lite, NetWare, and so on) and select the one you want to use when you reboot the machine.

Determine the Amount of Memory Your Apps Need

When working with memory managers that come with DOS, you may end up experimenting with TSRs for quite some time. If you are managing a number of users, this can be very time consuming.

A better approach is to determine the minimum amount of conventional memory your applications can live with. For example, some programs need up to 525KB to run their basic offerings, and any additional memory lets you use more program options. If you run Windows, you also have that overhead to figure into the equation. You'll also want the DOS windows to be as large as you can get them.

Once you have determined such a number, such as 560KB, set up a generic CONFIG.SYS and AUTOEXEC.BAT that will reach that number. This will cut down on 80% of your memory management workload and keep most users happy.

For the other 20% (which will probably take 80% of your time), set up a strategy for handling:

- Common applications

- Common communications protocols, such as IPX or TCP/IP
- Certain applications and protocols together
- Creating separate CONFIG.SYS and AUTOEXEC.BAT for different environments with a program like BOOT

You might find that DOS 5.0's memory managers don't provide enough flexibility to handle all the nuances of certain configurations and still be able to run applications properly. When you reach this point, it's time to move to a third-party memory management program, or to OS/2 v2.0.

Remember, not all BIOSes are created equal when it comes to how many UMBs you have to work with. Many of the BIOSes used with Intel 80386 and 80486 microprocessors offer up to 159KB of UMB space, while BIOSes such as AMI's only offers up to 95KB. This is because AMI's extensive diagnostic capabilities are built into the BIOS. While these capabilities are marvelous for initial installation and setup, they do take away from the UMB space that memory managers use.

Also, you need to consider peripheral hardware when figuring the amount of UMBs you have to work with. For example, video adapters take away from UMBs (as discussed in Managing Memory in a DOS Workstation: Part 1 in the August 1992 NetWare Application Notes). A CD-ROM driver can take a chunk of UMBs for its configuration. IOMEGA's Bernoulli box also uses conventional or UMBs for its configuration.

But don't let this discourage you. Memory managers such as Quarterdeck's QEMM386 handle most of these hardware needs quite well, giving you plenty of UMB support. QEMM386 and similar programs will be covered in more detail in a future AppNote.

Experiment with Load Order

Determining how to best place TSRs into UMBs can get tricky, and you can spend a lot of time fussing with the loading order. Here are a few rules of thumb to work from.

In the CONFIG.SYS file:

- Load HIMEM.SYS before EMM386.EXE.
- Use DEVICEHIGH= after loading EMM386.EXE.
- Use DOS=HIGH to load DOS into HMA.
- Use DOS=HIGH, UMB to link UMBs to conventional memory.

In the AUTOEXEC.BAT file:

- Load TSRs that can't unload before those that can
- Load NetWare TSRs in the proper order:

  For NetWare, IPX then NETX

  Or LSL, network driver, IPXODI, NETX
- Load NetWare Lite TSRs in proper order:

  LSL

  network driver

  IPXODI

SERVER

CLIENT

- NLCACHE can load wherever it doesn't hinder loading SERVER
- Remember NetWare Lite's SERVER memory requirements

You can sometimes place smaller TSRs in conventional memory, so that you are able to load a larger TSR into UMBs. The amount of memory the two smaller TSRs take should be less than the larger TSR. This experimenting can take most of your time, and while it often becomes a challenge to get the most you can, don't lose sleep over a couple of kilobytes!

You have greater flexibility loading and unloading TSRs from the AUTOEXEC.BAT than from the CONFIG.SYS. (This includes cache drivers and the mouse driver.) When you make changes to the CONFIG.SYS file, you must reboot the workstation before the changes take effect. If you change the AUTOEXEC.BAT file, you can simply run AUTOEXEC again. You might need to unload some TSRs first, but you do have greater felxibility with the AUTOEXEC.BAT file.

Use the MEM /C command to see how your UMB management is working. Add "|MORE to the command so you can examine one screenful of information at a time. The most useful information from this command is found in the Conventional Memory and Upper Memory sections. These screens show you which TSRs made it to the UMB area and which ones didn't. You can use this information to decide which smaller TSR to load in conventional memory. For example, sometimes loading ODI's LSL and network driver in conventional memory may allow you to load NETX into UMBs.

Use Configuration Parameters to Optimize Memory Usage

By changing settings in DOS, IPX or IPXODI, NETX, SERVER, and other programs, you can change their load size. You change the load size of DOS through the BUFFERS, FILES, STACKS, environment variables, and other CONFIG.SYS parameters.

IPX, ODI, TCPIP, and NETX are all affected by the settings placed in the NET.CFG file, which can ultimately expand the size of each of these files. Also, each version of IPX and NETX is a little larger or smaller than its predecessors.

The load size of utilities like NLCACHE, SMARTDRV, and RAMDRIVE is influenced by how you set them up at installation. You can also configure NetWare Lite's SERVER.EXE to be larger than its initial size; however, it won't get any larger than 64KB (which is why it looks for that much memory when it initially loads).

Because of the dynamic nature of file configurations, you'll need to work with the numbers you see on each individual workstation, not the example numbers in this AppNote. The examples reflect how specific systems were set up; your workstations may show very different numbers.

Resolving the Packed file is corrupt Error

After maximizing the amount of available conventional memory to more than 615KB, you might see a message that says Packed file is corrupt when you try to run certain programs. These programs were compiled with a certain type of compiler that looks for a program to be loaded in the first 20KB of conventional memory. When they don't find one there, they return a "Packed file is corrupt" message and won't load.

You can easily solve this problem by pulling one of the workstation's programs that is larger than 20KB, such as NETX, back down into conventional memory instead of loading it high. Loading NETX in conventional memory drops the available memory to around 590KB. In my research, I have found that the

problem usually goes away at about 607KB of conventional memory, but you may want to experiment because it does depend on the application.

Conclusion

Microsoft's memory managers can provide enough conventional memory for most users. Since they come with MS-DOS 5.0 and Windows 3.1, you save money by trying them first. However, some users are pushing the limits of these memory managers that come with DOS. For those users, we'll look at other memory management solutions in further AppNotes in this series.

The next AppNote on memory managers will cover DR-DOS 6.0's memory management capabilities, as well as using Novell's EMSNETX and XMSNETX shells in conjunction with memory management.