CHAPTER 5:
DISK MANAGEMENT

Boot Disks

In order for a floppy diskette or a hard disk partition to be bootable under PC-DOS, the kernel files must be the first files on the disk and must have the "hidden" and "system" attributes. Under PC-MOS, the kernel file, $$MOS.SYS, can be copied onto the root directory of the boot disk just as any other file. No special position is required and the hidden and system attributes need not be used. This greatly simplifies the making of a floppy boot disk which already has other files on it.

In order to make a bootable disk with MOS, the following is required:

1. Execute the command MSYS A: to write a boot sector on the diskette.

2. Copy the files $$MOS.SYS, $$SHELL.SYS and COMMAND.COM to the root directory of the floppy diskette.

3. Create suitable CONFIG.SYS and AUTOEXEC.BAT files.

4. Copy any of MOS's system utilities and device drivers into the root directory or a subdirectory of your choosing.

MOS may even be booted from a 360K floppy disk by not including every system utility program. Even when all drivers and utilities are loaded onto a hard disk the disk space required for a MOS system is quite modest.

Setting Up the Cache

Most operating systems use some type of disk buffering to improve performance. In PC-DOS, the CONFIG.SYS statement BUFFERS=.. is used to define how much memory to use for disk buffering. Generally the more buffers used the better, up to the limit of how much memory can reasonably be allocated to such a purpose. One consideration with large buffers in PC-DOS, however, is that the size of the TPA is reduced (see the PC-DOS memory map in Chapter 1).

DOS uses these buffers during disk read operations to store disk information in RAM. Entire disk sectors are stored even when the actual request was for less data. When a read is then done for the next data in consecutive order, if it is already available in the buffer, the time that would be incurred for a another physical disk read operation is avoided. Thus buffers are generally the most useful in the case of sequential disk accesses.

Another type of disk buffering is called caching. This buffering of disk data is done at a lower level than the "buffers" method, and is a more sophisticated approach. Although PC-DOS is not normally shipped with a cache driver included, many third party products are available. When such a driver is used under PC-DOS, the high level buffers (from CONFIG.SYS "BUFFERS=") are still used but their size may be reduced.

Prior to release 2.30 of PC-MOS, a separate caching device driver, $CACHE.SYS, was provided which was used in addition to the higher level "buffers" method of disk buffering. With release 2.30, the disk buffering method was enhanced by combining the low and high level buffering into one new type of caching subsystem.

Since this is all managed within the operating system kernel itself, no external device driver is needed and the overall disk system performance much is greater since the number of processes involved have been reduced and centralized.

The caching system MOS used prior to 2.30, when an external $CACHE.SYS device driver was loaded, was what is known as a "write through" cache. When a write to disk operation was done, although a copy of the data was put into the cache it was also written immediately to the physical disk media. Caching a copy of this data would speed up any future read accesses to this same data but, in terms of the immediate performance of the write operation, caching provided no benefit.

One of the enhancements made to the disk buffering system with release 2.30 was an option to support the caching of write operations. This is commonly referred to as postponed write. When a write operation is done where the postponed write feature is used, the data from a write operation is held in RAM for a certain length of time before actually being written to the physical disk drive.

Using postponed write is somewhat like having a RAMdisk in place which automatically echoes its data to the physical disk. When an application does a write operation, the instantaneous speed advantage of a RAMdisk is achieved. Then, after a short delay, a physical disk access occurs and all postponed disk writes are updated in one operation. Optimization is achieved by reducing repetitive disk head movement.

Working with an operating system in which postponed write caching is being used can take some getting used to. For example, when you issue a COPY command, the actual physical disk write may not even begin until after you have received the next command prompt and have begun typing your next command. If your COPY command involves a floppy disk drive which has an open drive door, the critical error window will also pop up after a similar delay.

With the performance benefits of postponed write caching come additional caveats as well. Removing a floppy disk from a drive immediately after issuing a COPY command and seeing the following command prompt appear can result in an incomplete file transfer. A critical error condition will be generated in this case. To prevent this possibility, always wait until the drive access light goes out before removing a floppy.

Powering a computer down or rebooting it must also be done with a consciousness of the state of the caching system. If a power down or reboot is done while data is still being held in the cache, waiting to be written, incomplete file updates will result. The use of an un-interruptable power supply should be strongly considered when postponed write caching is involved since you otherwise have no recourse for the effects of an unexpected brownout or blackout.

In the case where an intentional powering down or reboot is to be done, a safe shutdown may be made through the use of the MOSADM CACHE OFF command. Issuing this command forces any and all cache data waiting to be written to be flushed out of the cache and onto the disk. From this point on, until a MOSADM CACHE ON command is issued, no postponed writes will be done.

Note that an additional consideration, when making a deliberate shutdown of a system, is to force any data waiting in a print spool buffer to be written out to the disk. This is done through the use of the pop up menu provided by the SPOOL.COM program.

If the heightened awareness required to operate a system with postpone write caching is not deemed to be worth the performance benefit, please note this feature of the caching subsystem is an option. Through the use of appropriate cache configuration parameters, this delayed write effect may be disabled in favor of a direct, write-through, type of operation. This will be covered in more detail later on in this chapter.

In addition to the performance benefits, another significant difference between the "buffers" method and MOS's disk caching is in how the memory involved is allocated. Providing that memory management is being used, only a small window of memory, typically only 4K, must be allocated out of the first Megabyte for access to the cache data. Therefore, declaring that you want a large cache of 1024K for instance, does not reduce the available base memory to a large degree. You must, of course, have that much extended memory available for cache use.

Cache Sizing and Tuning

As with most things, there is no absolute best situation, just a range of different compromises. For example, you can assign more memory to the cache to improve its performance but only up to a point. There is a point of diminishing returns based on both the response time improvement and the cost of installing more extended memory. Likewise, using a longer timing factor with the postponed write feature can improve performance but only with the increased risk of data loss in the event of a power failure.

There are a number of different variables involved in the fine tuning of MOS's caching subsystem. Some of these variables are set once, and then don't change unless you edit your CONFIG.SYS file and reboot. These "static" parameters consist of:

1. The total amount of memory to reserve for the cache.

2. The size of each cache buffer unit.

3. A pair of timing parameters which control the postponed write feature.

The remaining factor involved in overall caching performance is much more difficult to quantify. This is the type of disk activity which is occurring in the system. In a multiuser environment, where all users are sharing one or more hard disks, a wide variety of disk operations is likely. Given this situation, only general guidelines can be given regarding the tuning of a cache. The best method involves a mix of educated guesses coupled with timing tests through the course of typical types of disk operations for your system.

The last write timer, LASTW, acts as a minimum guarantee factor -- controlling the minimum amount of time which can pass before a physical update is made. Conversely, the first write timer, FIRSTW, acts as a maximum guarantee of the longest time period before the cache is flushed. Since it makes no sense to have LASTW greater than FIRSTW, the only case where FIRSTW will come into play is when a series of short bursts of disk activity prevents LASTW from ever expiring. Basically, LASTW is the main timer and FIRSTW is an insurance policy.

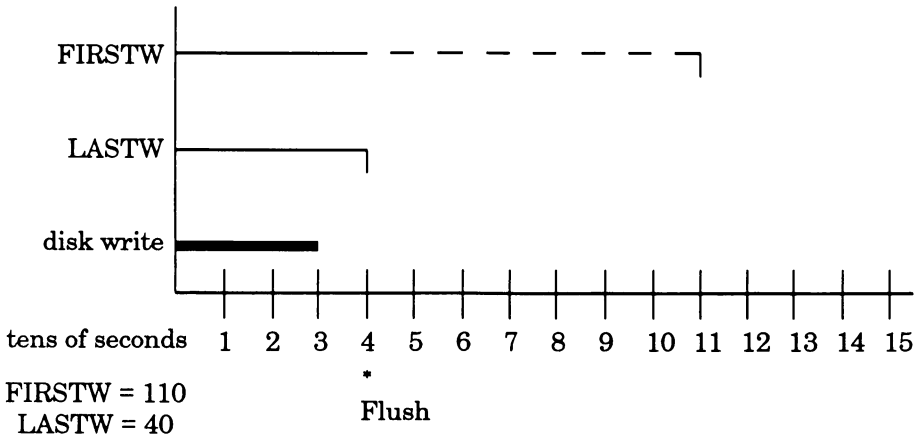


Figure 5 - 1

A time line chart showing what happens when last timer expires before first timer.

Figure 5-1 shows a disk write operation which lasts for 30 seconds. As soon as the write operation begins, FIRSTW starts counting for its duration of 110 seconds. Since LASTW expires before FIRSTW does, the data in the cache is flushed (written) to the disk at the 40 second mark.

Note that from this point on, there is no need for FIRSTW to keep on counting so it is turned off. As you can see, any write operations which are shorter than the value of LASTW will be flushed when LASTW expires without FIRSTW coming into play.

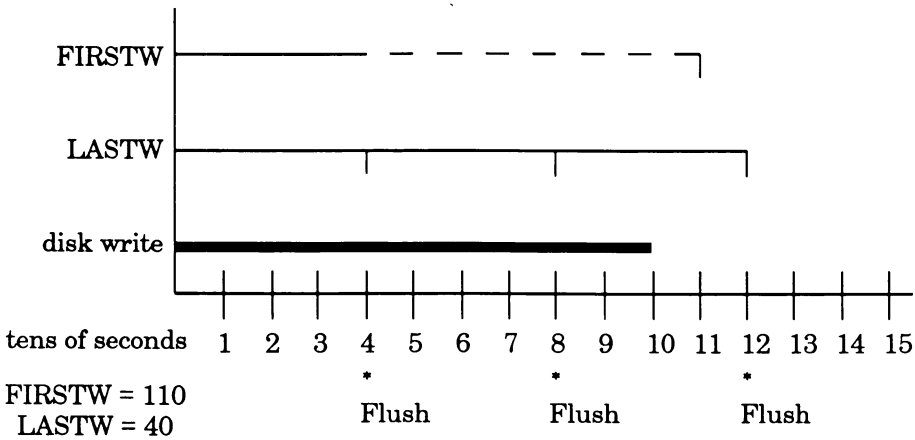


Figure 5 - 2

A time line chart showing what happens when a long period of disk activity occurs.

In figure 5-2, where a longer burst of disk activity occurs, the first flush to the physical disk occurs at the 40 second mark. At this point, since the disk operation is still in effect, LASTW restarts its countdown. This results in another flush at the 80 second mark and another restart of LASTW. Since the first flush turned off FIRSTW, it is up to the final expiration of LASTW, at 120 seconds, to bring the physical disk into full synchronization with the cache. Again, as in the case above, LASTW is the primary control factor.

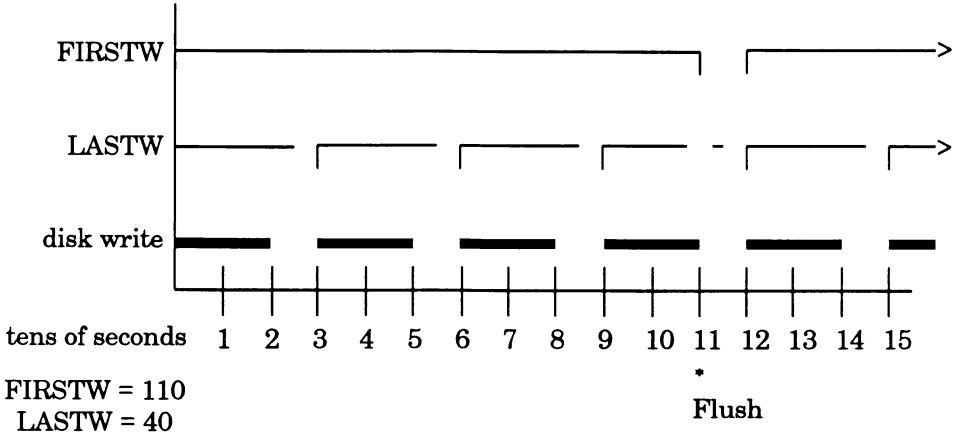


Figure 5 - 3

A time line chart showing what happens when a series of short disk writes occur.

In the case where a series of disk write operations occur in short bursts, LASTW may never get a chance to expire. As shown in figure 5-3, each time a new write operation occurs, LASTW is restarted. Since the countdown of FIRSTW has not been interrupted, when it finally expires at the 110 second mark, the cache will be flushed. If this pattern of disk writes continues, FIRSTW will restart (along with LASTW) when the next burst of disk activity occurs. This is what is meant by referring to FIRSTW as an insurance policy.

Disk Maintenance

The PC-MOS command set contains a number of features which can be of aid in disk maintenance chores. The DIR command supports options to sort the directory listing by one of four different keys:

dir /sn	alphabetize by file name
dir /se	alphabetize by file extension
dir /sd	sort by date, in oldest to newest order
dir /ss	sort by size, in smallest to largest order

DIR also supports a /D option which will list only the subdirectories within a given directory. In addition, examining the overall directory structure of an entire disk volume, or a selected portion of it can be done easily with MOS's DIRMAP utility. When you wish to review which files have their archive bits set, to see how current your backups are, run DIRMAP with its /A option. This will list only files which have been created or modified since the last backup operation. (Note that with some backup utilities, a "differential" backup will not reset the archive bit).

When it's time to clean up a directory, use the ERASE command, which may also be activated through the name DELETE or DEL. The /V option is available to make MOS prompt you before deleting each file individually. For example, if, out of a group of twenty files with the .DOC extension, you wish to delete eight files, you might find using DEL *.DOC /V easier than entering eight explicit DEL commands.

Listing 5-4

The XDEL.BAT batch file - for deleting a group of files:

```
:loop
if (%1) == ( ) abort
echo deleting %1
del %1
next
goto loop
```

In the case where a number of different types of files needs to be cleaned up, a batch file such as XDEL.BAT can be very useful (see Listing 5-4). The batch process uses the NEXT command in a loop structure to allow a variable number of command line parameters to be used. You may use DIR/W to produce a concise list of files and then enter XDEL followed by the names of each file to be deleted. Wildcards are permissible.

Another useful command line option available with ERASE (also DELETE and DEL) is /Y. Normally, when you attempt to erase all files within a directory with DEL *.*, you are prompted for verification. Using the command form DEL *.* /Y will prevent this prompt from occurring. Be careful with this operand. When the optional /Y operand is included, MOS assumes you know what you are doing.

Listing 5-5

ZAPDIR.BAT can simplify the removal of a directory:

```
del %1 /y
rd %1
```

Listing 5-5 shows an application of DEL's /Y option in a batch file called ZAPDIR.BAT. This utility would be used with the name of a directory as the command line parameter. Note that this simple scale version will only work for directories which don't contain any child directories.

Navigating around on a hard disk with many directories can become tedious. With statements like CD \SALES\HISTORY\JANUARY the keyboard gymnastics required to hit the backslash key can even become annoying. This is even more troublesome given that six different keyboard makers will locate the backslash key in six different places.

The batch file C.BAT (Listing 5-6) is one approach towards simplifying subdirectory navigation. To move to the root directory, simply enter C and press return. To move to the sales history directory referred to above, enter C SALES HISTORY JANUARY. Typing in a command with spaces instead of backslashes is much faster. Being large, the space bar is easy to hit, and it is still in the same place on every keyboard.

If your hard disk is divided into multiple volumes, you could create a copy of C.BAT for each volume (e.g. D.BAT for the D: drive) and edit the first command in the batch file to match the drive. In addition, using this batch file with a "?" as a parameter will provide a review of the child directories in the directory you've just changed to. For example, D ACCNTG ? will make D: the current drive, \ACCNTG the current directory, and list any subdirectories which exist within D:\ACCNTG.

Listing 5-6

This batch file, C.BAT, can simplify the task of navigating through a disk's directory structure.

```
c:
if (%1) == ( ) goto noparm
if %1 == ? goto showroot
cd \%1
if (%2) == ( ) abort
if %2 == ? goto showdirs
cd %2
if (%3) == ( ) abort
if %3 == ? goto showdirs
cd %3
if (%4) == ( ) abort
if %4 == ? goto showdirs
cd %4
abort
:showroot
cd \
:showdirs
dir /d
abort
:noparm
cd \
```

You may notice that C.BAT will not work for directories more than four levels deep. This batch file could have been constructed using a loop structure and the NEXT command to make it more universal. However, the method used here is much faster since no branching will be required for the most common case where a change is made to some other directory besides the root, and the "?" is not used. If you work with directory levels deeper than four, add more sections to C.BAT.

As an extension of this shortcut to directory navigation, make a unique batch file for any directories which you need to switch to often. If the directory in which you operate the most is named C:\WORK, then you could create CW.BAT which would contain the following statements:

```
C:
CD \WORK
```

By placing such batch files in one directory which is within your command search path, no matter where you are on the hard disk, entering CW will immediately bring you to your work directory.

Renaming Files and Directories

MOS's RENAME (or REN) command has several enhancements to aid in disk management. Just as with PC-DOS, of course, REN may be used to change the name of an individual file or a group of files. In addition, under MOS, REN will also allow subdirectories to be renamed. For example, issuing REN WORK WORK00 will work exactly as you would expect it should. There's no need to create a new directory, copy over all the files from the old directory, delete all the files from the original directory and then remove the original directory.

Another powerful feature of REN is its ability to move a file from one place to another within a disk volume. This eliminates having to copy a file from its current position on the disk to a new location and then deleting the original copy. The /M option is used to tell REN that you wish to move a file or group of files.

Using the command REN \XYZ.DOC \WORK/M is superior to a the method of copying and deleting a file not only because its one command instead of two, but also since the only actual change that needs to be made on the disk is to update the directory entry for XYZ.DOC. Therefore, moving XYZ.DOC will take only a second even if the file is several megabytes in size.

Listing 5-7

MOVETO.BAT and MOVE2.BAT work together to simplify the task of moving one file or a diverse group of files from one place to another within a disk volume:

MOVETO.BAT

```
if (%2) == ( ) goto parmerror
set mvto=%1
endslash %1
if errorlevel 2 goto error
if errorlevel 1 set mvto=%mvto%\
next
:loop
if (%1) == ( ) goto alldone
for %%a in (%1) do move2 %%a %mvto%
next
goto loop
:error
echo.
echo parameter error in ENDSLASH.EXE
:alldone
set mvto=
abort
:parmerror
echo. must have at least two parameters
```

```
MOVE2.BAT

echo moving %1 to %2
if exist %2%1 del %2%1
ren/m %1 %2*.*
filemode +a %2%1
```

Listing 5-7 shows how a loop structure can be used with REN/M to move a diverse group of files to a new directory. Actually, this utility is comprised of a pair of batch files and one additional utility program called ENDSLASH.EXE. The main batch file, MOVETO.BAT, is responsible for processing the command line parameter list and passing the information to MOVE2.BAT. This second batch file is where the actual file moving is done. Note that unlike the COPY command, using REN/M will not overwrite an existing copy of a file. This is ensured by the second statement in MOVE2.BAT.

```
MOVETO \NEWFILES XDEL.BAT *.SYS MODEM.DOC
```

The syntax for MOVETO.BAT is the same as for COPYTO, as described above. The first parameter is the destination directory and all following parameters are regarded as source file specifications. One important difference is that, unlike COPYTO.BAT, MOVETO.BAT can only be used where the source and destination are within the same disk volume. For the case where you need to move a group of files from on volume to another, you could use COPYTO.BAT followed by XDEL.BAT (e.g. recall the COPYTO command line with the up arrow and edit the start of the line to delete the destination and replace COPYTO with XDEL).

Due to the manner in which parameters must be treated within MOVETO.BAT, and its companion batch file, MOVE2.BAT, a simple utility program is used to provide some special parameter analysis. ENDSLASH.EXE is a small program which is supplied with one command line parameter and returns an ERRORLEVEL based on tests which are done on that parameter.

In MOVETO.BAT the initial value of the replaceable parameter %1 holds the value of the destination directory. In order to handle all cases properly, MOVETO.BAT must insure that the %1 string ends with a backslash, so ENDSLASH.EXE is used to perform this test. When ENDSLASH.EXE finds no backslash at the end of the %1 string, it returns an ERRORLEVEL 1 which causes MOVETO.BAT to use the following statement to append a backslash onto the copy of the %1 string as stored in the environment variable MVTO:

```
SET MVTO=%MVTO%\
```

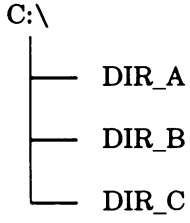
The companion disk to this manual includes a copy of ENDSLASH.EXE, as well as copies of MOVETO.BAT and MOVE2.BAT. If you wish, you can also make your own copy. An ERRORLEVEL 2 is returned by ENDSLASH to signal that no parameter was supplied.

The secondary batch file, MOVE2.BAT, makes use of the FILEMODE command to force the archive bit of the file just moved to be set. This is done to insure that the next backup operation will catch this file in its new location. Since renaming a file doesn't change its contents, REN doesn't affect the archive bit. For both the FILEMODE and ENDSLASH commands, it would be best, for the sake of execution speed, to include the full directory name before the command name.

One final note concerning MOVETO.BAT is the use of the FOR IN DO command. In the event that the string associated with %1 at the time this FOR statement is executed contains any wildcards, the FOR command will perform a directory search and make one call to MOVE2.BAT for each matching file found. Note that the value of %1 at this point is no longer the destination directory name, since the NEXT command is used to shift the values associated with the replaceable parameters. At this point, %1 will hold one of the source file specifications.

Pruning and Grafting

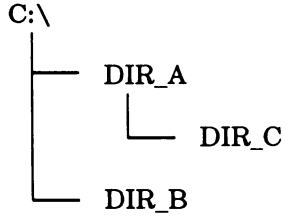
The last application of the REN command for disk maintenance involves pruning part of a volume's directory tree structure off of one place and grafting it onto another place. Given the following directory structure:



Issuing the command:

```
REN \DIR_C \DIR_A\DIR_C /M
```

would produce:



Any and all files within DIR_C are still within DIR_C within its new location and issuing the command: REN \DIR_A\DIR_C \DIR_C /M would restore the directory structure to its original configuration. Note that complete path names must be used for both parameters and that it is possible to change the name of the directory being moved within the same command.

While there are a few third party disk utilities available which perform prune and graft functions of whole directories as MOS's REN /M does, these should not be used on a MOS system unless no other tasks exist other than the foreground. REN /M does extra processing and performs extra checks to insure that no other tasks have any dependencies on any directories which would be affected by such a wholesale restructuring of a disk's directory tree. Third party utilities are not likely to have access to that level of information.

RAMdisks

PC-MOS includes a device driver named $RAMDISK.SYS which supports the emulation of a disk drive using extended memory. The speed advantages of such a pseudo disk drive can be a great benefit in speeding up large complex batch processes or in facilitating a form of intertask communications. If you plan to use $RAMDISK.SYS to speed up your batch file operations, you could have your AUTOEXEC.BAT copy your library of batch files to the RAM disk when the foreground boots.

Note that the use of a RAM disk must be considered when estimating your extended memory needs. In addition, a spot in FREEMEM must be reserved for use by $RAMDISK.SYS. This reserved area must be 16K in size and, like all memory windows, it must be aligned on a 4K boundary.

As with all hardware or software entities which use any of MOS's FREEMEM space, it's always best to position them so that the remaining space, which is available for MOS to use for its overhead, is one contiguous area.

EXCEPT and ONLY

The best way to understand commands like EXCEPT and ONLY is through examples. Say for instance that you have a directory on your disk which contains a large number of files and you must find a certain file. Say also that you can't quite remember the exact name of this file but you'll recognize it when you see it.

The problem is that you are now faced with scanning through several hundred files in order to find the one you're interested in. What is needed is a way to narrow the field a bit. If you know that the file you're looking for doesn't use a file extension of the type .TXT, .DOC, .REP, .COM, .EXE, .BAT or .LST and there are many files in the directory which do use these extensions, then using the EXCEPT command as follows could greatly aid your search:

```
EXCEPT (*.TXT *.DOC *.REP *.COM *.EXE *.BAT *.LST) DO DIR /W
```

The batch file ALLBUT.BAT (see Listing 5-8) can simplify this task even further by reducing the details involved in building the command line for the EXCEPT command. This batch file is an interesting combination of a loop structure which processes a variable number of command line parameters and the concatenation of data into an environment variable. The parameters you specify after ALLBUT are the extensions of the file groups which are to be excluded from a directory command; thus the name "allbut".

Entering the following will result in the formation of the same EXCEPT command line as shown above:

```
ALLBUT TXT DOC REP COM EXE BAT LST
```

Listing 5-8

ALLBUT.BAT can simplify searching for files in a large directory:

```
if %1! == ! abort
set allbut=*.%1
:loop
next
if !%1 == ! goto allparms
set allbut=*.%1,%allbut%
goto loop
:allparms
except (%allbut%) do dir /p
set allbut=
```

Through the use of ALLBUT.BAT, this process of maintaining a disk is made all that much simpler and more intuitive. You could, of course, create another batch file which would use the ONLY command in place of the EXCEPT command. This would simplify the case where you wanted to review only certain groups of files. Such a batch file could be named SHOWSET.BAT. For more details on the use of a loop structure and the NEXT command to process a variable number of command line parameters, and of environment variables refer to Chapter 4.

One area of caution with this batch file is that it is possible to concatenate enough file extensions together that the resulting EXCEPT command line will be longer than the 128 character limit imposed on batch file lines. When a batch file line is longer than 128 characters, MOS's command processor will truncate any excess.

An important thing to understand concerning the behavior of EXCEPT and ONLY is that they are not just a function of $$SHELL.SYS which only applies to internal commands such as DIR and DEL. Their action is system wide which means that whatever command is invoked by EXCEPT or ONLY is affected. A typical example of an external program which can benefit from this type of file set control is a backup utility.

For example, if you had two directories on your disk, \DEMOS and \TEMP, which you did not want to spend time backing up then the following command would backup all files in all directories in the current volume except any files and child directories within \DEMOS and \TEMP:

```
EXCEPT (DEMOS TEMP) DO EXPORT *.* A: /S
```

Note that since no path characters are allowed within the file list, you would have to be in the root directory to issue this command successfully.

Since EXCEPT and ONLY will affect external applications programs you must be careful to not get yourself in trouble. If you think that setting up a batch file with a command such as:

```
ONLY (*.DOC) DO WORDPROC
```

will set up a more safe environment for your users, preventing them from editing any files other than .DOC files, you could be in for some unpleasant surprises. EXCEPT and ONLY will restrict file read operations but not file writes. It would be possible for someone using the WORDPROC word processor to create other files besides those with a .DOC extension without knowing that they were overwriting existing files.

A safer approach would be to set such users up in their own directory or use MOS's security. Certainly the best approach would be to train your users to know what they should edit and what they shouldn't.

Another important caveat concerning EXCEPT and ONLY is how they will work, and not work, across a network. When either of these commands is used, an internal list of the files in the exclusion or inclusion set is created and made available to the MOS system kernel. When an EXCEPT or ONLY command is used to copy files from a network satellite machine to a server, the expected result will be attained. However, when file operations are done by a satellite which involve copying files from a server, or worse, deleting files on a server machine, the operating system kernel within the server will have no way to know that an exclusion or inclusion list should be consulted before carrying out the file operation.

For example, if D: is a network drive, issuing the command:

```
ONLY (*.TXT *.DOC) DO COPY *.* D:
```

would produce the desired result since the kernel on the satellite machine is in charge of selecting the files to be copied to the server. Likewise, the command:

```
EXCEPT (*.TXT *.DOC) DO DEL *.* /Y
```

could be used to clean up a directory on the satellite machine. However, the following command would result in all files being copied from the current directory of the network server:

```
ONLY (*.TXT *.DOC) DO COPY D:*.*
```

Even more significant:

```
EXCEPT (*.TXT *.DOC) DO DEL D:*.*/Y
```

would be interpreted by the network server's kernel as a request to delete all files in the current D: drive.

IMPORTANT: You must be sure you understand when a network file operation will be carried out by the satellite and when it will be carried out by the host. If you aren't familiar with network operations to this degree, do not use EXCEPT and ONLY in a network environment.