Digital Microsystems ™

HIDOS PROGRAMMER'S MANUAL

Version 1.0

Digital Microsystems
1755 Embarcadero, Oakland, CA 94606
(415) 532-3686 TWX 910-366-7310

TABLE OF CONTENTS

1.0 INTRODUCTION.

1.1 CP/M OVERVIEW.

Computers generally run a resident program called the Operating System (OS) that interprets the user's commands so that application programs can be run, files manipulated (with ERA, REN, PIP, for example) or disk information obtained (e.g., using DIR, STAT). In microcomputers, one standard operating system is called CP/M (Control Program/Monitor). CP/M and every other operating system use some method of determining which disk drive they are talking to, and which drives they can talk to.

For HiNet, which supports CP/M and several other operating systems, the drive can be assigned to a local floppy disk or a partition on a HiNet Network Master or local Hard Disk. Generally, the OS keeps track with a mapping between LOGICAL and PHYSICAL devices. In CP/M the logical devices are the familiar 'drives A, B, C, D' as well as printer, paper reader/punch, and console device, depending upon the specific hardware in the computer system.

CP/M keeps track of data on a disk in FILES. To the user, a file is just a program or a collection of data with a name. CP/M keeps track of files on a disk in a reserved space called the DIRECTORY. The directory contains all the information CP/M needs to be able to read from and write to each file. This information includes pointers to the space on the disk that the file occupies.

Space on a disk in CP/M is measured in logical blocks. The sizes of these blocks, which can be either 1k, 2k, 4k, 8k or 16k, are determined by the "Disk Parameter Block" (DPB). The DPB values are generally not under the application program's or programmer's control. The BIOS programmer makes a DPB for each kind of disk and disk drive available as directed in Digital Research's CP/M documentation.

1.2 DIRECTORIES AND FILE ALLOCATION

When a logical drive is accessed for the first time (after warm or cold booting) CP/M scans the directory and makes a map of the blocks currently used by the files on the disk. This is called the **ALLOCATION VECTOR** (AV). The OS uses this vector to allocate new blocks to files when writing, and to de-allocate blocks when files are deleted. In CP/M it is kept in the BIOS, i.e., in high memory above the TPA (user program space).

CP/M's security for a drive is maintained by a method called checksums. A **CHECKSUM VECTOR** (CV) is computed when the Allocation Vector is computed. This vector consists of one byte for every 128-byte CP/M record of the directory. The byte contains the sum of all the bytes in the record. Whenever the directory is changed (e.g., when a file is erased, closed, or when a new file or extent is opened) the checksum byte is updated.

Whenever a record of the directory is accessed, the corresponding checksum byte is recalculated. If the newly calculated value doesn't match the old value in the vector, the drive is set to R/O. CP/M thinks that the disk has been changed without it being notified, and tries to keep the user from destroying data by not allowing any writes until the checksums agree again.

As an example, consider two people using the same partition on the same hard disk at the same time. If user A changes information in the directory (say by ERA or PIP) the user's checksum vector will be updated properly. But the checksum vector in user B's memory will not be updated. The next time user B reads that part of the directory, the checksum byte for that record will not agree and he will get a "BDOS R/O ERROR". This is one problem with sharing.

There is also a variation on the previous problem. If user A allocates a block to a file, user B's OS does not know about it because only user A's Allocation Vector is updated. User B's OS could allocate the same block and overwrite user A's data. In this case, data is lost with no warning to the users, since the Operating Systems do not detect any error.

1.3 SHARING PARTITIONS

To enable more than one person to use a CP/M disk at the same time, a method must be devised so that when one person makes a change to the directory or the Allocation Vector (AV), everyone will know. The method DMS chose puts the AV on the disk instead of in the BIOS for shared partitions. Each user's OS knows (by virtue of a flag in the ALLOC table) that the drive is shared. Whenever the directory or the AV is to be updated, the user's OS locks the partition (via a HiNet BIOS lock command) so that it has sole access to the drive. Then it does its updating and when finished unlocks the partition so that another user can make changes if he or she wants to. In this way blocks are not allocated to more than one file and the directory is always kept up to date. Any user can read when the drive is locked, but only the person who has locked the drive can write to it.

HIDOS allows more than one person to work in the same partition at the same time but <u>NOT on the same file</u>. When working on a file, CP/M keeps in local memory a copy of the directory entry (in the form of a File Control Block), and modifies this copy as changes are made to the file (changes meaning adding or erasing blocks). The changes are not reflected in the directory until the file is closed, or a new extent is needed.

Since a local copy is kept by CP/M, the locking mechanism used above will not work. In fact, it is extremely impractical for any distributed-processor CP/M network to take care of this situation on the OS level. It would, of course, be desirable for the OS to take care of everything so that existing software could run with no modifications.

WHY THE OPERATING SYSTEM CAN'T LOCK RECORDS.

To show that it is impractical for the OS to provide for record locking, we will give some examples to illustrate the problem. If we assume the OS is totally responsible for locking and unlocking records, then the OS needs some rule to follow.

The OS does not know if a record needs to be locked when it is read. Thus every record read must be locked. This is not efficient, since many times a read will be to examine a record only. If no updating is involved there is no need to lock the record.

Assuming every read requires a lock, let User A and User B work on the same file in the same partition. User A reads and then locks a record X. When User B wants to read record X his OS would realize that the record was locked and could:

- ignore it and read it anyway,
- return without reading,
- return telling program record is locked,
- wait until read is granted.

The first two choices are obviously unacceptable. The third choice brings out a CP/M problem. The only values CP/M returns after a read are 0 and 1, representing either success or a failure (error). Therefore, in this case all the OS could do is report a failure to the User. If the program checks for this kind of error it will probably abort the program--an unacceptable result.

The fourth case leaves User B hanging while waiting for the record to be unlocked. How long will User B have to wait? The problem becomes-- when does the OS unlock the record? Several options are available:

• wait until another record is read by the same user,

• wait until that record is written back by the same user,

• wait for the file containing the record to be closed,

• wait until the user logs out.

We cannot expect that the user's program can tell the OS anything about the status of the record since we are assuming in these cases that the OS is totally responsible.

The last three options can be dismissed as impractical since:

• User A may never log out.

• He may never close the file since he may have only read from it.

• He may also never write that record back for the same reason.

The first option is also not acceptable but for a different reason. User programs can have logical records of almost any size. The OS, however, deals with a set record size. The OS only knows of that record size and can thus only lock records of the length it knows about.

This leads to a real problem. If the user's logical record size is bigger than the OS's record size, all of the user-requested record may not be locked. For example, let the user's record size be a 2-Kbyte chunk in some database, and the OS's record size be 128 bytes. When the user's program requests to read a 2K record, the OS will lock each 128-byte record as it is read, UNLOCKING the last 128-byte record it locked. Thus only the last 128 bytes of the 2K chunk remains locked.

A similar problem occurs when the user's record size is not an even divisor of the OS's record size. The user's records will generally go across the OS record boundaries. Therefore the OS will lock only one of the OS records needed to lock all of the user's record.

1.5 APPLICATION PROGRAMS & RECORD LOCKING

The solution to these problems is for the application program to do record locking and unlocking. The HiNet BIOS provides a locking /unlocking mechanism for this purpose.

The application program is designed to determine when a record needs to be locked, and when it can be unlocked. The program must avoid typical locking problems (like mutual lockout--when two programs have each locked records the other needs). The user program MUST do its own locking/unlocking to share a file or records.

2.0 USING HIDOS--LIMITS AND RESTRICTIONS.

Important! Do not use these BIOS calls on a shared partition:

Home	SetDma
SelDisk	Read
SetTrk	Write
SetSec	SecTran

<u>Only use BDOS calls for Reads and Writes!</u>

It is permissible to use DMS extended BIOS calls with the exception of SendNet and RecNet. If you are using HiNet commands with SendNet or RecNet, be sure you understand how HIDOS and HiNet work.

Do **NOT** change DPB's since HIDOS has special entries in the DPB that are non-standard.

Do **NOT** allow more than one person to use or modify a file at the same time. The application program that manipulates a file can allow this if it does some kind of locking on the records or file during read/modify/write routines. If the application program does not explicitly do locking then do not share files. Also, take care that no one is using a file in a shared partition when that file is erased by another user.

It is a good idea to establish a method for identifying files so that those people working on the same shared partition will not confuse their files with someone else's. If it is desired that more than one person have access to the same files, use the NetLock/NetUnLock utilities or some other method to avoid more than one person working on the same file at the same time. Always use the filename for the lockstring, rather than a shared partition name.

WARNING---In a shared partition, do not use any program that creates a temporary file of a fixed name. If two people are using such a program, the temporary files will get confused with disastrous results. For example, many compilers and word processors create temporary files of a fixed name. (WORDSTAR creates a temporary file called EDBACKUP.$$$ for every large file that is opened for editing or reading.) You will probably need to experiment or talk to the program manufacturer to be sure of this.

Directory information is volatile for shared disks. When a 'DIR' is done, remember that the information is instantly 'old' and may be incorrect. Someone else may have modified the directory information immediately after you asked to get it. Thus, files may disappear even though they were there for a 'DIR'. Therefore, you must make sure that your files are only used by you (you could use the NetLock/NetUnlock utilities).

Disk space usage information may not be correct. To get the most up-to-the-minute information on how much space is left on the disk, do a warm boot first. Remember, between the time you warm boot and a program such as STAT checks the disk space usage, someone else working in that partition could have changed the value without your local computer knowing about it.

Another point to watch: if a file is written to but is never closed, the directory will not show that the blocks allocated for the writes are used. They will, however, be allocated in the shared Allocation Vector on the disk since every block allocation is 'instantly' reflected in the shared Allocation Vector on the disk. Therefore there is unusable space on the disk. This leads to erroneous disk space information from programs such as STAT. You can run SHRALLOC to clean this up (see section 5.0).

3.0 HINET LOCKSTRINGS AND NETLOCK

HiNet NETLOCK is a warning device that tells the User if the partition or file in question is already being updated. This is a User-dependent system for file security.

Each User, before updating files in a shared partition, enters the lockstring sequence--**NETLOCK filename**--to secure a file. The Master checks to see if the lockstring is already in the NETLOCK Table. If it isn't then the lock is granted.

If the filename lockstring is already in the NETLOCK Table then the message ****This file or partition is locked** is displayed. The User must then wait until the lockstring is accepted when he or she resubmits it.

NETLOCK will not prevent a User from writing to a file that is locked. It is a warning only.

To unlock a file after updating it, the User enters the command NETUNLOK filename. The lockstring for the filename is removed from the Network's NETLOCK Table.

NOTE---NETLOCK lockstrings must be the name of the specific file being updated. Do not use HIDOS shared partition names for lockstrings.

HIDOS uses a similar method of lockstrings when it updates the Allocation Vector and the Directory; see the following section.

4.0 PROGRAMMING TECHNIQUES UNDER HIDOS

HIDOS is a modified version of the CP/M 2.2 BDOS and it essentially works in the same way. These modifications allow shared access to hard disk partitions. It is up to the transient program to do file and/or record locking as is necessary for the application.

When sharing a disk, the directory and the Allocation Vector must be kept accurate and up to date for all users. Whenever a BDOS function that modifies either of these is called, the local HIDOS does a HiNet lock over the network to the master. The HIDOS lockstring is the name of the partition. The OS will wait for access if the partition is locked. It should not have to wait for long since no HIDOS lock can last for longer than a BDOS function call. When the BDOS function is finished the partition is unlocked.

The directory and the Allocation Vector are both on the disk. Whenever either of these needs to be changed, it is read in from the disk, modified, and written back. This is all done under the protection of the HIDOS lock. In this way the data on the disk is always up to date.

In shared partitions, the Allocation Vector is stored in the first block after the directory. A file with the name of the partition followed by an exclamation point (!) is created with the block containing the Allocation Vector allocated to it. The file is stored under CP/M User 15. The file serves only as protection for the Allocation Vector and as a flag that the Allocation Vector was set up on the disk.

The local OS can tell that a partition is shared by checking a bit in the control byte of the hard disk's allocation table maintained by the system utility ALLOC. The byte is stored as part of the DPB in the BIOS. Whenever a disk is logged in, the local HIDOS checks to see if the disk is shared.

The BDOS functions that modify the directory are: Open, Close, Delete, Make, Rename, Set File Attributes, Read/Write Sequential/Random (when closing an extent and/or opening a new extent). The BDOS functions that modify the Allocation Vector are Delete and Write Sequential/Random.

When the Allocation Vector is needed for allocation or deallocation it is read in from the disk using parameters from the 'DMS DPB' into the Allocation Vector space in the BIOS. When the BDOS is done with the modification the Allocation Vector is written back to the disk before unlocking and returning from the BDOS call. The DMS DPB is an extended CP/M DPB.

HIDOS uses a directory high water mark different from the CP/M high water mark. In CP/M, when a disk is logged in (i.e., first accessed after a warm or cold boot) the entire directory is scanned. During this time the Allocation Vector and Checksum Vector are computed and initialized. At the same time, an internal variable is set to the last used entry in the directory, and this entry is the CP/M high water mark. If a file by the name of $$$.SUB is found, the appropriate flag is returned to the CCP to indicate there is a submit file to be run. Various internal variables are set up as well.

In HIDOS shared partitions, the high water mark is kept in the directory. It is set up by the system utility COMPRESS. An E8 hex is put in the entry following the last used entry, and is always kept up to date. The CP/M high water mark can only go up. However, the HIDOS high water mark goes up and down as necessary.

When a shared disk is logged in under HIDOS (first accessed after a warm or cold boot) the directory does not need to be scanned since the Allocation Vector is already set up and stored on the disk. The checksum vector security is not used on shared partitions since changes are expected to occur. The high water mark is already set up. The result of these changes is that shared HIDOS partitions boot very quickly. COMPRESS also compresses the directory so that directory searches execute much quicker than on normal CP/M partitions.

NOTE---Since the directory is not scanned at every warm or cold boot, SUBMIT files cannot be run on shared partitions.

5.0 SHARED PARTITION MAINTENANCE

Periodically COMPRESS and SHRALLOC should be run in a shared partition.

COMPRESS will compress the directory which, through normal use, aquires many gaps in it.

SHRALLOC will recompute the Allocation Vector and get rid of any discrepancies between the directory and the allocation vector.

Discrepancies can occur if a program writes to the disk but does not close the file. The blocks allocated to the file by the writes are reflected in the Allocation Vector but not in the directory. Thus, even though the blocks are not used, they cannot be re-allocated.

NOTE---The file **PARTITION-NAME.!** stores the allocation vector on the disk for each partition. If this file is missing when SHRALLOC is run, a warning message is displayed. The partition must then be changed from shared to non-shared in the ALLOC Table. SHRALLOC can then be run to restore the Allocation Vector; the partition can then be marked as shared again in the ALLOC Table.

When SHRALLOC is run it creates and then deletes a temporary file called TEMPFILE. If something goes wrong during the program's execution this file will be visible in the directory.

WARNING---When running COMPRESS and SHRALLOC you must make sure that no one else is using the partition.

NOTE---The next release of HINET/HIDOS will replace SHRALLOC with an automatic function in the ALLOC Table program. When a partition is marked as shared in the ALLOC Table, the Allocation Vector will be created at that time by ALLOC. COMPRESS will still be available to periodically clean up the directory.

6.0 FILE AND RECORD LOCKING

The idea behind file and record locking is to allow more than one person the ability to access and modify the same data at the same time, with each person getting the most recent data. In order to assure that you always have the most recent data, you need to do a "read" knowing that no one else has accessed the data file or record with the intention of modifying it. The procedure is to get ownership of the right to update the data (LOCK the data in question), read it, modify it, write it back, and then release ownership so another person can gain ownership. Read access without locking could always be granted with the understanding that someone else may be currently modifying what you have read.

As an example, consider an airline reservation system. The operator does unlocked reads to check seating availability. When the customer agrees to an available seat, the operator does a lock, then a read and checks to make sure the seat is still available--since someone else may have taken it between the time he or she did the unlocked read and the locked read. If still available, the operator reserves the seat by updating the record with that data, writing it back and unlocking the record.

If everyone only did locked reads, system performance would greatly suffer with people waiting for access to be granted for their locks before they could read or examine data. Such waiting is not necessary since most reads don't need to be locked.

6.1 RECORD LOCKING PROCEDURES

It is important to realize that you must reread any data that is to be modified before locking/modification/writing since what you have read without locking may not be current. Someone else may be changing the data while you are examining it.

You should develop a method for naming what needs to be locked. The file name is fine for file locking; for records the filename and record number combined could be a good name.

The HiNet locking mechanism locks a string of, at most, 13 bytes. See section 6.4 for examples of its use in CBASIC and Z80 Assembler.

To update a record, follow these procedures: lock, or wait for the lock to be granted, read the record, update it, write it back, and unlock it. If the record does not exist (i.e., it is not yet there to read) skip the read step. Probably some initialization should be done to the record. The method of determining if the record is there or not is application-dependent. For some applications all records can be allocated initially. For others, only extension may be allowed so that all allocated records are contiguous.

To extend a file you need to know which record is the current end. A specific record (say the first) can hold a pointer to the end. In this case, lock the record with the pointer. Using a random write (or sequential, if appropriate) write the record after the last record. This becomes the new last record, so update the pointer. Write the pointer record back to the disk. Unlock the record with the pointer.

6.2 DATA RECORD SIZE VS. CP/M RECORD SIZE

The logical record size equals the data record size and the application program record size. Complications can arise if the logical record size to be locked is not the same size as, or is not a multiple of, the CP/M record size (128 decimal, 80 hex bytes). It is highly recommended that the data record size be 128 bytes or an integer multiple of 128. The problem is that a CP/M record can contain parts of more than one logical record. Thus the logical record can be locked, but not the CP/M record. Therefore, more than one person can have the CP/M record in CP/M memory, each thinking he or she has sole ownership to modify that record. When they write back the logical record, that part of the CP/M record corresponding to some other logical record will be set to what it was when the read was done, overwriting any changes someone else may have made.

If you decide that you want a logical record size which is not equal to an integral number of CP/M records, you must lock the CP/M records, i.e., use the CP/M record name(s) that are being used by more than one logical record. There will be one or two records to lock.

Let us consider these four aspects of the problem:

1. The data record is much smaller than the CP/M record.

2. The data record is slightly smaller than the CP/M record.

3. The data record is much larger than the CP/M record.

4. The data record is slightly larger than the CP/M record.

Example 1. The data record is much smaller than the CP/M record.

CP/M records ...'r', 's', 't',...

```
|       r        |       s        |        t        |
-------------------------------------------------------
| )( 5 )( 6 )( |7 )( 8 )( 9 )(| 10 )( 11 )( 12 )(|
```

logical records ... 5, 6, 7, 8, 9, 10, 11, 12,...

If logical record 6 is locked, read, changed, written back, and unlocked by user A, and at the same time User B locks, reads, changes and writes back logical record 5, the last one to write will overwrite the previous user's change. This occurs because the same CP/M record "r" is read and written each time.

Example 2. The data record is slightly smaller than the CP/M record.

CP/M records ... 'c', 'd', 'e', 'f', 'g',...

```
|    c    |    d    |    e    |    f    |    g    |
--+---------+---------+---------+---------+---------+---
|         |         |         |         |         |
|  )( 3   |)(  4  )(   5   )(| 6   )(  | 7 )(     |
|         |         |         |         |         |
```

logical records ...3, 4, 5, 6, 7,...

The same problem exists as in #1. Notice this time that the logical record generally crosses a physical record boundary.

Example 3. The data record is much larger than the CP/M record.

CP/M records ...'j', 'k', 'l', 'm', 'n', 'o',...

```
|   j   |   k   |   l   |   m   |   n   |   o   |
-------------------------------------------------
|       |       |       |       |       |       |
|  )(   |   17  |       |)(     |   18  |       |)(
|       |       |       |       |       |       |
```

logical records ...17, 18,...

In this case if user A works on logical record 17 and user B on logical record 18 the conflict arises in CP/M record 'm'.

Example 4. The data record is slightly larger than the CP/M record.

CP/M records ...'j', 'k', 'l', 'm', 'n', 'o',...

```
|   j   |   k   |   l   |   m   |   n   |   o   |
-----------------------------------------------------
|       |       |       |       |       |       |
|)(   7 |)(   8 |  )( 9|    )( | 10  )(|  11   |)(
|       |       |       |       |       |       |
```

logical records ...7, 8, 9, 10, 11,...

The same situation as #3 occurs here, only now almost all the CP/M records are shared by two logical records (except CP/M record 'o' which is totally contained in logical record 11, so no problem there).

6.3 CALCULATION OF CP/M RECORDS USED BY LOGICAL RECORDS

Given a logical record we need to find the CP/M records that must be locked to avoid logical record conflict. There are one or two CP/M records in each of the four cases. The procedure is to find the CP/M records used by the first and last bytes of the logical record. We assume that the logical records are logically continuous and linearly numbered (i.e., records are numbered 2,3,4,5...).

To find the CP/M record used by the last byte of the logical record, first get the logical record number. If the first logical record is record "0" then add one to the logical record number. Now multiply this number by the logical record size and then divide by the CP/M record size (128 decimal). If there is a remainder, round up. The result is the CP/M record the END of the logical record uses.

Now, to find the CP/M record used by the beginning of the logical record, repeat the above procedure for the logical record just before the CP/M record. In this case, before dividing by the CP/M record length, add one so that the first byte of the logical record in question will be included.

These two records are the ones to lock. If they are the same record then only one record needs to be locked. If locking two CP/M records, watch out for lock-out. If you lock one record and the other is locked, unlock the first, wait a random amount of time and retry, since you may be competing with someone else for the same records.

It is assumed that all CP/M records between the first and last CP/M records of the logical record do not need to be locked since anyone wanting to read them must also lock the ends. This assumes no overlap of logical records.

If the logical data file has something other than logical records (such as a file header or record headers) then the size of this must be taken into account.

EXAMPLES

1: Logical file name = DBASE1
logical record size = 136 bytes
logical records = 1,2,3,4,5,.....
(Note: first record=1)

no headers or inter record info.

Want to lock logical record 23.

```
        (23 * 136) / 128 = 24.44 ---> 25
( (22 * 136) + 1 ) / 128 = 23.38 ---> 24
```

So lock 24 and 25. Lockstrings could be DBASE24 and DBASE25.

2: Logical file name = DBASE1
logical record size = 136 bytes
logical records = 0,1,2,3,4,5,.....
(Note: first record=0)
no headers or inter record info.

Want to lock logical record 23.

```
     ( (23+1) * 136 ) / 128 = 25.5    ---> 26
( [(22+1) * 136] + 1 ) / 128 = 24.44  ---> 25
```

So lock 25 and 26. Lockstrings could be DBASE25 and DBASE26.

3: Logical file name = DBASE1
logical record size = 136 bytes
logical records = 1,2,3,4,5,....
(Note: first record=1)
Assume there is a 32-byte file header before logical record 1.

Want to lock logical record 75.

```
     [(75 * 136) + 32] / 128 = 79.9  --> 80
{[(74 * 136) + 32] + 1} / 128 = 78.88 --> 79
```

So lock 80 and 79. Lockstrings could be DBASE79 and DBASE80.

4: Logical file name = SMALLDATA
logical record size = 18 bytes
logical records = 1,2,3,4...
(Note: first record=1)

Assume no headers or inter-record data.

Want to lock logical record 345.

```
     (345 * 18) / 128 =   48.5 --> 49
{(344 * 18) + 1 } / 128 =   48.3 --> 49
```

So lock 49. Lockstring could be SMALLDATA49.

6.4 HINET BIOS LOCK AND UNLOCK

Record locking and unlocking are invoked by first constructing a "lockstring" and then calling a BIOS lock or unlock entry point. The lockstring should indicate the file and record to be locked. Note that the lockstring can, in fact, contain any sequence of bytes. However, to allow different applications to utilize record locking on the same HiNet system requires that a convention be established. The recommended convention is to use the file name as the first 8 characters and the record number as the last 5 characters of the lock string.

The addresses of the BIOS lock and unlock entry points need to be calculated at program run time. The entry points are addresses in the Digital Microsystem extended BIOS jump table.The addresses are calculated as follows:

BIOS Lock

1. Get the address of the standard BIOS warm boot jump. This is kept at locations 1 and 2.

2. Add 93 (5d hex) to the warm boot address. This is the offset to the lock function.

3. The result is the address of the BIOS lock entry point.

BIOS Unlock

1. Get the address of the standard BIOS warm boot jump. This is kept at locations 1 and 2.

2. Add 99 (63 hex) to the warm boot address. This is the offset to the unlock function.

3. The result is the address of the BIOS unlock entry point.

Before calling the BIOS lock or unlock entry points, locations 74 (4A hex) and 75 (4B hex) should point to the lockstring, i.e., contain the address of the string to be locked. The first byte of the string is an integer from 1 to 13, indicating the length of the string.

The BIOS routines return immediately and put the outcome of the request in location 73 (49 hex). This is the status of the request.

Lock Request

Returned Status	Meaning
0	Lock accepted. The lockstring was entered into the master lockstring table.
1	Lock denied. The lockstring is already in the table, i.e. the string is already locked.
2	Lock table full, or string length byte is bad (= 0 or > 13).

Unlock Request

Returned Status	Meaning
0	Unlock accepted. String was found in master lockstring table and removed.
2	Unlock failed. String was not found in master lockstring table, or string length byte is bad (= 0 or > 13).

The CBASIC functions "fn.lock" and "fn.unlock" can be used to interface with the lock and unlock routines in the BIOS. Similar interface functions can easily be written for other compilers.

```
DEF FN.LOCKWORK%(STRING$,FUNC%)
    ADDR% = SADD(STRING$)
    HIGH% = (ADDR%/100h) AND 0FFh
    IF ADDR% < 0 THEN HIGH% = HIGH% - 1
    POKE 4AH,ADDR% AND 0FFH
    POKE 4BH,HIGH%

    CALL ((PEEK(2)*100h) OR PEEK(1)) + FUNC%
    FN.LOCKWORK% = PEEK(49H)
    RETURN
FEND

DEF FN.LOCK%(STRING$)
    FN.LOCK% = FN.LOCKWORK%(STRING$,5DH)
    RETURN
FEND
DEF FN.UNLOCK%(STRING$)
```

```
        FN.UNLOCK% = FN.LOCKWORK%(STRING$,63H)
        RETURN
    FEND
```

The following program demonstrates how to use the record locking functions. First, a file containing 128 records is created. Several users can then simultaneously run this program, and update different records in the file at will. The program will allow only one user at a time to update any particular record; however, several users are allowed to update DIFFERENT records in the file simultaneously. The lock functions are on the "LOCKFNS.BAS" file.

The statement "READ #1,R;" is needed after a write to force CBASIC to flush its I/O buffer for file number 1. Without this statement, the record will not be updated on the disk until the next random read or write to that file. This is due to a peculiarity in the I/O algorithms used by CBASIC. Similar problems may be encountered with other compilers.

```
%INCLUDE LOCKFNS
    FILENAME$ = "DEMO.DAT"
    INPUT "ENTER 0 TO CREATE, 1 TO
                            UPDATE DEMO FILE";I
    IF I = 0 THEN \
       CREATE FILENAME$ RECL 128 AS 1 :\
       FOR I = 1 TO 128 :\
       PRINT #1;I :\
       NEXT I :\
       CLOSE 1
    OPEN FILENAME$ RECL 128 AS 1
```

```
100     INPUT "RECORD NUMBER";R
        LOCKSTRING$ = "DEMO     "+STR$(R)
        WHILE FN.LOCK%(LOCKSTRING$) <> 0
           WEND
        READ #1,R;I
        PRINT "OLD VALUE";I
        INPUT "NEW VALUE";I
        PRINT #1,R;I
        READ #1,R;        REM  flush the record
        I% = FN.UNLOCK%(LOCKSTRING$)
        GO TO 100
        END
```

```
TDL Z80 CP/M DISK ASSEMBLER VERSION 2.21                          PAGE 1


                        ;       The following is a z80 assembly program in TDL
                        ;mnemonics. It shows how to compute the address of and
                        ;use the BIOS netlock/netunlock functions.
                                .pabs
                                .phex
0100                            .loc    100h

0000                    wboot   ==      0
0005                    bdos    ==      5
0009                    print   ==      9
000D                    cr      ==      0Dh
000A                    lf      ==      0Ah

004A                    locAddr ==      4Ah     ;address of lock string

0049                    locStat ==      49h     ;BIOS lock status returned
                                                ;as set below

0000                    locAccept ==    0       ;lock or unlock is accepted

0001                    locDeny ==      1       ;lock request, string exists

0002                    locReject ==    2       ; if lock, then table full or
                                                ;lockstring length = 0 or > 13
                                                ; if unlock, then string not in
                                                ;table or lockstring length = 0
                                                ;or > 13.

005D                    locOffset ==    5Dh     ;Offset from standard BIOS jump
                                                ;table (warm boot jump) into
                                                ;Digital Microsystem's extended
                                                ;BIOS jump table to the netlock
                                                ;call.

0063                    unlocOffset ==  63h     ;Offset from standard BIOS jump
                                                ;table (warm boot jump) into
                                                ;Digital Microsystem's extended
                                                ;BIOS jump table to the netunlock
                                                ;call.

                        ;=======================================================
                        ;

0100                    start:
0100    FB                      ei                      ; for zdt
0101    31 0186                 lxi     sp,stack        ; set up stack
0104    CD 011D                 call    Lock            ; try to lock
0107    CD 013B                 call    UnLock          ; try to unlock
010A    C3 0000                 jmp     wboot           ; exit via warm boot.
```

```
TDL Z80 CP/M DISK ASSEMBLER VERSION 2.21                              PAGE 2

                        ;------------------------------------
                        ;
 010D                   NetLock:
 010D    2A 0001                lhld    wboot + 1       ; Address of standard
                                                        ;BIOS jump table

 0110    11 005D                lxi     D,locOffset     ; Offset into DMS
                                                        ;extended BIOS jump
                                                        ;table
 0113    19                     dad     D               ; HL = address of DMS
                                                        ;netlock call.
 0114    E9                     pchl

                        ;------------------------------------
                        ;
 0115                   NetUnLock:
 0115    2A 0001                lhld    wboot + 1       ; Address of standard
                                                        ;BIOS jump table
 0118    11 0063                lxi     D,unlocOffset   ; Offset into DMS
                                                        ;extended BIOS jump
                                                        ;table
 011B    19                     dad     D               ; HL = address of DMS
                                                        ;netunlock call.
 011C    E9                     pchl

                        ;----------------------------------------
                        ;
                        ; Try to lock string 'locString'.
                        ;
                        ;.................

 011D                   Lock:
 011D    21 0278                lxi     H,locString     ; Set up address of
 0120    22 004A                shld    locAddr         ;string to lock.

 0123    CD 010D                call    NetLock         ;Ask master lock string

 0126    3A 0049                lda     locStat         ; Get returned status
 0129    FE00                   cpi     locAccept       ; Was lock granted?
 012B    CA 0154                jz      lkGranted

 012E    FE01                   cpi     locDeny         ; Is lockstring already
 0130    CA 0159                jz      locked          ;in master's table?

 0133    FE02                   cpi     locReject       ; Was lock rejected?
 0135    CA 015E                jz      tableFull       ; String length bad
                                                        ;or lock table is full.

 0138    C3 016D                jmp     lockError       ; If none of above,
                                                        ;HiNet error.
```

```
TDL Z80 CP/M DISK ASSEMBLER VERSION 2.21                                PAGE 3

                        ;---------------------------------------------
                        ;
                        ; Try to unlock string 'locString'.

                        ;................

013B                    UnLock:
013B    21 0278                 lxi     H,locString     ; Set up address of
013E    22 004A                 shld    locAddr         ;string to lock.

0141    CD 0115                 call    NetUnLock       ; Ask master to unlock
                                                        ;locString

0144    3A 0049                 lda     locStat         ; Get returned status
0147    FE00                    cpi     locAccept       ; Was unlock granted?
0149    CA 0163                 jz      unLkGranted

014C    FE02                    cpi     locReject       ; Was unlock rejected?
014E    CA 0168                 jz      notLocked       ; String length bad
                                                        ;or locString not in
                                                        ;table, i.e. locString
                                                        ;is not locked.

0151    C3 016D                 jmp     lockError       ; If none of above,
                                                        ;HiNet error.

                        ;---------------------------------------------

0154                    lkGranted:
0154    11 0186                 lxi     D,locOkMsg
0157    1817                    jmpr    PrintMsg

0159                    locked:
0159    11 0196                 lxi     D,lockdMsg
015C    1812                    jmpr    PrintMsg

015E                    tableFull:
015E    11 01BF                 lxi     D,fullTableMsg
0161    180D                    jmpr    PrintMsg

0163                    unLkGranted:
0163    11 0205                 lxi     D,unLkOkMsg
0166    1808                    jmpr    PrintMsg

0168                    notLocked:
0168    11 0217                 lxi     D,notInTable
016B    1803                    jmpr    PrintMsg

016D                    lockError:
016D    11 0269                 lxi     D,netErrMsg

0170                    PrintMsg:
0170    0E09                    mvi     C,print
```

```
TDL Z80 CP/M DISK ASSEMBLER VERSION 2.21                               PAGE 4

  0172     CD 0005               call    bdos
  0175     C9                    ret

                         ;..............................

  0176     767676767676          .byte   76h,76h,76h,76h,76h,76h,76h,76h
  017E     767676767676          .byte   76h,76h,76h,76h,76h,76h,76h,76h
  0186                   stack:

  0186     4C6F636B2067 locOkMsg:        .ascii  'Lock granted.'[cr][lf]'$'

  0196     4C6F636B2064 lockdMsg:        .ascii  'Lock denied, locString '

  01AD     616C72656164                  .ascii  'already locked.'[cr][lf]'$'

  01BF     4C6F636B2064 fullTableMsg:    .ascii  'Lock denied, master '
  01D3     6C6F636B7374                  .ascii  'lockstring table is full,'
  01EC     206F72206261                  .ascii  ' or bad string length.'
  0202     0D0A24                        .ascii  [cr][lf]'$'

  0205     556E6C6F636B unlkOkMsg:       .ascii  'Unlock granted.'[cr][lf]'$'

  0217     556E6C6F636B notInTable:      .ascii  'Unlock failure, locString not'
  0234     20696E206D61                  .ascii  ' in master lockstring table,'
  0250     206F72206261                  .ascii  ' or bad string length.'
  0266     0D0A24                        .ascii  [cr][lf]'$'

  0269     48694E657420 netErrMsg:       .ascii  'HiNet error.'[cr][lf]'$'


  0278     0D           locString:       .byte   13
  0279     4F7572444261                  .ascii  'OurDBase12345'

                                 .end


+++++ SYMBOL TABLE +++++

BDOS    0005        CR      000D        FULLTA  01BF        LF      000A
LKGRAN  0154        LOCACC  0000        LOCADD  004A        LOCDEN  0001
LOCK    011D        LOCKDM  0196        LOCKED  0159        LOCKER  016D
LOCOFF  005D        LOCOKM  0186        LOCREJ  0002        LOCSTA  0049
LOCSTR  0278        NETERR  0269        NETLOC  010D        NETUNL  0115
NOTINT  0217        NOTLOC  0168        PRINT   0009        PRINTM  0170
STACK   0186        START   0100        TABLEF  015E        UNLKGR  0163
UNLKOK  0205        UNLOCK  013B        UNLOCO  0063        WBOOT   0000
 .BLNK. 0000:03 X    .DATA. 0000*   X    .PROG. 0000'   X
```

6.5 NETWORK BUFFER USAGE

The HiNet BIOS normally provides a 1k network buffer to enhance system performance. However, for some programs such as multi-user data bases, data must not be buffered or obsolete data may mistakenly be taken as current.

In the past, programs that had to ensure that all data was current would first read (unwanted) data into the 1k buffer so that the read of desired data would come across the network and not from the 1k buffer. This is neither elegant or efficient. Starting with the HiNet BIOS version 247 there is a DMS-specific BIOS jump vector (SetNetMode) that allows a transient (i.e., user) program to select the network buffer usage mode. The three buffer modes are:

0) Always use the 1k network buffer. This is the default mode; it is automatically selected after a cold or warm boot.

1) Do not use the buffer contents on the next NetRead request - force a network transmission to ensure current data. This will replace the 1k network buffer contents; all subsequent NetReads will use the buffer contents.

2) Do not use the buffer contents until a cold or warm boot or until the program changes the network buffer usage mode.

The SetNetMode jump vector is available in both the network Master and the network Stations but will result in a Call Error on a stand-alone system. Since the network Master never has the 1k network buffer, the SetNetMode jump vector will do nothing - it is there simply so that networking programs do not have to check to see if they are running on a Master or Station.

To call the SetNetMode vector perform the following steps:

1) Load locations 0001 and 0002. This is the address of the warm boot vector.

2) Add the offset of the DMS-specific jump table to the offset of the network function that is to be accessed and move the value into register DE.

3) Add the value of register DE to the contents of register HL.

4) Load register C with the desired mode:

   0 => always use the network buffer

   1 => don't use the network buffer the next time only

   2 => never use the network buffer

5) Execute the code at the address obtained in step 2.

The previous NetMode value is returned in register A in case you wish to restore the NetMode to its previous state.

Reproduced below is a tested assembly program fragment that sets the NetBufMode to Buffer Mode 1, "Do not use the 1k buffer for the next NetRead only".

```
        .ident netjmp
        .pabs
        .phex
        .loc    100h
;
; This program tests code which is to be included
; in the HiDos programmer's guide.
;
BiosVector  ==  01h     ;CP/M W B jump address
DMSoffset   ==  (5Dh-3) ;First jump in DMS table
Netmodedisp ==  (15*3)  ;# of jumps to SetNetMode
NotNextTime ==  01      ;Direct read next time
;
        lhld    BiosVector      ;CPM warm boot
        lxi     D,DMSoffset+Netmodedisp
                        ;# of bytes to SetNetMode
        dad     D       ;HL = addr of code in bios
        mvi     C,NotNextTime ;direct read next time
        pchl            ;execute it, return to
                        ;calling routine
;
        .END
```

At system assembly time, the choice may be made to not include the 1k network buffer in the system at all; this will automatically ensure that all NetRead requests get current data from the network. This generally provides poorer performance than when using the network buffer in conjunction with the SetNetMode vector. The distribution versions of the HiNet bios all use the network buffer for the stations. If the HiNet BIOS is assembled without the network buffer then the SetNetMode vector is still present but does nothing.

INDEX