CCPMPRG0.WS4   (Concurrent CP/M Programmer's Reference Guide, Chapter 0)

(Retyped by Emmanuel ROCHE.)

Digital Research
Concurrent CP/M
Operating System
Release 3.1
Programmer's Reference Guide

First Edition: January 1984

Foreword

Concurrent CP/M is a multi- or single-user operating system targeted specifically for the Intel 8086/8088 family of microprocessors. It supports multiple CP/M programming environments, each implemented on a virtual console. A different task runs concurrently in each environment.

This manual describes the invariant programming interface to Concurrent CP/M. It supports the applications programmer who must create applications programs that run in the Concurrent CP/M environment.

Section 1 offers an overview of the entire operating system.

Section 2 describes the structure of the Concurrent CP/M file system.

Section 3 explains the format, structure, and uses of the transient commands in the Conncurrent CP/M environment.

Section 4 explains the creation of transient command files in the Concurrent CP/M environment.

Section 5 documents the structure and creation of resident system processes or resident command files permanently installed in the Concurrent CP/M environment.

Section 6 describes all the Concurrent CP/M system calls.

Concurrent CP/M is supported and documented through four manuals:

- The "Concurrent CP/M Operating System User's Guide" (hereinafter cited as "Concurrent CP/M User's Guide") documents the user's interface to Concurrent CP/M, explaining the various features used to execute applications programs and Digital Research utility programs.

- The "Concurrent CP/M Operating System Programmer's Reference Guide" (hereinafter cited as "Concurrent CP/M Programmer's Reference Guide") documents the applications programmer's interface to Concurrent CP/M, explaining the internal file structure and system entry points, information that is essential for creating applications programs that run in the Concurrent CP/M environment.

- The "Concurrent CP/M Operating System Programmer's Utilities Guide" (hereinafter cited as "Programmer's Utilities Guide") documents the Digital Research utility programs that programmers use to write, debug, and verify applications programs written for the Concurrent CP/M environment.

- The "Concurrent CP/M Operating System System Guide" (hereinafter cited as "Concurrent CP/M System Guide") documents the internal, hardware-dependent structures of Concurrent CP/M.

Table of Contents

Figures

Listings

EOF

CCPMPRG1.WS4   (Concurrent CP/M Programmer's Reference Guide, Chapter 1)

(Retyped by Emmanuel ROCHE.)

Section 1: Concurrent CP/M system overview

1.1 Introduction

Concurrent CP/M is a multi- or single-user, multitasking operating system that lets you run multiple programs simultaneously by initiating tasks on two or more terminals or virtual consoles. Applications programs have access to system calls used by Concurrent CP/M to control the multiprogramming environment. As a result, Concurrent CP/M supports extended features, such as communication among and synchronization of independently running processes. Figure 1-1 depicts the relationships between application programs, virtual environments, virtual consoles, and the user terminal.

```
                   +---------+----------------+
                   | Logical :    Physical    |
                   |   OS    :   I/O system   |
+-------------+--\+------+......+-----\+---------+  |
| Application |--/|   Virtual   :-----/| Virtual |  |
|   program   |   | environment :  :   | console |  |
|      N      |/--|      N      :/-----|    N    |  |
+-------------+\--+------+......+\-----+---------+  |
                   :         :               |
+-------------+--\+------+......+-----\+---------+  |
| Application |--/|   Virtual   :-----/| Virtual |  |
|   program   |   | environment :  :   | console |  |
|      2      |/--|      2      :/-----|    2    |  |
+-------------+\--+------+......+\-----+---------+  |
                   |         :               |
+-------------+--\+------+......+---------------------\+----------+
| Application |--/|   Virtual   :----------------------/| Physical |
|   program   |   | environment :  :               |   | console  |
|      1      |/--|      1      :/----------------------|          |
+-------------+\--+------+......+\---------------------+----------+
                   |         :               |
+-------------+--\+------+......+-----\+---------+  |
|  Terminal   |--/|   Virtual   :-----/| Virtual |  |
|  Message    |   | environment :  :   | console |  |
|  Process 0  |/--|      0      :/-----|    0    |  |
+-------------+\--+------+......+\-----+---------+  |
                   |         :               |
                   +---------+-------+-+------+
                    /\ /\ /\         | |
                    || || ||         | |
  +---------------------+| || ||         | |
  |+---------------------+ || ||         | |
      ||     +--------------+| ||       | |
      ||     |+--------------+ ||       | |
      ||     ||        +-----+|       | |
      ||     ||        |+-----+       | |
      ||     ||        ||           | |
      \/     \/        \/           \ /
   +-------+ +-------+  +-------+      +----------+
   | Disk  | | Disk  |  | Disk  |      | Hardcopy |
   | drive | | drive |...| drive |      | printer  |
   |   A   | |   B   |  |   P   |      +----------+
   +-------+ +-------+  +-------+
```

Figure 1-1. Concurrent CP/M virtual/physical environments

In the Concurrent CP/M environment, there is an important distinction between a program and a process. A program is simply a block of code residing somewhere in memory or on disk; it is essentially static. A process, on the other hand, is a dynamic entity. You can think of it as a logical machine that executes not only the program code, but also the operating system routines necessary to support the program's functions.

When Concurrent CP/M loads a program, it creates a process associated with the loaded program. Subsequently, it is the process, rather than the program, that obtains access to the system's resources. Thus, Concurrent CP/M monitors the process, not the program. This distinction is a subtle one, but vital to your understanding of system operation as a whole.

Processes running under Concurrent CP/M fall into two categories: transient processes and Resident System Processes (RSPs). Transient processes run programs loaded into memory from disk in response to a user command or system calls made by another process. Resident System Process run code that is a part of the operating system itself. RSPs become an integral part of the operating system image during system generation. They are immediately available to perform operating system tasks. For example, the CLOCK process is an RSP that maintains the time of day within the operating system.

The following list briefly summarizes Concurrent CP/M's capabilities.

- Interprocess communication, synchronization, and mutual exclusion functions are provided by system queues.

- A logical interrupt mechanism using flags allows Concurrent CP/M to interface with any physical interrupt structure.

- System timing function enable process running under Concurrent CP/M to compute elapsed times, delay execution for specified intervals, and to access and set the current date and time.

- Shared file system allows multiple programs to access common data files while maintaining data integrity.

- Shared code support eliminates program loading of another copy of the same program, and conserves memory space.

- 8087 support takes advantage of fast 8087 math instructions.

- Virtual console handling lets a single user run multiple programs, each in its own console environment.

- Real-time process control allows communications and data acquisition without loss of information.

Functionally, Concurrent CP/M is composed of several distinct modules, as shown in Figure 1-2.

```
           +------------+  +------------+
           | Aplication |  | Terminal   |
           | processes  |  | Message    |
           |            |  | Process XX |
           +------------+  +------------+
                 ||              ||
                 \/              \/
     +-----------------------------------------------+
     |               OS Supervisor                   |
     +-----------------------------------------------+
        ||          ||          ||          ||
        \/          \/          \/          \/
     +-----------+  +-------+   +---------+  +---------+
     | Character |  | Basic |   | Memory  |  |  Real-  |
     |    I/O    |  | Disk  |   |  pool   |  |  Time   |
     |   module  |  |  OS   |   | manager |  | Monitor |
     +-----------+  +-------+   +---------+  +---------+
       : : : :          ||          ||          ||
       V V V V          ||          ||          ||
     +---------+        ||          ||          ||
     | Virtual |        ||          ||          ||
     | console |        ||          ||          ||
     | session |        ||          ||          ||
     | manager |        ||          ||          ||
     +---------+        ||          ||          ||
        ||          ||          ||          ||
        \/          \/          \/          \/
     +-----------------------------------------------+
     |          :          :          :              |
     |          :   Extended I/O System   :          |
     |          :          :          :              |
     +-------+---------------------------------------+
       ^   |        ^                      ^
       |   |        +-----------+          |
       |   +--------+           |          |
       V            V           V      +-----+-----+
     +---------+ +----------+ +----------+ | Interrupt |
     | System  | | Hardcopy | | Diskette | |  control  |
     | console | | printer  | |  drives  | |   logic   |
     +---------+ +----------+ +----------+ +-----------+
```

Figure 1-2. Concurrent CP/M functional modules

- The Supervisor (SUP)
- The Real-Time Monitor (RTM)
- The Memory Management module (MEM)
- The Character I/O module (CIO)
- The Virtual Console Screen Manager
- The Basic Disk Operating System (BDOS)
- The Extended I/O System (XIOS)
- The Terminal Message Process (TMP)

The SUP module handles miscellaneous system calls, such as returning the version number or the address of the System Data Area. SUP also calls other system calls, when necessary.

The RTM module monitors the execution of running processes, and arbitrates conflicts for the system's resources.

The MEM module allocates and frees memory upon demand from executing processes.

The CIO module handles all character I/O for console and list devices in the system.

The Virtual Console Screen Manager extends the CIO to support virtual console environments.

The BDOS is the hardware-independent module that contains the logically invariant portion of the file system for Concurrent CP/M. The BDOS file system is explained in detail in Section 2, "The Concurrent CP/M file system".

The XIOS is the hardware-dependent module that defines the interface of Concurrent CP/M to a specific hardware environment. See the "Concurrent CP/M System Guide" for an explanation of the XIOS.

When Concurrent CP/M is executing a single program on a single virtual console, its speed approximates that of CP/M-86. But, when multiple processes are running on several virtual consoles, the execution of each individual process slows according to the proportion of I/O to CPU resources it requires. A process that performs a large amount of I/O in proportion to computing exhibits only minor speed degradation. This also applies to a process that performs a large amount of computing, but runs concurrently with other processes that are largely I/O-bound. On the other hand, significant speed degradation occurs when more than one compute-bound process is running.

1.2 Supervisor (SUP)

The Supervisor module (SUP) manages the interface between processes and the operating system kernel. It also manages internal communication between operating system modules. All system calls, whether they originate from a transient process or internally from another system module, go through a common table-driven function interface in SUP. SUP also handles the P_LOAD (Load Process) and P_CLI (Call Command Line Interpreter) system calls.

1.3 Real-Time Monitor (RTM)

The Real-Time Monitor (RTM) is the real-time multitasking nucleus of Concurrent CP/M. The RTM performs process dispatching, queue management, flag management, device polling, and system timing tasks. User programs can also call many of the RTM system calls used to perform these tasks.

1.3.1 Process dispatching

Although Concurrent CP/M is a multiprocess operating system, only one process has access to the CPU resource at any given time. Unless you specifically write a program to communicate or synchronize execution with other processes, a process is unaware of other processes competing for system resources.

The primary task of the RTM is to transfer, or dispatch, the CPU resource from one process to another. The RTM module called the Dispatcher performs this task. The RTM maintains two data structures, the Process Descriptor (PD) and the User Data Area (UDA), for each process running under Concurrent CP/M. The Dispatcher uses these data structures to save and restore the current state of each running process.

Each process in the system resides in one of three states: ready, running, or suspended. A ready process is one that is waiting for the CPU resource only. A running process is one that the CPU is currently executing. A suspended process is one that is waiting for a system resource or specified event, such as the occurrence of an interrupt, an indication that polled hardware is ready, or the expiration of a delay period.

Any existing process is represented on a system list. The Dispatcher removes a process from one list, and places it on another. The Process Descriptor of the currently running process is the first entry on the Ready List. Other processes ready to run are represented on the Ready List, in order of priority. Suspended processes are on other system lists, depending on why the processes were suspended.

A dispatch operation can be summarized as follows:

1. The Dispatcher suspends the process from execution, and stores its current state in the Process Descriptor and the UDA.

2. The Dispatcher places the process on an appropriate system list, depending on why the Dispatcher was called. For example, if a process is to delay for a certain number of system ticks, its Process Descriptor is placed on the Delay List. When a process releases a resource, the process is usually placed back on the Ready List. If another process is waiting for the resource, that process is taken off its current system list and also placed on the Ready List.

3. The highest priority process on the Ready List is chosen for execution. If two or more processes have the same priority, the process that has waited the longest executes first.

4. The Dispatcher restores the state of the selected process from its Process Descriptor and UDA, and gives it the CPU resource.

5. The process executes until it needs a busy resource, a resource needed by another process becomes available, or an interrupt occurs. At this point, a dispatch occurs, allowing another process to run.

Only processes on the Ready List are eligible for selection during dispatch. By definition, a process is on the Ready List if it is waiting only for the CPU resource. Processes waiting for other system resources cannot execute until the resources that they require are available. Concurrent CP/M blocks a process from executing if it is waiting for:

- a queue message, so it can complete a Q_READ operation.

- space to become available in a queue, so it can complete a Q_WRITE operation.

- a console or list device to become available.

- a specified number of system clock ticks before it can be removed from the system Delay List.

- an I/O event to complete.

These situations are discussed in greater detail in the following sections.

A running process not needing a resource and not releasing one runs until an interrupt causes a dispatch. While not all interrupts cause dispatches, the system clock generates interrupts every clock tick, and forces a dispatch each time. Clock ticks usually occur 60 times a second (approximately every 16.67 milliseconds in the USA, every 20 milliseconds, or 50 times a second, in Europe), and allow time sharing within a real-time environment.

Concurrent CP/M is a priority-driven system. This means that, during a dispatch, the operating system gives the CPU resource to the process with the best priority. The Dispatcher allots equal shares of the system's resources to processes with the same priority. With priority dispatching, the system never passes control to a lower-priority process if there is a higher-priority process on the Ready List. Because high-priority, compute-bound processes tend to monopolize the CPU resource, it is best to reduce their priority, to avoid degrading overall system performance.

1.3.2 Queue management

Queues perform several critical functions for processes running under Concurrent CP/M. A process can use a queue for communicating with another process, synchronizing its execution with that of another process, and for exclusion of other processes from protected system resources. A process can make, open, delete, read from, or write to a queue with system calls similar to those used to manage disk files.

Each system queue consists of two parts: the queue descriptor, and the queue buffer. Concurrent CP/M implements these special data structures as memory files that contain room for a specified number of fixed-length messages.

When the Q_MAKE system call creates a queue, this queue is assigned a unique 8-character name. As the name "queue" implies, messages are read from a queue on a first-in, first-out basis.

A process can read from or write to a queue conditionally or unconditionally. If the queue is empty when a conditional read is performed, or full when a conditional write is performed, the system returns an error code to the calling process. On the other hand, if a process attempts an unconditional queue operation in these circumstances, the system suspends it from exection until the operation becomes possible.

More than one process can wait to read or write a queue message from the same queue at the same time. When these operations become possible, the system restores the highest priority process first; processes with the same priority are restored on a first-come, first served basis.

Mutual exclusion queues are a special type of queue under Concurrent CP/M. They contain one message of zero length, and their names follow a convention, beginning with the uppercase letters "MX". A mutual exclusion queue acts as a binary semaphore, ensuring that only one process uses a resource at any time.

Access to a resource protected by a mutual exclusion queue takes place as follows:

1. A process issues an unconditional Q_READ call to the MX queue protecting the resource, thereby suspending itself if the message is not available.

2. When the message becomes available, the process accesses the protected resource. Note that, from the time the process issues the unconditional read, any other process attempting to access the same resource is suspended.

3. The process writes the zero-length message back to the queue when it has finished using the protected resource, thus freeing the resource for other processes.

As an example, the system mutual exclusion queue, MXdisk, ensures that processes cannot access the file system simultaneously. Note that the BDOS, not the application software, executes the preceding series of queue calls. Therefore, the mutual exclusion process is transparent to the programmer, who is only responsible for originating the disk system calls.

Mutual exclusion queues differ from normal queues in another way. When a process reads a message from a mutual exclusion queue, the RTM notes the Process Descriptor address within the Queue Descriptor. This establishes the owner of the queue message. If the operating system aborts the process while it owns the mutual exclusion message, the RTM automatically writes the message back to all mutual exclusion queues whose messages are owned by the aborted process. This grants other processes access to protected resources owned by the aborted process.

1.3.3 System timing functions

Concurrent CP/M's timing system calls include keeping the time of day and delaying the execution of a process for a specified period of time. An internal process called CLOCK provides the time of day for the system. This process issues DEV_WAITFLAG system calls on the system's one second flag, Flag 2. When the XIOS Tick Interrupt Handler sets this flag, it initiates the CLOCK process, which then increments the internal time and date.

Subsequently, the CLOCK process makes another DEV_WAITFLAG call, and suspends itself until the flag is set again. Concurrent CP/M provides system calls that allow you to set and access the internal date and time. In addition, the file system uses the internal time and date to record when a file was updated, creater or last accessed.

The P_DELAY system call replaces the typical programmed delay loop for delaying process execution. P_DELAY requires that Flag 1, the system tick flag, be set approximately every 16.67 milliseconds, or 60 times a second in the USA (20 milliseconds and 50 times a second in Europe); the XIOS Tick Interrupt Handler also sets this flag. When a process makes a P_DELAY system call, it specifies the number of ticks for which the operating system is to suspend it from execution. The system maintains the address of the Process Descriptor for the process on an internal Delay List, along with its current delay tick count. When a DEV_SETFLAG call occurs, setting Flag 1, the tick count is decremented. When the delay count goes to zero, the system removes the process from the Delay List, and places it on the Ready List.

Note: The length of a tick might vary from installation to installation. For instance, in Europe, a tick is commonly 20 milliseconds, yielding 50 ticks per second. The description of the P_DELAY system call in Section 6, "System calls", describes how to determine the correct number of ticks to delay 1 second.

1.4 Memory module (MEM)

Concurrent CP/M supports an extended, fixed partition model of memory management; the Memory Module handles all memory management system calls. In practice, the exact method that the operating system uses to allocate and free memory is transparent to the application program. Therefore, you should take care to write code independent of the memory management model; use only Concurrent CP/M specific memory system calls described in Section 6, "System calls".

1.5 Basic Disk Operating System (BDOS)

Except for auxiliary device support, the Concurrent CP/M BDOS is an upward-compatible version of the single-tasking CP/M-86 BDOS. It handles file creation and deletion, facilitates sequential or random file access, and allocates and frees disk space. In most cases, CP/M-86 programs that make BDOS calls for I/O can run under Concurrent CP/M without modification. Concurrent CP/M's BDOS is extended to provide support for multiple virtual consoles and list devices. In addition, the file system is extended to provide services required in a multitasking environment. The major extensions to the file system are:

- File locking. Files opened under Concurrent CP/M cannot be opened or deleted by other tasks. This feature prevents accidental conflicts with other tasks.

- Shared access to files. As a special option, independent users can open the same file in shared or unlocked mode. Concurrent CP/M supports record locking and unlocking commands for files opened in this mode, and protects files opened in shared mode from deletion by other tasks.

- Date Stamps. The BDOS optionally supports two time and date stamps, one recording when a file is updated, and the other recording when the file was created or last accessed.

- Password Protection. The password protection feature is optional at either the file or drive level. The operator or applications program assigns disk drive passwords, while application programs can assign file protection passwords in several modes.

- Extended Error Module. Besides the default error mode, Concurrent CP/M has two optional error-handling modes that return an error code to the calling process, in the event of an unrecoverable disk error.

1.6 Character I/O module (CIO)

The Character I/O module handles all console and list I/O. Under Concurrent CP/M, every character I/O device is associated with a data structure called a Console Control Block (CCB) or a List Control Block (LCB). These data structures reside in the XIOS. The CCB contains the current owner, status information, line editing variables, and the root of a linked list of Process Descriptors (PDs) that are waiting for access. More than one process can wait for access to a single console. These processes are maintained on a linked list of Process Descriptors in priority order. The LCBs contain similar information about the list devices. See the "Concurrent CP/M System Guide" for more information about LCBs and CCBs.

1.7 Virtual console screen management

Virtual console screen management is coordinated by four separate modules: the CIO, the PIN (Physical INput) and VOUT (Virtual OUTput) processes, and the

XIOS. The line editing associated with the C_READSTR call is performed in the CIO. The PIN process handles keyboard input for all the virtual consoles; it also traps and implements the Ctrl-C, Ctrl-S, Ctrl-Q, Ctrl-P, and Ctrl-O functions. The VOUT process spools console output from processes running on background buffered mode consoles, and handshakes with the PIN process to display spooled console output when the background console is brought to the foreground. The XIOS decides which special keys represent the virtual consoles, and returns a special code from IO_CONIN when you request a screen switch. The XIOS also implements any screen saving and restoring when screens are switched. See the "Concurrent CP/M System Guide" and the discussion of the IO_SWITCH function.

The PIN process reads the keyboard by directly calling the XIOS IO_CONIN function. This is the only place in the operating system IO_CONIN is called. The PIN scans the input stream from the keyboard for switch screen requests and the special function keystrokes Ctrl-C, Ctrl-S, Ctrl-Q, Ctrl-P, and Ctrl-O. All other keyboard input is written to the VINQ (Virtual console INput Queue) associated with the foreground virtual console. The data in the VINQ becomes a type-ahead buffer for each virtual console, and is returned to the process attached to that console as it performs console input.

When PIN sees a Ctrl-C, it calls P_ABORT to abort the process attached to the virtual console, flushes the type-ahead buffer in the VINQ, turns off Ctrl-S, and performs a DRV_RESET call for each logged-in drive. The P_ABORT call succeeds when the Process Keep flag is not ON, saving the Terminal Message Processes (refer to P_CREATE for information on the Process Descriptor). The DRV_RESET calls affect only the removable media drives, as specified in the CKS field of the Disk Parameter Blocks in the XIOS (refer to the "Concurrent CP/M System Guide" for further details on Disk Parameter Blocks).

Ctrl-S stops any output to the screen. Ctrl-S stays set when a virtual console is switched to the background.

Ctrl-O discards any console output to the virtual console. Ctrl-O is turned off when any other key is subsequently pressed, except for the keys representing the virtual consoles.

Ctrl-P echoes console output to the default list device specified in the LIST field of the process descriptor attached to the virtual console. If the list device is attached to a process, a "PRINTER BUSY" message appears.

All of the above control keys can be disabled by the C_MODE call. When one of the above control characters is disabled with C_MODE or when the process owning the virtual console is using the C_RAWIO call, the PIN does not act on the control character, but instead writes it to the VINQ. It is thus possible to read any of the above control characters from an application program. These control keys are discussed in depth in the "Concurrent CP/M User's Guide".

1.8 Extended Input/Output System (XIOS)

The XIOS module is similar to the CP/M-86 Basic Input/Output System (BIOS) module, but it is extended in several ways. Primitive operations, such as console I/O, are modified to support multiple virtual consoles. Several new primitive system calls, such as DEV_POLL, support Concurrent CP/M's additional features, including elimination of wait loops for real-time I/O operations.

1.9 Terminal Message Processes (TMP)

The Concurrent CP/M Terminal Message Processes (TMP) are resident system processes that accept command lines from the virtual consoles, and call the Command Line Interpreter (CLI) to execute them. The TMP prints the prompt on the virtual consoles.

Each virtual console has an independent TMP defining that console's environment, including default disk, user number, printer, and console.

1.10 Transient programs

Under Concurrent CP/M, a transient program is one that is not system-resident. The system must load such programs from disk into available memory each time they execute. The command file of a transient program is identified by the filetype CMD. When you enter a command at the console, the operating system searches on disk for the appropriate CMD file, loads it, and initiates it. Concurrent CP/M supports three different execution models for transient programs: the 8080 Model, the Small Model, and the Compact Model. Sections 4.1.1 through 4.1.3 describe these models in detail.

1.11 System call calling conventions

When a Concurrent CP/M process makes a system call, it loads values into the registers shown in Table 1-1, and initiates Interrupt 224 (via the INT 224 instruction), reserved by the Intel Corporation for this purpose.

```
     Table 1-1. Registers used by system calls

     Entry Parameters:
        Register CL: System call number
                 DL: Byte parameter
             or DX: Word parameter
             or     Address: Offset
                 DS: Address: Segment

     Returned  Values:
        Register AL: Byte return
             or AX: Word return
             or     Address: Offset
                 ES: Address: Segment
                 BX: Same as AX
                 CX: Error code
```

Concurrent CP/M preserves the contents of registers SI, DI, BP, SP, SS, DS, and CS through the operating system calls. The ES register is preserved when it is not used to hold a return segment value. Error codes returned in CX are shown in Table 6-5, "CX error codes".

1.12 SYSDAT: System status

The SYSTAT utility is a development tool that shows the internal state of Concurrent CP/M. SYSTAT describes memory allocation, current processes, system queue activity, and many informative parameters associated with these system data structures. Furthermore, SYSTAT presents two views: either a static snapshot of system activity, or a continuous, real-time window into Concurrent CP/M.

You can specify SYSTAT in one of two modes. If you know which display you want, you can specify it in the invocation, using an option shown in the menu below. If you do not specify an option, select a display from this menu by typing:

```
      A>systat <cr>
```

The screen clears, and the main menu appears:

```
      Which Option?

        H(elp)
        M(emory)
        O(verview)
        P(rocesses - All)
        Q(ueues)
        U(ser Processes)
        C(onsoles)
        E(xit)
      ->_
```

Press the appropriate letter to obtain a display.

When you select H(elp), the HELP file demonstrates the proper syntax and available syntax:

```
      To use SYSTAT with the menu: At the system prompt type SYSTAT <CR>

      To use SYSTAT without the menu: At the system prompt type the command

            SYSTAT [option] -or-
            SYSTAT [option C] -or-
            SYSTAT [option C ##]

      -where-

      -> option =
         M(emory) P(rocesses) O(verview) C(onsoles)
```

```
           U(ser) P(rocesses) Q(ueues) H(elp)

        -> C = Continuous display
          ## = 1-2 digits indicating the period,
             in seconds, between display refreshes.

        Type any letter to return to the menu.
```

The M, P, Q, and U and C options ask you if you prefer a continuous display. If you type "y", Concurrent CP/M asks for a time interval, in seconds, and then displays a real-time window of information. If you type "n", a static snapshot of the requested information appears. In either case, press any key to return to the menu.

The M(emory) option displays all memory potentially available to you, but it does not display restricted memory. The partitions are listed in memory-address order. Length parameter is shown in paragraph values.

The O(verview) option displays an overview of the system parameters, as specified at system generation time. The display is not continuous.

The P(rocess) option displays all system processes, and the resources they are using.

The Q(ueues) option displays all system queues, listing queue readers, writers, and owners.

The U(ser Processes) option displays only user-initiated processes, in the same format as the P(rocess) option.

The C(onsoles) option displays console information; that is, background, foreground, buffered, suspended, purging, Ctrl-Q, and so on.

The E(xit) option returns you to system level from the menu, as does Ctrl-C.

EOF

CCPMPRG2.WS4   (Concurrent CP/M Programmer's Reference Guide, Chapter 2)

(Retyped by Emmanuel ROCHE.)

Section 2: The Concurrent CP/M file system

The Basic Disk Operating System (BDOS) file system supports from one to sixteen logical drives. Each logical drive has two regions: a directory area and a data area. The directory area defines the files that exist on the drive and identifies the data area space that belongs to each file. The data area contains the file data defined by the directory.

The directory area consists of sixteen logically independent directories. These directories are identified by user numbers 0 through 15. During execution, a process runs with a system parameter called the user number set to a single value. The user number specifies the current active directories for all drives on the system. For example, the Concurrent CP/M DIR utility displays only files within a directory selected by the current user number.

The file system automatically allocates directory and data area space when a process creates or extends a file, and returns previously allocated space to free space when a process deletes or truncates a file. If no directory or data space is available for a requested operation, the BDOS returns an error code to the calling process. The allocation and retrieval of directory and data space is transparent to the calling process. As a result, you need not be concerned with directory and drive organization when using the file system calls.

An eight-character filename and a three-letter filetype field identify each file in a directory. Together, these fields must be unique for each file within a directory. However, files with the same filename and filetype can reside in different user directories without conflict. Processes can also assign an eigth-character password to a file, to protect it from unauthorized access.

All system calls that involve file operations specify the requested file by filename and filetype. For some system calls, multiple files can be specified by a technique called "ambiguous reference". This technique uses question marks and asterisk as wildcards characters to give the file system a pattern to match as it searches a directory.

The file system supports two categories of system calls: file-access system calls and drive-related system calls. The file-access system calls have mnemonics beginning with "F_", and the drive-related system calls have mnemonics beginning with "DRV_". The next two sections introduce the file system calls.

2.1.1 File-access system calls

Most of the file-access system calls can be divided into two groups: system calls that operate on files within a directory and system calls that operate on records within a file. However, the file-access category also includes several miscellaneous functions that either affect the execution of other file-access system calls or are commonly used with them.

System calls in the first file-access group include calls to search for one or more files, delete one or more files, rename or truncate a file, set file attributes, assign a password to a file, and compute the size of a file. Also included in this group are system calls to open a file, to create a file, and to close a file.

The second file-access group includes system calls to read or write records to a file, either sequentially or randomly, by record position. BDOS read and write system calls transfer data in 128-byte units, which is the basic record size of the file system. This group also includes system calls to lock and unlock records, and thereby allows multiple processes to have coordinated access to records within a commonly accessed file.

Before making read, write, lock, or unlock system calls for a file, you must first open or create the file. Creating a file has the side effect of opening the file for record access. In addition, because Concurrent CP/M supports three different modes of opening files (Locked, Unlocked, and Read-Only), there can be other restrictions on system calls in this group that are related to the open mode. For example, you cannot write to a file that you have opened in Read-Only mode.

After a process has opened a file, access to the file by other processes is restricted until the file is closed. Again, the exact nature of the restrictions depends on the open mode. However, in all cases, the file system does not allow a process to delete, rename, or change a file's attributes if another process has opened the file. Thus, the F_CLOSE system call performs two steps to terminate record access to a file. It permanently records the current status of the file in the directory, and removes the open-file restrictions limiting access to the file by other processes.

The miscellaneous file-access system calls include calls to set the current user number, set the DMA address, parse an ASCII file specification, and set a default password. This group also includes system calls to set the BDOS Multisector Count and the BDOS Error Mode. The BDOS Multisector Count determines the number of 128-byte records to be processed by the read, write, lock, and unlock system calls. The Multisector Count can range from 1 to 128; the default value is one. The BDOS Error Mode determines whether the file system intercepts certain errors or returns on all errors to the calling process.

2.1.2 Drive-related system calls

BDOS drive-related system calls select the default drive, compute a drive's free space, interrogate drive status, and assign a directory label to a drive. A drive's directory label controls whether the file system enforces file password protection for files in the directory. It also specifies whether the file system is to perform date and time stamping of files on the drive.

This category also includes system calls to reset specified drives, and to control whether other processes can reset particular drives. When a drive is reset, the next operation on the drive re-activates it by logging it in. Logging in a drive initializes the drive for directory and file operations. The purpose of a drive reset call is to prepare for a media change on drives that support removable media. Under Concurrent CP/M, drive reset calls are conditional. A process cannot reset a drive if another process has a file open on the drive.

The following table summarizes the BDOS file system calls.

Table 2-1. File system calls

Mnemonic	Description
DRV_ACCESS	Access drive
DRV_ALLOCVEC	Get drive allocation vector
DRV_ALLRESET	Reset all drives
DRV_DPB	Get Disk Parameter Block address
DRV_GET	Get default drive
DRV_GETLABEL	Get Directory Label
DRV_FLUSH	Flush data buffers
DRV_FREE	Free drive
DRV_LOGINVEC	Return Logged In Vector
DRV_RESET	Reset drive
DRV_ROVEC	Return R/O Vector
DRV_SETLABEL	Set Directory Label
DRV_SET	Set (select) drive
DRV_SETRO	Set drive to Read-Only
DRV_SPACE	Get free space on drive
F_ATTRIB	Set file's attributes
F_CLOSE	Close file
F_DELETE	Delete file
F_DMASEG	Set DMA Segment
F_DMAGET	Get DMA Segment
F_DMAOFF	Set DMA Offset
F_ERRMODE	Set BDOS Error Mode
F_LOCK	Lock record in file
F_MAKE	Make a new file
F_MULTISEC	Set BDOS Multisector Count
F_OPEN	Open file
F_PARSE	Parse filename
F_PASSWD	Set default password
F_RANDREC	Return record number for file Read-Write
F_READ	Read record sequentially from file
F_READRAND	Read random record from file
F_RENAME	Rename file
F_SIZE	Compute file size
F_SFIRST	Directory search first
F_SNEXT	Directory search next

F_TIMEDATE	Return file time/date stamps password mode
F_TRUNCATE	Truncate file
F_UNLOCK	Unlock record in file
F_USERNUM	Set/Get directory User Number
F_WRITE	Write record sequentially into file
F_WRITERAND	Write random record into file
F_WRITEXFCB	Write file's XFCB
F_WRITEZF	Write random record with Zero Fill

The following section contains information on important topics related to the file system. Read these sections carefully before attempting to use the system calls described individually in Section 6, "System calls".

2.2 File naming conventions

Under Concurrent CP/M, a file specification consists of four parts: a drive specifier, the filename field, the filetype fields, and the file password field. The general format for a command line specification is shown below:

```
{d:}filename{.typ}{;password}
```

The drive specifier field specifies the drive where the file is located. The filename and filetype fields identify the file. The password field specifies the password if a file is password-protected.

The drive, type, and password fields are optional, and delimiters are required only specifying their associated fields. The drive specifier can be assigned a letter from A to P, where the actual drive letters supported on a given system are determined by the XIOS implementation. When the drive letter is not specified, the current default drive is assumed.

The filename and password fields can contain one to eight non-delimiter characters. The filetype field can contain one to three non-delimiter characters. All three fields are left justified and padded with blanks, if necessary. Omitting the optional type or password fields implies a field specification of all blanks.

Under Concurrent CP/M, the P_CLI system call interprets ASCII command lines and loads programs. The P_CLI system call makes F_PARSE system calls to parse file specifications from a command line. F_PARSE recognizes certain ASCII characters as delimiters when it parses a file specification. These characters are shown in Table 2-2.

Table 2-2. Valid filename delimiters

ASCII	Hex equivalent	
null	00h	
space	20h	
ENTER	0Dh	("<--+" key)
tab	09h	("-->|" key)
:	3Ah	(colon)

.	2Eh	(full stop)
;	3Bh	(semicolon)
=	3Dh	(equal sign)
,	2Ch	(comma)
[	5Bh	(left square bracket)
]	5Dh	(right square bracket)
<	3Ch	(less sign)
>	3Eh	(more sign)
|	7Ch	(vertical bar)

The F_PARSE system call also excludes all control characters from the file specification fields, and translate all lowercase letters to uppercase.

Avoid using parentheses and the backslash character, "\", in the filename and filetype fields, because they are commonly used delimiters. Use asterisk and question mark characters, "*" and "?", only to make an ambiguous file reference. When F_PARSE encounters an asterisk in a filename or filetype field, it pads the remainder of the field with question marks. For example, a filename of "X*.*" is parsed to "X???????.???". The BDOS F_SFIRST, F_SNEXT, and F_DELETE system calls match a question mark in the filename or filetype fields to the corresponding position of any directory entry belonging to the current user number. Thus, a search operation for "X???????.???" finds all the files in the current user directory beginning with a "X". Most other file-access BDOS system calls treat the presence of a question mark in the filename or filetype fields as an error.

It is not mandatory to follow the file naming conventions of Concurrent CP/M when you create or rename a file with BDOS system calls directly from an application program. However, the conventions must be used if the file is to be accessed from a command line. For example, the P_CLI system call cannot locate a command file in the directory if its filename or filetype field contains a lowercase letter.

As a general rule, the filetype field names the generic category of a particular file, and the filename field distinguishes individual files within each category. Although they are generally arbitrary, Table 2-3 lists come of the generic filetype categories that have been established.

Table 2-3. Filetype conventions

Filetype	Description
A86	8086 assembler source file
ASC	ASCII characters file
ASM	8080 assembler source file
BAK	Back-up file
BAS	BASIC source file
BSX	BASEX source file
CBL	COBOL source file
CMD	8086 command file
COM	8080 command file
CON	CCP/M Module
DAT	Data file
DRD	DR DRAW file

DRG	DR GRAPH file
FCL	FOCAL source file
H86	8086 hex file
HEX	8080 hex file
INT	Intermediate file
L86	8086 library file
LIB	8080 library file
LSP	Lisp source file
LOG	Logo source file
LST	List file
PAS	Pascal source file
PLI	PL/I source file
PRL	Page ReLocatable file
PRN	Printer file
REL	Relocatable file
RSP	Resident System Process
SPR	System Page Relocatable
STC	STOIC source file
SUB	SUBMIT file
SYM	Symbol file
SYS	System file
TEX	TEX formatter source file
WS4	WordStar 4 word processor filec
$$$	Temporary file

2.3 Disk drive and file organization

The file system can support up to 16 logical drives, identified by the letters A through P. A logical drive usually corresponds to a physical drive on the system, particularly for physical drives that support removable media such as floppy disks. High-capacity hard disks, however, are commonly divided into multiple logical drives. If a disk contains system tracks reserved for the boot loader, these tracks precede the tracks of the disk mapped by the logical drive. In this manual, references to drives mean logical drives, unless explicitly stated otherwise.

The maximum file size supported on a drive is 32 megabytes. The maximum capacity of a drive is determined by the data block size specified for the drive in the XIOS. The data block size is the basic unit in which the BDOS allocates space to files. Table 2-4 displays the relationship between data block size and total drive capacity.

Table 2-4. Drive capacity

Data block size	Maximum drive capacity
1 KB	256 KB
2 KB	64 MB
4 KB	128 MB
8 KB	256 MB
16 KB	512 MB

Each drive is divided into two regions: a directory area and a data area. The directory area contains from one to sixteen blocks located at the beginning of the drive. The actual number is set in the XIOS. Directory entries residing in this area define the files that exist on the drive. In addition, the directory entries belonging to a file identify the data blocks in the drive's data area that contains the file's records. The directory area is logically subdivided into sixteen independent directories identified as user 0 through 15. Each independent directory shares the actual directory area on the drive.

Each disk file may consist of a set of up to 262,144 (40000h) 128-byte records. Each record of a file is identified by its position in the file. This position is called the record's Random Record Number. If a file is created sequentially, the first record has a position of zero, while the last record has a position one less than the number of records in the file. Such a file can be read sequentially, beginning at record zero, or randomly by record position. Conversely, if a file is created randomly, records are added to the file by specific position. A file created in this way is called "sparse" if positions exist within the file where a record has not been written.

The BDOS automatically allocates data blocks to a file, to contain the file's records on the basis of the record positions consumed. Thus, a sparse file that contains two records, one at position zero, the other at position 262,143, consumes only two data blocks in the data area. Sparse files can be created and accessed only randomly, not sequentially. Note that any data block allocated to a file is permanently allocated until the file is deleted or truncated. These are the only mechanisms supported by the BDOS for releasing data blocks belonging to a file.

Source files under Concurrent CP/M are treated as a sequence of ASCII characters, where each line is followed by a carriage return/line-feed sequence, 0Dh followed by 0Ah. Thus, a single 128-byte record could contain several lines of source text. The end of an ASCII file is denoted by a Ctrl-Z character (1Ah), or a real end-of-file, returned by the BDOS read system call. Note that these source file conventions are not supported in the file system directly, but are followed by Concurrent CP/M utilities such as TYPE and ASM-86. In addition, Ctrl-Z characters embedded within other types of files, such as CMD files, do not signal end-of-file.

2.4 File Control Block definition

The File Control Block (FCB) is a system data structure that serves as an important channel for information exchange between a process and BDOS file-access system calls. A process initializes an FCB to specify the drive location, filename and filetype fields, and other information that is required to make a file-access call. For example, in an F_OPEN system call, the FCB specifies the name and location of the file to be opened. In addition, the file system uses the FCB to maintain the current state and record position of an open file. Some file-access system calls use special fields within the FCB for invoking options. Other file-access system calls use the FCB to return data to the calling program. All BDOS random I/O system calls require the calling process to specify the Random Record Number in a 3-byte field at the end of the FCB.

When a process makes a BDOS file-access system call, it passes an FCB address to the BDOS. This address has two 16-bit components: register DX (which contains the offset) and register DS (which contains the segment). The length of the FCB data area depends on the BDOS system call. For most system calls, the minimum length is 33 bytes. For the F_READRAND, F_WRITERAND, F_WRITEZF, F_LOCK, F_UNLOCK, F_RANDREC, F_SIZE, and F_TRUNCATE system calls, the minimum FCB length is 36 bytes. When the F_OPEN or F_MAKE system calls open a file in Unlocked mode, the FCB must bet at least 35 bytes long. Figure 2-1 displays the FCB data structure in two formats.

```
+----+------+------+----+----+----+----+--------+----+----+----+----+
| DR | NAME | TYPE | EX | CS | RS | RC | D0-D15 | CR | R0 | R1 | R2 |
+----+------+------+----+----+----+----+--------+----+----+----+----+
  00   01...  09...  12   13   14   15   16...    32   33   34   35

    +----+----+----+----+----+----+----+----+
00h | DR | F1   F2   F3   F4   F5   F6   F7 |
    +----+----+----+----+----+----+----+----+
08h | F8 | T1   T2   T3 | EX | CS | RS | RC |
    +----+----+----+----+----+----+----+----+
10h | D0   D1   D2   D3   D4   D5   D6   D7 |
    +----+----+----+----+----+----+----+----+
18h | D8   D9   D10  D11  D12  D13  D14  D15|
    +----+----+----+----+----+----+----+----+
20h | CR | R0 | R1 | R2 |
    +----+----+----+----+
```

Figure 2-1. FCB -- File Control Block

The fields in the FCB are defined as follows:

Table 2-5. FCB field definitions

Format: Field
Definition

DR
Drive Code (0-16).
0 -> use default drive for file
1 -> auto disk select drive A
2 -> auto disk select drive B
16 -> auto disk select drive P

F1...F8
Contains the filename in ASCII uppercase, with high bit = 0, F1', ..., F8' denote the high-order bit of these positions, and called "attribute bits".

T1,T2,T3
Contains the filetype in ASCII uppercase, with high bit = 0, T1', T2', and T3' denote the high-order bit of these positions, and are also called "attribute bits".

T1' = 1 -> Read-Only file,
T2' = 1 -> System file,
T3' = 1 -> File has been archived.

EX
Contains the current extent number. This field is initialized to 0 by the calling process, but it can range from 0 to 31 during file I/O.

CS
Contains the FCB checksum value for open FCBs.

RS
Reserved for internal system use.

RC
Record count for extent EX. This field takes on values from 0 to 255 (values greater than 128 imply a record count of 128).

D0...D15
Normally, filled in by Concurrent CP/M, and reserved for system use. Also used to specify the new filename and filetype, with the F_RENAME system call.

CR
Current record to read or write in a sequential file operation. This field is normally set to zero by the calling process when a file is opened or created.

R0,R1,R2
Optional Random Record Number, in the range 0-262,143 (0-3FFFFh). R0,R1,R2 constitute an 18-bit value, with low byte R0, middle byte R1, and high byte R2.

Note: The 2-byte File ID is returned in bytes R0 and R1 of the FCB when a file is successfully opened in Unlocked mode (refer to Section 2.10, "File security").

2.4.1 FCB initialization and usage

The calling process must initialize bytes 0 through 11 of the referenced FCB before making the following file-access system calls: F_ATTRIB, F_DELETE, F_MAKE, F_OPEN, F_RENAME, F_SFIRST, F_SIZE, F_SNEXT, F_TIMEDATE, F_TRUNCATE, and F_WRITEXFCB. Normally, the DR field specify the drive location of the file, and the name and type fields specify the name of the file. You must also set the EX field of the FCB before calling F_MAKE, F_OPEN, F_SFIRST, and F_WRITEXFCB. Except for the F_WRITEXFCB system call, you can usually set this field to zero. Note that the F_RENAME system call requires the calling process to place the new filename and filetype in bytes D1 through D11.

The remaining file-access calls that use FCBs require an FCB that has been initialized by a prior file-access system call. For example, the F_SNEXT system call expects an FCB initialized by a prior F_SFIRST call. In addition, the F_LOCK, F_READ, F_READRAND, F_UNLOCK, F_WRITERAND, and F_WRITEZF system calls require an FCB that has been activated for record operations. Under

Concurrent CP/M, only the F_OPEN and F_MAKE system calls can activate an FCB.

If you intend to process a file sequentially from the beginning, using the
F_READ and F_WRITE system calls, you must set the CR field to zero before you
make your first read or write call. In addition, when you make an F_LOCK,
F_READRAND, F_UNLOCK, F_WRITERAND, or F_WRITEZF system call, you must set
bytes R0 through R2 of the FCB to the requested Random Record Number. The
F_TRUNCATE system call also requires the FCB random record field to be
initialized.

The F_SFIRST, F_SNEXT, and F_DELETE system calls support multiple or ambiguous
reference. In general, a question mark in the filename, filetype, or EX fields
matches all values in the corresponding positions of directory entries during
a directory search operation. File directory entries maintained in the
directory area of each disk drive have the same format as FCBs, except for
byte 0, which contains the file's user number, and bytes 32 through 35, which
are not present. The search system calls, F_SFIRST and F_SNEXT, also recognize
a question mark in the FCB DR field, and, if specified, they return all
directory entries on the disk, regardless of user number, including empty
entries. A directory FCB that begins with 0E5h is an empty or erased directory
entry.

When the F_OPEN and F_MAKE system calls activate an FCB for record operations,
they copy the FCB's matching directory entry from disk, excluding byte 0, into
the FCB in memory. In addition, these system calls compute and store a
checksum value in the CS field of the FCB. During subsequent record operations
on the file, the file system uses this checksum field to verify that the FCB
has not been modified by the calling process in an illegal way. Thus, all
read, write, lock, and unlock operations on a file must specify a valid
activated FCB; otherwise, the BDOS returns a checksum error. The BDOS performs
this checking to protect the integrity of the file system. In general, you
should not modify bytes 0 through 31 of an open FCB, except to set interface
attributes (see Section 2.4.3, "Interface attributes"). Other restrictions
related to activated FCBs are discussed in Section 2.10, "File security".

The BDOS updates the memory copy of the FCB during file processing, to
maintain the current position within the file. During file operations, the
BDOS also updates the memory copy of the FCB, to close the allocation of data
blocks to the file. At the termination of file processing, the F_CLOSE system
call permanently records this information on disk.

Note that the BDOS does not record the data blocks allocated to a file during
write operations in the disk directory until the calling process issues an
F_CLOSE call. Therefore, a process that creates or modifies files must close
the files at the termination of files processing. Otherwise, data might be
lost.

2.4.2 File attributes

The high-order bits of the FCB filename (F1', ..., F8') and filetype fields
(T1', T2', T3') are called "attribute bits". Attribute bits are 1-bit Boolean
fields, where "1" indicates "ON" or "TRUE", and "0" indicates "OFF" or

"FALSE". Attribute bits indicate two kinds of attributes within the file system: file attributes, and interface attributes. The file attributes are described in this section. Section 2.4.3 describes interface attributes.

The file attribute bits, F1', ..., F4', and T1', T2', T3', indicate that a file has a defined attribute. These bits are recorded in a file's directory FCB. File attributes can be set or reset only by the F_ATTRIB system call. When the F_MAKE system call creates a file, it initializes all file attributes to zero. A process can interrogate file attributes in an FCB activated by the F_OPEN system call, or in directory FCBs returned by the F_SFIRST and F_SNEXT system calls.

Note: The file system ignores the file attribute bits when it attempts to locate a file in the directory.

The file system defines file attributes T1', T2', and T3' as follows:

Table 2-6. File Attribute definitions

Format: Attribute
Definition

T1': Read-Only attribute
This attribute, if set, prevents write operations to a file.

T2': System attribute
This attribute, if set, identifies the file as a Concurrent CP/M system file. The Concurrent CP/M DIR utility does not usually display System files. In addition, user-zero system files can be accessed on a Read-Only basis from other user numbers.

T3': Archive attribute
User-written archive programs use this attribute. When an archive program copies a file to back-up storage, it sets the archive attribute of the copied files. The file system automatically resets the archive attribute of a directory entry when writing to the directory entry's region of a file. An archive program can test this attribute in each of the file's directory entries, using the F_SFIRST and F_SNEXT system calls. If all directory entries have the archive attribute set, the file has not been modified since the previous archive. The Concurrent CP/M PIP utility supports file archiving.

File attributes F1' through F4' of command files are defined as Compatibility Attributes under Concurrent CP/M (see Section 2.12, "Compatibility attributes"). However, for all other files, attributes F1' through F4' are available for definition by the user.

2.4.3 Interface attributes

The interface attributes are F5', F6', F7', and F8'. These attributes cannot be used as file attributes. Interface attributes F5' and F6' request options for BDOS file-access system calls. Table 2-7 lists the F5' and F6' attribute definitions for the system calls that define interface attributes. Note that the F5' = 0 and F6' = 0 definitions are not listed if their definition simply implies the absence of the associated option.

Table 2-7. BDOS interface attributes F5' and F6'

Format: System call	Attribute
F_ATTRIB	
	F5' = 1 : Maintain extended file lock
	F6' = 1 : Set file byte count
F_CLOSE	
	F5' = 1 : Partial close
	F6' = 1 : Extend file lock
F_DELETE	
	F5' = 1 : Delete file XFCBs only, and maintain extended file lock
F_LOCK	
	F5' = 0 : Exclusive Lock
	F5' = 1 : Shared Lock
	F6' = 0 : Lock existing records only
	F6' = 1 : Lock logical records
F_MAKE	
	F5' = 0 : Open in Locked mode
	F5' = 1 : Open in Unlocked mode
	F6' = 1 : Assign password to file
F_OPEN	
	F5' = 0 : Open in Locked mode
	F5' = 1 : Open in Unlocked mode
	F6' = 0 : Open in mode specified by F5'
	F6' = 1 : Open in Read-Only mode
F_RENAME	
	F5' = 1 : Maintain extended file lock
F_TRUNCATE	
	F5' = 1 : Maintain extended file lock
F_UNLOCK	
	F5' = 1 : Unlock all locked records

Section 6, "System calls", details the above interface attribute definitions for each of the preceding system calls. Note that the BDOS always resets interface attributes F5' and F6' before returning to the calling process. Interface attributes F7' and F8' are reserved for internal use by the file system.

2.5 User Number conventions

The Concurrent CP/M user facility divides each drive directory into sixteen logically independent directories, designated as "user 0" through "user 15". Physically, all user directories share the directory area of a drive. In most other aspects, however, they are independent. For example, files with the same name can exist on different user numbers of the same drive with no conflicts. However, a single file cannot extend across more than one user number.

Only one user number is active for a specific process at one time. For this process, the current user number applies to all drives on the system. Furthermore, the FCB format does not contain a field that can override the current user number. As a result, all file and directory operations reference only directory entries associated with the current user number.

However, it is possible for a process to access files on different user numbers, by setting the user number to the file's user number with the F_USERNUM system call before issuing the BDOS call. However, if a process attempts to read or write to a file under a user number different from the user number that was active when the file was opened, the file system returns an FCB checksum error.

When the P_CLI system call initiates a transient process or Resident System Process (described in detail in Section 5, "Resident System Processes generation"), it sets the user number to the default value established by the process issuing the P_CLI system call. The sending process is usually the TMP. However, the sending process can be another process, such as a transient program that makes a P_CHAIN call. A transient process can change its user number by making an F_USERNUM call. Changing the user number in this way does not affect the command line user number displayed by the TMP. Thus, when a transient process that has changed its user number terminates, the TMP restores and displays the original user number in the command line prompt when it regains control.

User 0 has special properties under Concurrent CP/M. The file system automatically opens files listed under user zero but requested under another user number if the file is not present under the current user number, and if the file on user zero has the system attribute (T2') set. This convention allows utilites, including overlays and any other commonly accessed files, to reside on user zero, but remains available to other users. This eliminates the need to copy commonly used utilities to all user numbers on a directory, and gives the Concurrent CP/M manager control over which files are directly accessible to the different user areas.

2.6 Directory Labels and XFCBs

The file system includes three special types of FCBs: the directory label, the XFCB (described in this section), and the SFCB (described in detail in Section 2.8, "File date and time stamps: SFCBs").

The directory label specifies for its drive whether password support is to be activated, and if date and time stamping for files is to be performed. The format of the directory label is shown below in Figure 2-2.

```
+----+------+------+----+----+----+----+----------+-----+-----+
| DR | Name | Type | DL | S1 | S2 | RC | Password | TS1 | TS2 |
+----+------+------+----+----+----+----+----------+-----+-----+
  00   01...  09...  12   13   14   15   16...      25... 29...
```

Figure 2-2. Directory Label format

The fields in the Directory Label are defined as follows:

Table 2-8. Directory Label field definitions

Format: Field
Definition

DR
Drive Code (0-16).
0 -> use default drive for file
1 -> auto disk select drive A
2 -> auto disk select drive B
16 -> auto disk select drive P

Name
Directory Label name.

Type
Directory Label type.

DL
Directory Label data byte.
Bit 7 : Enable password support
Bit 6 : Perform access time stamping
Bit 5 : Perform update time stamping
Bit 4 : Peform create time stamping
Bit 0 : Directory Label exists
(Bit references are right to left, relative to 0.)

S1, S2, RC
Reserved for system use.

Password
8-byte password field (encrypted).

TS1
4-byte creation time stamp field.

TS2
4-byte update time stamp field.

Only one directory label can exist in a drive's directory area. The directory label name and type fields are not used to search for a directory label; they can be used to identify a disk.

You can use the DRV_SETLABEL system call to create a directory label or update its fields. This system call can also assign a password to a directory label. The directory label password, if assigned, cannot be circumvented, whereas file password protection on a drive is an option controlled by the directory label. Thus, access to the directory label password provides the ability to bypass password protection on the drive.

Note: The file system provides no specific system call to read the directory label FCB directly. However, you can read the directory label data byte directly with the BDOS system call DRV_GETLABEL. In addition, you can use the BDOS search system calls F_SFIRST and F_SNEXT to find a directory label. You can identify the directory label by a value of 32 (20h) in byte 0 of the directory FCB.

The XFCB is an Extended FCB that can optionally be associated with a file in the directory. If present, it contains the file's password and password mode. The format of the XFCB is shown below in Figure 2-3.

```
+----+------+------+----+----+----+----+----------+-----+------+
| DR | Name | Type | PM | S1 | S2 | RC | Password |  RESERVED  |
+----+------+------+----+----+----+----+----------+-----+------+
  00   01...  09...  12   13   14   15   16...      25... 29...
```

Figure 2-2. Directory Label format

The fields in the XFCB are defined as follows:

Table 2-9. XFCB field definitions

Format: Field
Definition

DR
Drive Code (0-16).
0 -> use default drive for file
1 -> auto disk select drive A
2 -> auto disk select drive B
16 -> auto disk select drive P

Name
Filename field.

Type
Filetype field.

PM
Password Mode.
Bit 7 : Read mode
Bit 6 : Write mode

Bit 5 : Delete mode
(Bit references are right to left, relative to 0.)

S1, S2, RC
Reserved for system use.

Password
8-byte password field (encrypted).

Reserved
8-byte area reserved for future use.

An XFCB can be created only on a drive that has a directory label, and only if the directory label enables password protection. For drives in this state, there are two ways to create an XFCB for a file: with the F_MAKE system call or the F_WRITEXFCB system call. The F_MAKE system call creates an XFCB if the calling process requests that a password be assigned to the created file. The F_WRITEXFCB system call creates an XFCB when it is called to assign a password to an existing file. You can identify an XFCB in the directory by a value of 16 (10h) + N in byte 0 of the FCB, where N equals the user number.

2.7 File passwords

There are two ways to assign passwords to a file: by the F_MAKE system call or by the F_WRITEXFCB system call. You can also change a file's password or password mode with the F_WRITEXFCB system call if you can supply the original password. Note that you cannot change a file's password or password mode if password protection for the drive is disabled by the directory label. However, even if you cannot supply a file's password, you can delete a file's XFCB, thereby removing its password protection, if password protection is disabled on the drive.

The Concurrent CP/M BDOS provides password protection in one of three modes when password support is enabled by the directory label. Table 2-10 shows the difference in access level allowed to BDOS system calls when the password is not supplied.

Table 2-10. Password protection mode

Mode	Access level allowed without password
(1) Read	Cannot be read, modified, or deleted.
(2) Write	Can be read, but not modified or deleted.
(3) Delete	Can be read and modified, but not deleted.

If a file is password protected in Read mode, a process must supply the password to open the file. Processes cannot write to a file protected in Write mode without the password. A file protected in Delete mode allows read and write access, but a process must specify the password to delete or truncate the file, rename the file, or to modify the file's attributes. Thus, password protection in mode 1 implies mode 2 and 3 protection, and mode 2

protection implies mode 3 protection. All three modes require the user to
specify the password to delete or truncate the file, rename the file, or to
modify the file's attributes.

If a process supplies the correct password or the directory label disables
password protection, then access to the BDOS system calls is the same as for a
file that is not password-protected. In addition, the F_SFIRST and F_SNEXT
system calls are not affected by file passwords. The following BDOS system
calls test for passwords:

```
DRV_SETLABEL
F_ATTRIB
F_DELETE
F_OPEN
F_RENAME
F_WRITEXFCB
F_TRUNCATE
```

The BDOS maintains file passwords in the XFCB and directory label in encrypted
form. To make a BDOS system call for a file that requires a password, a
process must place the password in the first eight bytes of the current DMA,
or make it the default password with the F_PASSWD system call, before making
the system call.

Note: The BDOS maintains the assigned default password for each process.
Child processes inherit the default password of their parent process. You can
set a given TMP's default password using the SET command; all programs loaded
by this TMP inherit the same default password.

2.8 File date and time stamps: SFCBs

The Concurrent CP/M file system uses a special type of directory entry called
an "SFCB" to record date and time stamps for files. When a directory has been
initialized for date and time stamping, SFCBs reside in every fourth position
of the directory. Each SFCB maintains the date and time stamps for the
previous three directory entries, as shown in Figure 2-4.

```
          +----+----------------------------------------+
          |    |            FCB 1                       |
          |    |            FCB 2                       |
          |    |            FCB 3                       |
          +----+-----------+-----------+-----------+----+
          | 21 |  Stamps   |  Stamps   |  Stamps   | // |
          |    | for FCB 1 | for FCB 2 | for FCB 3 | // |
          +----+-----------+-----------+-----------+----+
   Byte #: 0    1           11          21          31   32
```

Figure 2-4. Directory record with SFCB

This figure shows a 128-byte directory record containing an SFCB. Directory

records have four directory entries, each 32 bytes long; SFCBs always occupy
the last 32-byte entry in the directory record.

The SFCB itself contains five fields. The first field is a single byte
containing the value 21h; this field identifies the SFCB within the directory.
The next three fields, called the "SFCB subfields", are each 10 bytes in
length, and contain the date and time stamps for their corresponding FCB
entries in the directory record. The last byte of the SFCB is reserved for
system use. Figure 2-5 shows the detail of the SFCB subfields.

```
          +---------------+---------------+----------+----------+
          | Create/Access |    Update     | Password | Reserved |
          | time and date | time and date |   mode   |          |
          +---------------+---------------+----------+----------+
     Byte #: 0             4               8          9         10
```

Figure 2-5. SFCB subfields

An SFCB subfield only contains valid information if its corresponding FCB in
the directory record is an extent zero FCB. This FCB is a file's first
directory entry. For password protected files, the SFCB subfield also contains
the password mode of the file; the password mode field is zero for files
without password protection. You can read SFCBs by making F_SFIRST and F_SNEXT
system calls. In addition, you can make an F_TIMEDATE system call to retrieve
the date and time stamps and password mode of a specified file. Refer to the
T_GET system call definition in Section 6, "System calls", for the description
of the format of a date and time stamp field.

Concurrent CP/M supports three kinds of file stamping: create, access, and
update. Create stamps record when the file was created, access stamps record
when the file was last opened, and update stamps record the last time the file
was modified. Create and access stamps share the same field. As a result, file
access stamps overwrite any create stamps.

The directory label of a properly initialized disk determines the type of date
and time stamping for files on the drive. The INITDIR utility initializes a
directory for date and time stamping by placing an SFCB in every fourth
directory entry. Disks not initialized in this way cannot support date and
time stamping. In addition, date and time stamping is not performed if the
disk's directory label is absent or does not specify date and time stamping,
or if the disk is Read-Only.

Note that the directory label is also time stamped, but these stamps are not
made in an SFCB; time stamps fields in the last eight bytes of the directory
label show when it was created and last updated. Access stamping is not
supported for directory labels.

The BDOS file system uses the system date and time when it records a date and
time stamp. This value is maintained in a field in the SYSDAT part of the
System Data Segment. The DATE utility sets the system time and date (refer to
the "Concurrent CP/M User's Guide" for details on using DATE).

2.9 File open modes

The file system provides three different modes for opening files. They are defined below.

Locked mode

A process can open a file in Locked mode only if the file is not currently opened by another process, and the file is not a Read-Only file (attribute T1' set). Once open in Locked mode, no other process can open the file until it is closed. Thus, if a process successfully opens a file in Locked mode, that process owns the file until the file is closed or the process terminates. Files opened in Locked mode support read and write operations unless the file is password-protected in Write mode, and the process is issuing the F_OPEN call cannot supply the password. In this case, the BDOS allows only read operations to the file.

If a file opened in Locked mode is a Read-Only file, the F_OPEN system call automatically changes the open mode to Read-Only mode. Read-Only mode is described below.

Note: Locked mode is the Default mode for opening files under Concurrent CP/M.

Unlocked mode

A process can open a file in Unlocked mode if the file is not currently open, or if another process has already opened the file in Unlocked mode. This mode allows more than one process to open the same file. Files opened in Unlocked mode support read and write operations unless the file is a Read-Only file (attribute T1' set) or the file is password-protected in Write mode and the process issuing the F_OPEN call cannot supply the password.

When opening a file in Unlocked mode, a process must reserve 35 bytes in the FCB because the F_OPEN system call returns a 2-byte value called the "File ID" in the R0 and R1 bytes of the FCB. The File ID is a required parameter for the F_LOCK and F_UNLOCK system calls. These BDOS system calls work only for files opened in Unlocked mode.

Read-Only mode

A process can open a file in Read-Only mode if the file is not currently opened by another process or if another process has opened the file in Read-Only mode. This mode allows more than one process to open the same file for Read-Only access.

The F_OPEN system call performs the following steps for files opened in Locked or Read-Only mode. If the current user number is non-zero, and the file to be opened does not exist under the current user number, the F_OPEN system call searches the user zero directory for the file. If the file exists under user zero and has the system attribute T2' set, the BDOS opens the file under user zero. The open mode is automatically forced to Read-Only when this is done.

The F_OPEN and F_MAKE system calls use FCB interface attributes F5' and F6' to specify the open mode. The interface attribute definitions for these functions are listed in Table 2-7.

Note: The F_MAKE system call does not allow opening the file in Read-Only mode.

2.10 File security

In general, the security measures implemented in the file system prevent accidental collisions between running processes. It is not possible to provide total security under Concurrent CP/M because the file system maintains file allocation information in open FCBs in the user's memory regio, and Concurrent CP/M does not require memory protection. However, the file system is designed to ensure that multiple processes can share the same file system without interfering with each other by:

- performing checksum verifications of open FCBs
- monitoring all open files and locked records via the system Lock List.

The BDOS validates the checksum of user FCBs before all I/O operations to protect the integrity of the file system from corrupted FCBs. The F_OPEN and F_MAKE system calls compute and assign checksums to FCBs. The F_READRAND, F_READ, F_WRITERAND, F_WRITEZF, F_WRITE, F_LOCK, and F_UNLOCK system calls subsequently verify and recompute the checksums when they change the FCB. The F_CLOSE system call also verifies FCB checksums. Note that FCB verifications by these system calls can be disabled (see Section 2.12, "Compatibility attributes"), but Concurrent CP/M's file security is reduced when this is done. If the BDOS detects an FCB checksum error, it does not perform the requested command. Instead, it either returns to the calling process with an error code or, if the system call is F_CLOSE and the BDOS Error mode is in the default state (see Section 2.18, "BDOS Error handling"), it terminates the calling process with an error message.

Concurrent CP/M uses a system data structure, called the "Lock List", to manage file opening and record locking by running processes. Each time a process opens a file or locks a record successfully, the file system allocates an entry in the system Lock List to record the fact. The file system uses the following information to:

- prevent a process from deleting, truncating, renaming, or updating the attributes of another process' open file.

- prevent a process from opening a file currently opened by another process, unless both processes open the file in Unlocked or Read-Only mode.

- prevent a process from resetting a drive on which another process has an open file.

- prevent a process from reading, writing, or locking a record currently locked by another process. Refer to Section 2.14, "Concurrent file access", for more information on record locking and unlocking.

The file system only verifies whether another process has the FCB-specified file open for the following file-access system calls: F_OPEN, F_MAKE, F_DELETE, F_RENAME, F_ATTRIB, and F_TRUNCATE. For file-access system calls that require an open FCB, the FCB checksum controls whether the calling process can use the FCB. By definition, a valid FCB checksum implies that the file has been successfully opened and an entry for the file resides in the system Lock List.

The most common way a process releases a lock entry for an open file is by closing the file. A close operation is permanent if it causes the removal of the file's open Lock List entry. The file system invalidates the FCB checksum field on permanent close operations, to prevent continued open file operations with the FCB.

However, not all close operatons are permanent. For example, if a process makes multiple F_OPEN or F_MAKE calls to an open file, a matching number of F_CLOSE calls must be made before the file system permanently closes the file. Of course, if you only open a file once, a single close operation permanently closes the file. In addition, a process can optionally make partial F_CLOSE calls to a file by setting interface attribute F5'. A partial close operation does not affect the open state of a file. In the above example, a partial close operation would not count against an F_OPEN or F_MAKE call. A partial close operation simply updates the directory to reflect the current state of the file.

As a general rule, under Concurrent CP/M a process should close files as soon as it no longer needs them, even if it has not modified them. While a process has a file open, access by other processes to the file is restricted. For example, after a process has opened a file in Locked mode, the file cannot be opened by other processes until the file is closed or the process terminates.

Furthermore, space in the system Lock List is limited. If a process attempts to open a file and no space remains in the system Lock List,or if the process exceeds the open file limit, the BDOS denies the open request and usually terminates the calling process. You can change the way the file system handles this error by making an F_ERRMODE system call. Note that the size of the system Lock List and the process open file limit are GENCCPM parameters.

There are several other situations where the file system removes open file entries from the system Lock List for a process. For example, if a process makes an F_DELETE call for a file it has open in Locked mode, the file system deletes the file and also purges the file's entry from the system Lock List. Deleting an open file is not recommended under Concurrent CP/M but it is supported for files opened in Lock mode, to provide compatibility with software written under earlier releases of MP/M and CP/M. The file system does not allow deletion of a file opened in Unlocked or Read-Only mode.

To ensure that the process does not use the open FCB corresponding to the deleted file, the file system subsequently checks all open FCBs for the process. Each open FCB is checked the next time it is used with a file-access system call that requires an open FCB. If a Lock List entry exists for the file, the BDOS allows the operation to proceed; if not, it indicates that the file has been purged and the file system returns an FCB checksum error.

The file system performs this verification of a process' open FCBs whenever it purges an open file entry from the system Lock List. The following list describes these situations:

- A process makes an F_ATTRIB, F_DELETE, F_RENAME, or F_TRUNCATE system call to a file it has open in Locked mode. These operations cannot be performed on a file open in Unlocked or Read-Only mode.

- A process issues a DRV_FREE call for a drive on which it has an open file.

- The BDOS detects a change in media on a drive that has open files. This is a special case because a process cannot control the occurrence of this situation, and because it can impact more than one process. Refer to Section 2.17, "Reset, access, and free drive", for more details on this situation.

Open FCB verification can affect performance because each verification operation requires a directory search operation. In general, you should avoid such situations when creating new programs for Concurrent CP/M.

2.11 Extended file locking

Extended file locking enables a Concurrent CP/M process to maintain a lock on a file after the file is permanently closed. This facility allows a process to set the attributes, delete, rename, or truncate a file without interference from other processes. In addition, this technique avoids the problems associated with using these system calls on open files (see Section 2.10, "File security").

A process can also re-open a file with an extended lock and continue open file processing. To illustrate how extended file locking might be used, a process can close an open file, rename the file, re-open the file under its new name, and continue with file operations without ever losing the file's Lock List itema and control over the file.

A process can only specify extended file locking for a file it has opened in Locked mode. To extend a file's lock, set interface attribute F6' when closing the file. The F_CLOSE system call interrogates this attribute only when it is closing a file permanently. Thus, interface attribute F5', signifying a partial close, must be reset when the F_CLOSE call is made. In addition, the close operation must be permanent. If a process has opened a file N times, the F_CLOSE system call ignores the F6' attribute until the file is closed for the Nth time.

Note that the access rules for a file with extended lock are identical to the rules for a file open in Locked mode. In addition, you cannot extend the lock of a Read-Only file (attribute T1' set), because a Read-Only file cannot be opened in Locked mode.

To maintain an extended file lock through an F_ATTRIB, F_RENAME, or F_TRUNCATE system call, set interface attribute F5' of the referenced FCB when making the call. The BDOS honors this attribute only if the file has been closed with an extended lock. Setting attribute F5' also maintains an extended file lock for the F_DELETE system call, but setting this attribute also changes the nature of the delete operation to an XFCB-only delete. If successful, all four of these system calls delete a file's extended lock item if they are called with attribute F5' reset. However, the extended lock item is not deleted if they return with an error code.

You can make an F_OPEN call to resume record operations on a file with an extended lock. Note that you can also change the open mode when you re-open the file. The following example illustrates the use of extended locks.

1. Open file EXLOCK.TST in Locked mode.

2. Perform read and write operations on the file EXLOCK.TST using the open FCB.

3. Close file EXLOCK.TST with interface attribute F6' set to retain the file's lock item.

4. Use the F_RENAME system call to change the name of the file to EXLOCK.NEW with interface attribute F5' set to retain the file's extended lock item.

5. Re-open the file EXLOCK.NEW in Locked mode.

6. Perform read and write operations on the file EXLOCK.NEW, using the open FCB.

7. Close file EXLOCK.NEW again with interface attribute F6' set to retain the file's lock item.

8. Set the Read-Only attribute and release the file's lock item by making an F_ATTRIB system call with interface attribute F5' reset.

At this point, the file EXLOCK.NEW becomes available for access by another process.

2.12 Compatibility attributes

Compatibility attributes provide a mechanism to modify some of the Concurrent CP/M file security rules for specific command files. Concurrent CP/M includes this facility because some programs developed under earlier Digital Research operating systems do not run properly under Concurrent CP/M. Most of the problems encountered by these programs occur because they were designed for single-tasking operating systems where file security is not required. For example, a program might close a file and then continue reading and writing to the file. Under CP/M-86, this does not cause a problem. However, under Concurrent CP/M, the file system intercepts open file operations with a de-activated FCB to ensure the integrity of the file system. With compatibility attributes, you have a tool for dealing with these kinds of situations.

You should use compatibility attributes only with existing programs that run properly under CP/M or CP/M-86. Do not use compatibility attributes with new programs that you develop under Concurrent CP/M.

Compatibility attributes are defined as file attributes F1' through F4' of program (CMD) files. You can use the Concurrent CP/M SET utility to set these file attributes from the command line. However, setting a command file's compatibility attributes has no effect unless the GENCCPM COMPATMODE option has been selected during system generation. If this has been done, the P_CLI system call interrogates file attributes F1' through F4' of the command file during program loading, and modifies the Concurrent CP/M file security rules for the loaded program.

The Concurrent CP/M BDOS defines the Compatibilty Attributes as shown below.

Table 2-11. Compatibility attribute definitions

Format: Attribute
Definition

F1'
Modify the rules for Locked mode.

When a process running with F1' set opens a file in Locked mode, it can perform read and write operations to the file as normal. However, to other processes on the system, it appears as if the file was opened in Read-Only mode. Thus, another process running with F1' set can open the same file in Locked mode and also perform write operations to the file. In addition, if a process with F1' reset attempts to open the file in Locked or Read-Only mode, the open attempt is allowed but the open mode is forced to Read-Only. Furthermore, write operations are not allowed when the process has F1' reset.

This compatibility mode is designed to allow multiple copies of the same program to run concurrently, even though the program might make read and write calls to a common file that it has opened in Locked mode. In addition, this compatibility mode allows other programs not in this compatibility mode to access the file on a Read-Only basis. Note that record locking is not supported for this modified open mode. In addition, to be safe, make all static files such as program and help files Read-Only if you use this compatibility attribute.

There is an alternative to sing this attribute if a program only makes read calls to the common file. By setting the file's Read-Only attribute, you force the open mode to Read-Only when the file is opened in Locked mode.

F2'
Change F_CLOSE to partial close.

Processes running with F2' set only make partial F_CLOSE system calls. This attribute is intended for programs that close a file to update the directory but continue to use the file. A side effect of this attribute is that files opened by a process are not released from the system Lock List until the process terminates. When using this attribute, it might be necessary to set the system Lock List parameters to higher values when you generate a system with GENCCPM.

F3'
Ignore close checksum errors.

This attribute changes the way the F_CLOSE system call handles Close Checksum errors. Normally, the file system prints an error message on the console and terminates the calling process. However, if this attribute is set, the F_CLOSE system call ignores the checksum error and performs the close operation. This interface attribute is intended for programs that modify an open FCB before closing a file.

F4'
Disable FCB Checksum verification for read and write operations.

Setting this attribute also sets attributes F2' and F3'. This attribute is intended for programs that modify open FCBs during read and write operations. Use this attribute very carefully, and only with software known to work, because it effectively disables Concurrent CP/M's file security.

Use the Concurrent CP/M SET utility to specify the combination of compatibility attributes that you want set in the program's command file. For example,

```
A>SET filename [F1=ON]
A>SET filename [F1=ON,F3=ON]
A>SET filename [F4=ON]
```

If you have a program that runs under CP/M or CP/M-86 but does not run properly under Concurrent CP/M, use the following guidelines to select the proper compatibility attributes for the program:

- If the program ends with the "File Currently Opened" message when multiple copies of the program are run, set compatibility attribute F1', or place all common static files under User 0 with the system and Read-Only attributes set.

- If the program terminates with the message "Close Checksum Error", set compatibility attribte F3'.

- If the program terminates with an I/O error, try running the program with attribute F2' set. If the problem persists, then try attribute F4'. Use attribute F4' only as a last resort.

2.13 Multisector I/O

The BDOS file system provides the capability to read or write multiple 128-byte records in a single BDOS system call. This multisector facility can be visualized as a BDOS burst mode, enabling a process to complete multiple I/O operations without interference from other running processes. In addition, the BDOS file system bypasses, when possible, all intermediate record buffering during multisector I/O operations. Data is transferred directly between the calling process' memory and the drive. The BDOS also informs the XIOS when it is reading or writing multiple records on a drive. The XIOS can use this information to further optimize the I/O operation, resulting in even better performance. As a result, the use of this facility in an application program can improve its performance and also enhance overall system throughput, particularly when performing sequential I/O.

The number of records that can be transferred with multisector I/O ranges from 1 to 128. This value, called the "BDOS Multisector Count", can be set by the F_MULTISEC system call. The P_CLI system call sets the Multisector Count to 1 when it initiates a transient program for execution. Note that the greatest potential performance increases are obtained when the Multisector Count is set to 128. Of course, this requires a 16 KB buffer. The Concurrent CP/M PIP utility performs its sequential I/O with a Multisector Count of 128.

The Multisector Count determines the number of operations to be performed by the following BDOS system calls:

- F_READ and F_WRITE system calls
- F_READRAND, F_WRITERAND, and F_WRITEZF
- F_LOCK and F_UNLOCK

If the Multisector Count is N, calling one of the above system calls is equivalent to making N system calls. With the exception of disk I/O errors encountered by the XIOS, if an error interrupts a multisector read or write operations, the file system returns the number of 128-byte records successfully transferred in register AH. Section 3.14, "Concurrent file access", describes how the Multisector Count affects the F_LOCK and F_UNLOCK system calls.

2.14 Concurrent file access

Concurrent CP/M supports two open modes, Read-Only and Unlocked, which allow concurrently running processes to access common files for record operations. The Read-Only open mode allows multiple processes to read from a common file, but processes cannot write to a file open in this mode. Thus, files remain static when they are opened in Read-Only mode. The Unlocked open mode is more complex, because it allows multiple processes to read and write records to a common file. As a result, Unlocked mode has some important differences from the other open modes.

When a process opens a file in Unlocked mode, the file system returns a 2-byte field called the "File ID" in the R0 and R1 bytes of the FCB. The File ID is a required parameter of Concurrent CP/M's record locking system calls, F_LOCK and F_UNLOCK, which are only supported for files open in Unlocked mode. Note that these system calls return a successful error code if they are called for files opened in Locked mode. However, they perform no action in this case, because, by definition, the calling process has the entire file locked.

The F_LOCK and F_UNLOCK system calls allow a process to establish and release temporary ownership to particular records within a file. You must set the FCB Random Record field and place the File ID in the first two bytes of the current DMA Buffer before making these calls. The file system locks and unlocks records in units of 128 bytes, which is the standard Concurrent CP/M record size. The number of records locked or unlocked is controlled by the BDOS Multisector Count, which can range from 1 to 128 (see Section 2.13, "Multisector I/O"). In order to simplify the discussion of record locking and unlocking, the following paragraphs assume that the Multisector Count is one. However, as discussed later in this section, the more general case of multiple record locking and unlocking is a simple extension of the single record case.

The F_LOCK system call supports two types of lock operations: exclusive locks and shared locks. Interface attibute F5' specifies the type of lock. F5' = 0 requests an exclusive lock; F5' = 1 requests a shared lock. If a process locks a record with an exclusive lock, other processes cannot read, write, or lock the record. The locking process, however, can access the record with no restrictions. You should use this type of lock when exclusive control over a record is required.

If a process locks a record with a shared lock, other processes cannot write to the record or make an exclusive lock of the record. However, other processes are allowed to read the record and make their own shared locks on the record. No process, including the locking process, can write to a record with a shared lock. Shared locks are useful when you want to ensure that a record does not change, but you want to allow other processes to read the record.

The F_LOCK system call also lets you change the lock of a record if there is no conflict. For example, you can convert an exclusive lock into a shared lock with no restrictions. On the other hand, a process cannot convert a record's shared lock to an exclusive lock if another process has a shared lock on the record.

The F_LOCK system call has another option, specified by interface attribute F6', which controls whether a record must exist in order to be locked. If you make an F_LOCK system call with F6' = 0, the file system returns an error code if the specified record does not exist within the file. Setting F6' to 1 requests a logical lock operation. Logical lock operations are only limited by the maximum Concurrent CP/M file size of 32 megabytes, which corresponds to a maximum Random Record Number of 262,143 (3FFFFh). You can use logical locks to control extending a shared file.

The F_UNLOCK system call is similar to the F_LOCK call, except that it removes locks instead of creating them. There are few restrictions on unlock operations. Of course, a process can only remove locks that it has made. The F_UNLOCK system call has one option, controlled by interface attribute F5'. If F5' is set to 1, the F_UNLOCK system call removes all locks for the file made by the calling process. Otherwise, it removes the locks specified by the

Random Record field and the BDOS Multisector Count. Note that the F_CLOSE system call also removes all locks for a file on permanent close operations.

If the BDOS Multisector Count is greater than one, the F_LOCK and F_UNLOCK system calls perform multiple record locking or unlocking. In general, multiple record locking and unlocking can be viewed as a sequence of N independent operations, where N equals the Multisector Count. However, if an error occurs on any record within the sequence, no locking or unlocking is performed. For example, both F_LOCK and F_UNLOCK perform no action and return an error code if the sum of the FCB Random Record Number and the BDOS Multisector Count is greater than 262,144. As another example, the F_LOCK system call also returns an error code if another process has an exclusive lock on any record within the sequence.

When a process makes an F_LOCK system call, the file system allocates a new entry in the system Lock List to record the lock operation and associate it with the calling process. A corresponding F_UNLOCK system call removes the locked entry from the list. While the lock entry exists in the system Lock List, the file system enforces the restrictions implied by the lock item.

Because each lock item includes a record count field, a multiple lock operation normally results in the creation of a single new entry. However, if the file system must split an existing lock entry to satisfy the lock operation, an additional entry is required. Similarly, an unlock operation can require the creation of a new entry if a split is needed. Thus, in the worst case, a lock operation can require two new lock entries and an unlock operation can require one. Note that lock item splitting can be avoided by locking and unlocking records in consistent units.

These considerations are important because the Lock List is a limited resource under Concurrent CP/M. The file system performs no action and returns an error code if insufficient available entries exist in the system Lock List to satisfy the lock or unlock request. In addition, the number of lock items a single process is allowed to consume is a GENCCPM parameter established at system generation time. The file system also returns an error code if this limit is exceeded.

The file system performs several special operations for read and write system calls to a file open in Unlocked mode. These operations are required because the file system maintains the current state of an open file in the calling process' FCB. When multiple processes have the same file open, FCBs for the same file exist in each process' memory. To ensure that all processes have current information, the file system updates the directory immediately when an FCB for an unlocked file is changed. In addition, the file system verifies error situations such as end-of-file, or reading unwritten data with the directory before returning an error. As a result, read and write operations are less efficient for files open in Unlocked mode when cmpared to equivalent operations for files opened in Locked mode.

2.15 File byte counts

Although the logical record size of Concurrent CP/M is restricted to 128 bytes, the file system does provide a mechanism to store and retrieve a byte count for a file. This facility can identify the last byte of the last record of a file. The F_SIZE system call returns the Random Record Number, + 1, of the last record of a file.

The F_ATTRIB system call can set a file's byte count. This is an option controlled by interface attribute F6'. Conversely, the F_OPEN system call can return a file's byte count to the CR field of the FCB. The F_SFIRST and F_SNEXT system calls also return a file's byte count. These system calls return the byte count in the CS field of the FCB returned in the current DMA Buffer.

Note that the file system does not acces or update the byte count value in BDOS read or write system calls. However, the F_MAKE system call does set the byte count value to zero when it creates a file in the directory.

2.16 Record blocking and deblocking

Under Concurrent CP/M, the logical record size for disk I/O is 128 bytes. This is the basic unit of data transfer between the operating system and running processes. However, on disk, the record size is not restricted to 128 bytes. These records, called "physical records", can range from 128 bytes to 4 KB in size. Record blocking and deblocking is required on systems that support drives with physical record sizes larger than 128 bytes.

The process of building up physical records drom 128-byte logical records is called "record blocking". This process is required in write operations. The reverse process of breaking up physical records into their component 128-byte logical records is called "record deblocking". This process is required in read operations. Under Concurrent CP/M, record blocking and deblocking is normally performed by the BDOS.

Record deblocking implies a read-ahead operation. For example, if a process reads a logical record that resides at the beginning of a physical record, the entire physical record is read into an internal buffer. Subsequent BDOS read calls for the remaining logical records access the buffer instead of the disk. Conversely, record blocking results in the postponement of physical write operations but only for data write operations. For example, if a transient program makes a BDOS write call, the logical record is placed in a buffer equal in size to the physical record size. The write operation on the physical record buffer is postponed until the buffer is needed in another I/O operation. Note that, under Concurrent CP/M, directory write operations are never postponed.

Postponing physical record write operations has implications for some application programs. For programs that involve file updating, it is often critical to guarantee that the state of the file on disk parallels the state of the file in memory after an update operation. This is only an issue on drives where physical write operations are postponed because of record blocking and deblocking. If the system should crash while a physical buffer is pending, data would be lost. To prevent this loss of data, the F_FLUSH system call can be called to force the write of any pending physical buffers associated with the calling process.

Note: The file system discards all pending physical data buffers when a process terminates. However, the file system automatically makes an F_FLUSH call in the F_CLOSE system call. Thus, it is sufficient to make an F_CLOSE system call to ensure that all pending physical buffers for that file are written to the disk.

2.17 Reset, access, and free drive

The BDOS system calls DRV_ALLRESET, DRV_RESET, DRV_ACCESS, and DRV_FREE allow a process to control when to re-initialize a drive directory for file operations. This process of initializing a drive's directory is called "logging-in the drive".

When you start Concurrent CP/M, all drives are initialized to the reset state. Subsequently, as processes reference drives, the file system automatically logs them in. Once logged-in, a drive remains in the logged-in state until it is reset by the DRV_ALLRESET or DRV_RESET system calls, or a media change is detected on the drive. If the drive is reset, the file system automatically logs in the drive again the next time a process references it. The file system logs in a drive immediately when it detects a media change on the drive.

Note that the DRV_ALLRESET and DRV_RESET system calls have similar effects, except that the DRV_ALLRESET system call affects all drives on the system. You can specify the combination of drives to reset with the DRV_RESET system call.

Logging-in a drive consists of several steps. The most important step is the initialization of the drive's allocation vector. The allocation vector records the allocation and de-allocation of data blocks to files, as files are created, extended, deleted, and truncated. Another function performed during drive log-in is the initialization of the directory checksum vector. The file system uses the checksum vector to detect media changes on a drive. Note that permanent drives, which do not support media changes, usually do not have checksum vectors.

Under Concurrent CP/M, the DRV_RESET operation is conditional. The file system cannot reset a drive for a process if another process has an open file on the drive. However, the exact action taken by a DRV_RESET operation depends on whether the drive to be reset is permanent or removable.

Concurrent CP/M determines whether a drive is permanent or removable by interrogating a bit in the drive's Disk Parameter Block (DPB) in the XIOS. A high-order bit of "1" in the DPB Checksum Vector Size field designates the drive as permanent. A drive's Removable or Non-removable designation is critical to the reset operation described below.

The BDOS first determines whether there are any files currently open on the drive to be reset. If there are none, the reset takes place. If there are open files, the action taken by the reset operation depends on whether the drive is removable and whether the drive is Read-Only or Read-Write. Note that only the DRV_SETRO system call can set a drive to Read-Only. Following log-in, a drive is always Read-Write.

If the drive is a permanent drive and if the drive is not Read-Only, the reset operation is not performed, but a successful result is returned to the calling process.

However, if the drive is removable or set to Read-Only, the file system determines whether other processes have open files on the drive. If they do, then it denies DRV_RESET operation and returns an error code to the calling process.

If all the open files on a removable drive belong to the calling process, the process is said to own the drive. In this case, the file system performs a qualified reset on the drive and returns a successful result. This means that the next time a process accesses this drive, the BDOS performs the log-in operation only if it detects a media change on the drive. The logic flow of the drive reset operation is shown in Figure 2-6.

```
      +------------+ Yes
      | Open files |------+
      | on drive ? |      |
      +-----+------+      |
            | No          V
            |      +-------------+ Yes
            |      |   Drive     |------+
            |      | removable ? |      |
            |      +-----+-------+      |
            |            | No           |
            |            V              |
            |      +-----+-------+ Yes  |
            |      | Drive R/O ? |------+
            |      +-----+-------+      |
            |            | No           |
            V            V              V
       +-------+    +--------+   +------------+ Yes
       | Reset |    | Do not |   | Open files |------+
       | drive |    | reset  |   | belong to  |      |
       +---+---+    | drive  |   |  another   |      |
           |        +---+----+   | process ?  |      |
           |            |        +-----+------+      |
           |            |              | No          |
           |            |              V             |
           |            |        +-----------+       |
           |            |        | Qualified |       |
           |            |        |   reset   |       |
           |            |        | performed |       |
           |            |        +-----+-----+       |
           V            |              |             V
      +---------+       |              |        +--------+
      |  Disk   |       |              |        |  Disk  |
      |  reset  |<------+--------------+        | reset  |
      | success |                               | denied |
      +---------+                               +--------+
```

Figure 2-6. Disk system reset

If the BDOS detects a media change on a drive after a qualified reset, it purges all open files on the drive from the system Lock List and subsequently verifies all open FCBs in file operations for the owning process (refer to Section 2.10, "File security", for details of FCB verification).

In all other cases where the BDOS detects a media change on a drive, the file system purges all open files on the drive from the system Lock List, and flags all processes owning a purged file for automatic open FCB verification.

Note: If a process references a purged file with a BDOS command that requires an open FCB, the file system returns to the process with an FCB Checksum error.

The primary purpose of the drive reset functions is to prepare for a media change on a drive. Because a drive reset operation is conditional, it allows a process to test whether it is safe to change disks. Thus, a process should make a successful drive reset call before prompting the user to change disks. In addition, you should close all your open files on the drive, particularly files that you have written to, before prompting the user to change disks. Otherwise, you might lose data.

The DRV_ACCESS and DRV_FREE system calls perform special actions under Concurrent CP/M. The DRV_ACCESS system call inserts a dummy open file item into the system Lock List for each specified drive. While that item exists in the system Lock List, no other process can reset the drive. The DRV_FREE system call purges the Lock List of all items, including open file items, belonging to the calling process on the specified drives. Any subsequent reference to those files by a BDOS system call requiring an open FCB results in an FCB Checksum error return.

The DRV_FREE system call has two important side effects. First of all, any pending blocking/deblocking buffers on a specified drive that belong to the calling process are discarded. Secondly, any data blocks that have been allocated to files that have not been closed are lost. Be sure to close your files before making this system call.

The DRV_SETRO system call is also conditional under Concurrent CP/M. The file system does not allow a process to set a drive to Read-Only if another process has an open file on the drive. This applies to both removable and permanent drives.

A process can prevent other processes from resetting a Read-Only drive by opening a file on the drive or by issuing a DRV_ACCESS call for the drive, and then making a DRV_SETRO system call. Executing DRV_SETRO before the F_OPEN or DRV_ACCESS call leaves a window in which another process could set the drive back to Read-Write. While the open file or dummy item belonging to the process resides in the system Lock List, no other process can reset the drive to take it out of Read-Only status.

2.18 BDOS Error handling

The Concurrent CP/M file system has an extensive error handling capability. When an error is detected, the BDOS responds in one of three ways:

1. It can return to the calling process with return codes in the AX register identifying the error.

2. It can display an error message on the console, and terminate the process.

3. It can display an error message on the console, and return an error code to the calling process, as in method 1.

The file system handles the majority of errors it detects by method 1. Two examples of this kind of error are the "File Not Found" error for the F_OPEN system call, and the "Reading Unwritten Data" error for the F_READ call. More serious errors, such as disk I/O errors, are normally handled by method 2. Errors in this category, called "physical and extended errors", can also be reported by methods 1 and 3 under program control.

The BDOS Error mode, which has three states, determines how the file system handles physical and extended errors. In the default state, the BDOS displays the error message and terminates the calling process (method 2). In Return Error mode, the BDOS returns control to the calling process with the error identified in the AX register (method 1). In Return and Display Error mode, the BDOS returns control to the calling process with the error identified in the AX register and also displays the error message at the console (method 3).

While both return modes protect a process from termination because of a physical or extended error, the Return and Display mode also allows the calling process to take advantage of the built-in error reporting of the file system. Physical and extended errors are displayed on the console in the following format:

```
CP/M Error on d: error message
BDOS Function = bb File = filename.typ
```

where "d" is the letter of the drive selected when the error condition occurs; "error message" identifies the error; "nn" is the BDOS Function number, and "filename.typ" identifies the file specified by the BDOS function. If the BDOS function did not involve an FCB, the file information is omitted.

Tables 2-12 and 2-13 detail BDOS physical and extended error messages.

Table 2-12. BDOS physical errors

Format: Message
    Meaning

Disk I/O
The "Disk I/O" error results from an error condition returned to the BDOS from the XIOS module. The file system makes XIOS read and write calls to execute

BDOS file-access system calls. If the XIOS read or write routine detects an
error, it returns an error code to the BDOS, causing this error message.

Invalid Drive
The "Invalid Drive" error also results from an error condition returned to the
BDOS from the XIOS module. The BDOS makes an XIOS Select Disk call before
accessing a drive to perform a requested BDOS function. If the XIOS does not
support the selected disk, it returns an error code resulting in this error.

Read/Only File
The BDOS returns the "Read/Only File" error message when a process attempts to
write to a file with the R/O attribute set.

Read/Only Disk
The BDOS returns the Read/Only Disk error "message when a process makes a
write operation to a disk that is in Read-Only status. A drive can be placed
in Read-Only status explicitly with the DRV_SETRO system call.

Table 2-13. BDOS extended errors

Format: Message
       Meaning

File Opened in Read/Only Mode
The BDOS returns the "File Opened in Read/Only Mode" error message when a
process attempts to write to a file opened in Read-Only mode. A process can
open a file in Read-Only mode explicitly by setting FCB interface attribte
F6'. In addition, if a process opens a file in LOcked mode, the file system
automatically forces the open mode to Read-Only mode when:

- the process opens a file with the Read-Only attribute set.

- the current user number is not zero, and the process opens a user zero file
with the system attribute set.

The BDOS also returns this error if a process attempts to write to a file that
is password-protected in Write mode, and it did not supply the correct
password when it opened the file.

File Currently Open
The BDOS returns the "File Currently Open" error message when a process
attemps to delete, rename, or modify the attributes of a file opened by
another process. The BDOS also returns this error when a process attempts to
open a file in a mode incompatible with the mode in which the file was
previously opened by another process or by the calling process.

Close Checksum Error
The BDOS returns the "Close Checksum Error" message when the BDOS detects a
chacksum error in the FCB passed to the file system with an F_CLOSE call.

Password Error
The BDOS returns the "Password Error" message when passwords are required and
the password is not supplied or is incorrect.

File Already Exists
The BDOS returns the "FIle Already Exists" error message for the F_MAKE and F_RENAME system calls when the BDOS detects a conflict on filename and filetype.

Illegal ? in FCB
The BDOS returns the "Illegal ? in FCB" error message when the BDOS detects a "?" character in the filename or filetype of the passed FCB for the F_ATTRIB, F_OPEN, F_RENAME, F_TIMEDATE, F_WRITEXFCB, F_TRUNCATE, and F_MAKE system calls.

Open File Limit Exceeded
The BDOS returns the "Open File Limit Exceeded" error message when a process exceeds the process file lock limit specified by GENCCPM. The F_OPEN, F_MAKE, and DRV_ACCESS system calls can return this error.

No Room in System Lock Limit
The BDOS returns the "No Room in System Lock List" error message when no room for new entries exists within the system Lock List. The F_OPEN, F_MAKE, and DRV_ACCESS system calls can return this error.

The following paragraphs describe the error return code conventions of the file system calls. Most file system calls fall into three categories in regard to return codes; they return an error code, a directory code, or an error flag. The error conventions let programs written for CP/M-86 run without modification.

The following BDOS system calls return a logical error in register AL:

```
     F_LOCK
     F_READ
     F_READRAND
     F_UNLOCK
     F_WRITE
     F_WRITERAND
     F_WRITEZF
```

Table 2-14 lists error code definitions for register AL.

Table 2-14. BDOS error codes

```
     Code    Definition
     ----    ----------
      00h    Function successful
      01h    Reading unwritten date
               No available directory space (Write Sequential)
      02h    No available data block
      03h    Cannot close current extent
      04h    Seek to unwritten extent
      05h    No available directory space
      06h    Random Record Number out of range
   *  08h    Record locked by another process
```

```
            (restricted to files opened in Unlocked mode)
        09h    Invalid FCB (previous BDOS F_CLOSE system call
            returned an error code, and invalidated the FCB)
        0Ah    FCB Checksum error
      *  0Bh    Unlocked file unallocated block verify error
     **  0Ch    Process record lock limit exceeded
     **  0Dh    Invalid File ID
     **  0Eh    No room in system Lock List
       0FFh    Physical error: refer to register AH

        * = Returned only for files opened in Unlocked mode.
       ** = Returned only by the F_LOCK and F_UNLOCK system
          calls for files opened in Unlocked mode.
```

For BDOS read and write system calls, the file system also sets register AH when the returned error code is a value other than zero or 0FFh. In this case, register AH contains the number of 128-byte records successfully read or written before the error was encountered. Note that register AH can only contain a non-zero value if the calling process has set the BDOS Multisector Count to a value other than one; otherwise, register AH is always set to zero. On successful system calls (Error Code = 0), register AH is also set to zero. If the Error Code if 0FFh, register AH contains a physical error code (see Table 2-15, "BDOS physical and extended errors").

The following BDOS system calls return a directory code in register AL:

```
       DRV_SETLABEL
       F_ATTRIB
       F_CLOSE
       F_DELETE
       F_MAKE
       F_OPEN
       F_RENAME
       F_SIZE
       F_SFIRST
       F_SNEXT
       F_TIMEDATE
       F_TRUNCATE
       F_WRITEXFCB
```

The directory code definitions for register AL follow:

```
       00h-03h =   successful function
         0FFh = unsuccessful function
```

With the exception of the F_SFIRST and F_SNEXT system calls, all functions in this category return with the directory code set to zero upon a successful return. However, for these two system calls, a successful directory code identifies the relative starting position of the directory entry in the calling process' current DMA Buffer.

If a process uses the F_ERRMODE system call to place the BDOS in Return Error mode, the following system calls return an error flag in register AL on physical errors:

DRV_GETLABEL
DRV_ACCESS
DRV_SET
DRV_SPACE
DRV_FLUSH

The error flag definition for register AL follows:

00h = successful function
0FFh = physical error: refer to register AH

The BDOS returns non-zero values in register AH to identify a physical or extended error if the BDOS Error mode is in one of the return modes. Except for system calls that return a Directory Code, register AL equal to 0FFh indicates that register AH identifies the physical or extended error. For functions that return a Directory Code, if register AL equals 255, and register AH is not equal to zero, register AH identifies the physical or extended error. Table 2-15 shows the physical and extended error codes returned in register AH.

Table 2-15. BDOS physical and extended errors

Code	Explanation
01h	Disk I/O Error: permanent error
02h	Read/Only Disk
03h	Read/Only File, File Opened in Read/Only Mode, File Password Protected in Write Mode, and Correct Password Not Specified
04h	Invalid Drive: drive select error
05h	File Currently Open in an incompatible mode
06h	Close Checksum Error
07h	Password Error
08h	File Already Exists
09h	Illegal ? in FCB
0Ah	Open File Limit Exceeded
0Bh	No Room in System Lock List

The following two system calls represent a special case, because they return an address in register AX:

DRV_ALLOCVEC
DRV_DBP

When the calling process is in one of the BDOS return error modes and the BDOS detects a physical error for these system calls, it returns to the calling process with registers AX and BX set to 0FFFFh. Otherwise, they return no error code.

Under Concurrent CP/M, the following system calls also represent a special case:

DRV_ALLRESET

DRV_RESET
DRV_SETRO

These system calls return to the calling process with registers AL and BL set to 0FFh if another process has an open file or has made a DRV_ACCESS call that prevents the reset or write protect operation. If the calling process is not in Return Error mode, these system calls also display an error message identifying the process that prevented the requested operation.

EOF

CCPMPRG3.WS4   (Concurrent CP/M Programmer's Reference Guide, Chapter 3)

(Retyped by Emmanuel ROCHE.)

Section 3: Transient commands

3.1 Transient program load and exit

A transient program is a file of type CMD that is loaded from disk and resides in memory only during its operation. A resident system process is a file of type RSP that is included in Concurrent CP/M during system generation. Section 4 describes the three system memory models that determine the initial values of segment registers in transient processes.

You can initiate a transient process by entering a command at a system console. The console's TMP (Terminal Message Processor) then calls the Command Line Interpreter system call (refer to the P_CLI system call), and passes to it the command line entered by the user. If the command is not an RSP, then the P_CLI system call locates and the loads the proper CMD file. P_CLI then calls the F_PARSE system call to parse up to two filenames following the command, and place the properly formatted FCBs at locations 005Ch and 006Ch in the Base Page of the initial Data Segment.

The P_CLI system call initializes memory, the Process Descritor, and the User Data Area (UDA), and allocates a 96-byte stack area, independent of the program, to contain the process' initial stack. If 8087 processing is required (see Section 3.1.2, "8087 support"), P_CLI allocates an additional 96 bytes for the UDA. Concurrent CP/M divides the DMA address into the DMA segment address and the DMA offset. P_CLI initializes the default DMA segment to the value of the initial data segment, and the default DMA offset to 0080h.

The P_CLI system call creates the new process with a P_CREATE system call, and sets the initial stack so that the process can execute a Far Return instruction to terminate. A process can also ends when it calls DRV_ALLRESET or P_TERM.

You can also terminate a process by typing a single Ctrl-C during console input. See C_MODE for details of enabling/disabling Ctrol-C. Ctrl-C, when typed at the system prompt ("A>"), forces a DRV_RESET call for each logged-in drive. This operation only affects removale media drives.

Note: Additional UDA space is allocated for 8087 processing only if the process is initialized by the P_CLI or P_LOAD system call. Other processes (such as RSPs) that require 8087 processing and do not use P_CLI or P_LOAD must allocate this additional UDA space themselves.

3.1.1 Shared code

Concurrent CP/M allows processes to share program code. This capability of sharing program code avoids un-necessary program loading of a code segment already in memory, and conserves memory space, since multiple copies of the same program code do not have to occupy different memory space. During program load of a "sharable" program code, the system allocates the code group separately from the rest of the program. This code group is maintained in memory, even after the program has terminated. Subsequent loading of the same program does not load the code group, but uses the existing one instead. Obviously, programs written with separate code and data can take advantage of this feature.

The system maintains a shared code group in memory until a memory request or a reset drive forces its release. The system maintains shared code groups in memory in Least Recently Used (LRU) order on the Shared Code List. If a memory request is made that cannot be satisfied, the list is drained, one at a time, until the memory request is satisfied, or the Shared Code List is emptied. If a drive is reset, the system purges all code groups from the Shared Code List loaded from that drive.

A shared code program is flagged by the value 09h in the G_Type field of the Code Group Descriptor in the CMD file Header Record (see Section 3.2, "Command file format"). The user may set this field by using the CHSET utility (see "Concurrent CP/M User's Guide"). Note that programs using the 8080 Memory model cannot be set to shared code.

3.1.2 8087 support

Concurrent CP/M provides optional 8087 support for systems that use the 8087 processor. This support is indicated by the Program Flag, byte 127 (7Fh), of the CMD file Header Record. Setting bit 6 (bit 0 is least significant bit) of the Program Flag indicates optional 8087 support, which means that, if the 8087 is present, the program uses it; otherwise, the program will emulate it. If bit 5 of the Program Flag is set, it indicates that the 8087 must be present in order for the program to run. If no 8087 is present and bit 5 of the Program Flag is set, the system returns an error when it tries to load the program. The CHSET utility can be used to set the program's header record for optional or required 8087 support.

If you use the P_CLI or P_LOAD system call to initiate and execute a process, the system allocates and extra 96 bytes to the UDA for 8087 support. If you require 8087 support and do not use the P_CLI or P_LOAD system call, you must specifically allocate this additional 96 bytes to the UDA, turn on the 8087 flag in the PD, and initialize the CW and SW fields in the 8087 UDA extension (see description of these fields in Section 6, "System calls", under the P_CREATE system call).

3.1.3 8087 exception handling

Although the system provides its own 8087 exception handling routine, the user might want to write his own 8087 exception handler. Appendix E includes instructions and information required by the user to write his own 8087 exception handler, with a sample listing of an 8087 exception handler routine.

3.2 Command file format

A CMD file consists of a 128-byte Header Record, followed immediately by the memory image. The command file Header Record is composed of 8 group descriptors (GDs), each 9 bytes long. Each group descriptor describes a portion of the program to be loaded. The format of the Header Record is shown in Figure 3-1.

```
+------+------+------+------+------+-------+-----+------+------+
| GD 1 | GD 2 | GD 3 | GD 4 | GD 5 | GD 6 | GD 7 | GD 8 |      |
+------+------+------+------+------+------+------+------+------+
<------------------------- 128 bytes -------------------------->
```

Figure 3-1. CMD file Header Record format

In Figure 3-1, "GD 1" through "GD 8" represent group descriptors. Each group descriptor corresponds to an independently loaded program unit, and has the format shown in Figure 3-2.

```
0h       1h         3h        5h       7h       9h
+--------+----------+--------+-------+-------+
| G_Type | G_Length | A_Base | G_Min | G_Max |
+--------+----------+--------+-------+-------+
```

Figure 3-2. Group Descriptor format

G-Type determines the group descriptor type. The valid group descriptors have a G_Type in the range 1 through 8, as shown in Table 3-1. All other values are reserved for system use. For a given CMD file Header Record, only a Code Group and one of any other type can be included.

If a program uses either the Small or Compact Model, the code group is typically pure; that is, it is not modified during program execution.

Table 3-1. Group Descriptors

G-Type	Group Type
1	Code Group (non-shared)
2	Data Group
3	Extra Group
4	Stack Group
5	Auxiliary Group # 1
6	Auxiliary Group # 2
7	Auxiliary Group # 3

8	Auxiliary Group # 4
9	Code Group (shared)

All remaining values in the group descriptor are given in increments of 16-byte paragraph units, with an assumed low-order 0 nibble to complete the 20-bit address.

Table 3-2. Group Descriptor fields

Format: Field
Description

G-Length
Gives the number of paragraphs in the group. Given a G-Length of 0080h, for example, the size of the group is 0800h = 2048 bytes.

A-Base
Defines the base paragraph address for a non-relocatable group

G-Min/G-Max
define the minimum and maximum size of the memory area to allocate to the group.

The memory model described by a Header Record is implicitly determined by the Group Descriptors (refer to Section 4.1, "Transient execution models"). The 8080 Memory Model is assumed when only a code group is present, because no independent data group is named. The Small Memory Model is assumed when both a code and data group are present but no additional Group Descriptors occur. Otherwise, the Compact Memory Model is assumed when the CMD file is loaded.

3.3 Base Page initialization

The Concurrent CP/M Base Page contains default values and locations initialized by the P_CLI and P_LOAD system calls, and used by the transient process.

The Base Page occupies the regions from offset 0000h trough 00FFh relative to the initial data segment, and contains the values shown in Figure 3-3.

```
bytes:  Lo   Mi   Hi    Lo    Hi
     0    1    2    3     4     5         6
     +----+----+----+------+-----+----------+
0000h | Code  Length | Code  Base |   M80    |
     +----+----+----+------+-----+----------+
0006h | Data  Length | Data  Base | Reserved |
     +----+----+----+------+-----+----------+
000Ch | Extra Length | Extra Base | Reserved |
     +----+----+----+------+-----+----------+
0012h | Stack Length | Stack Base | Reserved |
     +----+----+----+------+-----+----------+
0018h | Aux 1 Length | Aux 1 Base | Reserved |
      +----+----+----+------+-----+----------+
001Eh | Aux 2 Length | Aux 2 Base | Reserved |
      +----+----+----+------+-----+----------+
0024h | Aux 3 Length | Aux 3 Base | Reserved |
      +----+----+----+------+-----+----------+
002Ah | Aux 4 Length | Aux 4 Base | Reserved |
      +----+----+----+------+-----+----------+
0030h | Bytes 0030h through 004Fh are not    |
      | currently used, and are reserved     |
      | for future use by Digital Research.  |
      +----+----+----+------+-----+----------+
0050h |Driv| PW1 Addr| P1Len| PW2 Addr       |
      +----+----+----+------+-----+----------+
0056h |P2Ln| Reserved for future use         |
      +----+----+----+------+-----+----------+
005Ch |         Default File Name 1          |
      +----+----+----+------+-----+----------+
006Ch |         Default File Name 2          |
      +----+----+----+------+-----+----------+
007Ch | CR | RND Record Numb|                |
      +----+----+----+------+-----+----------+
0080h |     Default 128-byte DMA Buffer      |
      +----+----+----+------+-----+----------+
```

Figure 3-3. Concurrent CP/M Base Page values

The fields in the Base page are defined as follows:

- The M80 byte is a flag indicating whether the 8080 Memory Model was used during load. The values of the flag ae defined as:

  1 = 8080 Model
  0 = not 8080 Model

If the 8080 Model is used, the code length never exceeds 0FFFFh.

- The bytes marked "Aux 1" through "Aux 4" corresponds to a set of four optional independent groups that might be required for programs that execute using the Compact Memory Model. The initial values of these descriptors are derived from the Header Record in the memory image file.

- Length is stored using the Intel convention: Low, Middle, and High bytes.

- Base refer to the paragraph address of the beginning of the segment.

- The drive byte identifies the drive from which the transient program was read. 0 designates the default drive, while a value of 1 through 16 identifies drives A through P.

- Password 1 Addr (bytes 0051h-0052h) contains the address of the password field of the first command tail operand in the default DMA Buffer at 0080h. The P_CLI system call sets this field to 0 if no password is specified.

- P1 Len (byte 0053h) contains the length of the password field for the first command tail operand. The P_CLI system call sets this to 0 if no password is specified.

- Password 2 Addr (bytes 0054h-0055h) contains the address of the password field of the second command tail operand in the default DMA Buffer at 0080h. The P_CLI system call sets this field to 0 if no password is specified.

- P2 Len (byte 0056h) contains the length of the password field for the second command tail operand. The P_CLI system call sets this field to 0 if no password is specified.

- File Name 1 (bytes 005Ch-0067h) is initialized by the P_CLI system call for a transient program from the first command tail operand of the command line.

- File Name 2 (bytes 006Ch-0077h) is initialized by the P_CLI system call for a transient program from the second command tail operand of the command tail.

Note: File Name 1 can be used as part of a File Control Block (FCB) beginning at 005Ch. To preserve File Name 2, copy it to another location before using the FCB in file I/O system calls.

- The CR field (byte 007Ch) contains the current record position used in sequential file operations with the FCB at 005Ch.

- The optional Random Record Number (bytes 007Dh-007Fh) is an extension of the FCB at 005Ch, used in random record processing.

- The Default DMA Buffer (bytes 0080h-00FFh) contains the command tail when the P_CLI system call loads a transient program.

3.4 Parent/Child relationships

Under Concurrent CP/M, when one process creates another process, there is a parent/child relationship between them. The child process inherits most of the default values of the parent process. This includes the default disk, user number, console, list device, and password. The child process also inherits interrupt vectors 0, 1, 3, 4, 224, and 225, which the parent process initialized.

3.5 Direct video mapping

Processes which bypass Concurrent CP/M Character I/O system calls and use a video map or screen buffer directly cannot be monitored by the system, and continue to display characters on the screen evence when running in the background. Consequently, any screen displayed by the program in the foreground console is interspersed with characters displayed by the program in the background using direct video map I/O. To avoid the screen problems created by using direct video I/O, set bit 3 of the Program Flag to indicate to the system that the process is to be put in suspend mode whenever it is running in the background and may continue running only when it is switched to the foreground. The CHSET utility (see the "Concurrent CP/M User's Guide") can be used to set bit 3 of the Program Flag.

Note that bypassing the system Character I/O system calls negates the concurrency of a process, since the system suspends it from running (if bit 3 of Program Flag is set) unless it is running in the foreground.

EOF

CCPMPRG4.WS4 (Concurrent CP/M Programmer's Reference Guide, Chapter 4)

(Retyped by Emmanuel ROCHE.)

Section 4: Command file generation

4.1 Transient execution models

When the program is loaded, the initial values of the segment registers, the instruction pointer, and the stack pointer are determined by the specific type of memory model used by the transient process, indicated in the CMD file Header Record.

There are three memory models; the 8080 Memory model, the Small Memory model, and the Compact Memory model.

Table 4-1. Concurrent CP/M Memory Models

Model	Group Relationships
8080 Model	Code and Data Groups Overlap
Small Model	Independent Code and Data Groups
Compact Model	Three or More Independent Groups

The 8080 Model supports programs which are directly translated from CP/M-80 where code and data areas are intermixed. The 8080 Model consists of one group that contains all the code, data, and stack areas. Segment registers are initialized to the starting address of the region containing this group. The segment registers can, however, be managed by the application program during execution, so that multiple segments in the code group can be addressed.

The Small Model is similar to that defined by Intel, where the program consists of an independent code group and a data group. The Small Model is suitable for use by programs where code and data is easily separated. Note again that the code and data groups often consist of, but are not restricted to, 64K byte segments.

The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in programs. Each group can consist of one or more segments, but if any group exceeds 64 KB in size, or if auxiliary groups are present, then the application program must manage its own segment registers during execution, in order to address all code and data areas.

The three models differ primarily in how the operating system initializes the segment registers when it loads a transient process. The P_LOAD system call determines the memory model used by a transient program by examining the program group usage, as described in the following sections.

For all models, the system initializes an internal 96-byte stack area. The first two words of this stack are reserved for the double word return, for termination by a Far Return instruction. The initial program stack for all models is shown in Figure 4-1 below.

```
                     +---------------+
                     |  Ret Segment  |
     Far Return Address +---------------+
     SS:SP --------------> |  Ret Offset   |
                     +---------------+
                     |               |
                     |   92 Bytes    |
                     |               |
                     |               |
                     +---------------+
```

Figure 4-1. Initial program stack

The transient program can terminate by using the P_TERMCPM or P_TERM system call, or by executing a Far Return instruction when the SS and SP still point to the initial program stack.

4.1.1 The 8080 Memory Model

The 8080 Model is assumed when the transient program contains only a code group. In this case, the Command Line Interpreter (P_CLI) system call initializes the CS, DS, and ES registers to the beginning of the code group, and sets the SS and SP registers to a 96-byte initial stack area that it allocates.

Note: The P_CLI system call initializes the stack so that, if the process executes a Far Return instruction, it terminates. This system call sets the Instruction Pointer (IP) register to 0100h, thus allowing base page values at the beginning of the code group. Following program load, the 8080 Model appears as shown in figure 4-2.

```
                          +---------------+
                          |   Code/Data   |
                          |               |
                          |      ...      |
                          |               |
                          |   Code/Data   |
           CS:IP --> 0100h +---------------+
                          |   Base Page   |
   CS:0, DS:0, ES:0 --> 0000h +---------------+
```

Figure 4-2. Concurrent CP/M 8080 Memory Model

The intermixed code and data areas are indistinguishable. The Base Page values are described in Section 3.3, "Base Page initialization". The following ASM-86 example shows how to code an 8080 Model transient program.

```
          CSEG
          ORG     0100h
          ...
          ...     (code)
          ...
     endcs   EQU     $
     ;------------------------
          DSEG
          ORG     OFFSET endcs
          ...
          ...     (data)
          ...
     ;------------------------
          END
```

4.1.2 The Small Memory Model

The Small Model is assumed when the transient program contains both a code and data group. (In ASM-86, all code is generated following a CSEG directive. Data is defined following a DSEG directive, with the origin of the data segment independent of the code segment.) In this model, the P_CLI system call sets the CS to the beginning of the code group, the IP to 0000h, the DS and ES registers to the beginning of the data group, and the SS and SP registers to a 96-byte initial stack area that it initializes. Following program load, the Small Model appears as shown in figure 4-3.

```
                                               +-------+
                                               |       |
                                               | ...   |
                                               |       |
                         +-------+             | DATA  |
                         |       |             |       |
                         | ...   |       0100h +-------+
                         |       |             | Base  |
                         | CODE  |             | Page  |
                         |       |  DS:0, ES:0 --> 0000h +-------+
     CS:0, IP:0 --> 0000h +-------+
```

     Figure 4-3. Concurrent CP/M Small Memory Model

The machine code begins at CS+0000h, the Base Page values begin at DS+0000h, and the data area starts at DS+0100h. The following ASM-86 example shows how to code a Small Model transient program.

```
          CSEG
          ...
          ...     (code)
          ...
     ;------------------------
          DSEG
          ORG     0100h
          ...
```

```
        ...    (data)
        ...
;------------------------
        END
```

4.1.3 The Compact Memory Model

The Compact Model is assumed when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the P_CLI system call sets the CS, DS, and ES registers to the base addresses of their respective areas, with the IP set to 0000h, and the SS and SP registers set to a 96-byte stack area allocated by this system call.

Figure 4-4 shows the initial configuration of segments in the Compact Model. The values of the various segment registers can be changed during execution, by loading from the initial values placed in Base Page. This allowing access to the entire memory space.

```
                          +-------+
       +---------+        |       |
       |         |        | DATA  |        +-------+
       |  ...    |        |       |        |       |
       |         |        +-------+        | ...   |
 CS,IP |  CODE   |        | BASE  |        |       |
       |         |        | PAGE  |        | DATA  |
 0000h +---------+ DS:0000h +-------+ ES:0000h +-------+
```

Figure 4-4. Concurrent CP/M Compact Memory Model

If the transient program intends to use the stack group as a stack area, the SS and SP registers must be set upon entry. The SS and SP registers remain in the initial stack area, even if a stack group is defined.

Although it appears that the SS and SP registers should be set to address the stack group, there are two contradictions. First, the transient program might be using the stack group as a data area. In that case, the stack values set by the P_CLI system call to allow a Far Return to terminate a transient program could overwrite data in the stack area. Second, the SS register would logically be set to the base of the group, while the SP would be set to the offset of the end of the group. However, if the stack group exceeds 64 KB, the address range from the base to the end of the group exceeds a 16-bit offset value.

The following ASM-86 example shows how to code a Compact Model transient program.

```
        CSEG
        ...
        ...    (code)
        ...
;------------------------
        DSEG
```

```
          ORG     0100h
          ...
          ...    (data)
          ...
      ;------------------------
          ESEG
          ...
          ...    (more data)
          ...
      ;------------------------
          SSEG
          ...
          ...    (stack area)
          ...
      ;------------------------
          END
```

4.2 GENCMD

The GENCMD utility creates a CMD file from an input H86 file. GENCMD does not alter the original H86 file. The GENCMD invocation has the following form:

```
      GENCMD filename {parameter-list}
```

where the filename corresponds to the H86 input file with an assumed (and unspecified) file type of H86. GENCMD accepts optional parameters to specifically identify the 8080 Memory Model and to describe memory requirements of each segment group. The GENCMD parameters are listed following the filename, as shown in the command line above, where the parameter-list consists of a sequence of keywords (shown below) and values separated by commas or blanks.

```
      8080  CODE  DATA  EXTRA  STACK  X1  X2  X3  X4
```

The 8080 keyword forces a single code group, so that the P_LOAD system call sets up the 8080 Memory Model for execution, thus allowing intermixed code and data in a single segment. The form of this command is

```
      GENCMD filename 8080
```

The remaining keywords follow the filename or the 8080 option, and define specific memory requirements for each segment group, corresponding one-to-one with the segment groups defined in the previous section. In each case, the values corresponding to each group are enclosed in square brackets and separated by commas. Each value is a hexadecimal number representing a paragraph address or segment length in paragraph units denoted by hhhh, prefixed by a single letter which defines the meaning of each value:

Ahhhh	Load the group at absolute location hhhh
Bhhhh	The group starts at hhhh in the hex file
Mhhhh	The group requires a minimum of hhhh * 16 bytes
Xhhhh	The group can address a maximum of hhhh * 16 bytes

Generally, the CMD file Header Record values are derived directly from the H86 file, and the parameters shown above need not be included. The following situations, however, require the use of GENCMD parameters:

8080 Keyword
The 8080 keyword is included whenever ASM-86 is used in the conversion of CP/M-80 programs to the 8086/8088 environment when code and data are intermixed within a single 64 KB segment, regardless of the use of CSEG and DSEG directives in the source program.

Absolute Address
An absolute address (a hexadecimal value) must be given for any group that must be located at an absolute location. This value is not ususally specified, as Concurrent CP/M cannot ensure that the required memory region is available. In that case, the CMD file cannot be loaded.

Beginning Address of Groups
The B value is used when GENCMD processes a hex file produced by Intel's OH86, or a similar utility program that contains more than one group. The output from OH86 consists of a sequence of data records with no information to identify code, data, extra, stack, or auxiliary groups. In this case, the B value marks the beginning address of the group named by the keyword, causing GENCMD to load data following this address to the named group (refer to the examples below). Thus, the B value is usually used to mark the boundary between code and data segments when no segment information is included in the hex file. Files produced by ASM-86 do not require the use of the B value, because segment information is included in the H86 file.

Minimum Memory Value
The M value (minimum memory value) is included only when the hex records do not define the minimum memory requirements for the named group. Generally, the code group size is determined precisely by the data records loaded into the area. The total space required for the group is defined by the range between the lowest and highest data byte addresses. The data group, however, might contain uninitialized storage at the end of the group. Thus, no data records are present in the hex file that define the highest referenced data item. The highest address in the data group can be defined within the source program by including the ASM-86 directive DB 0 as the last data item in the assembly language source file. Alternatively, the M value can be included to allocate the additional space at the end of the group. Similarly, the stack, extra, and auxiliary group sizes must be defined using the M value, unless the highest addresses within the groups are implicitly defined by data records in the hex file.

Maximum Memory Size
The maximum memory size, given by the X value, is generally used when additional free memory might be needed for such purposes as I/O buffers or symbol tables. If the data area size is fixed, then the X parameter need not be included. In this case, the X value is assumed to be the same as the M value. The value XFFFF allocates the largest memory region available but, if used, the transient program must be aware that a three-byte length field is produced in the Base Page for this group, where the high-order byte may be non-zero. Programs converted directly from CP/M-80 or programs that use a 2-

byte pointer to address buffers should restrict this value to XFFF or less, producing a maximum allocation length of 0FFF0h bytes.

The following GENCMD command line transforms the file X.H86 into the file X.CMD with the proper header record:

```
        A>gencmd x code[a40] data[m30,xfff]
```

In this case, the code group is forced to paragraph address 0040h, or its equivalent, byte address 0400h. The data group requires a minimum of 0300h bytes, but can use up to 0FFF0h bytes, if available.

Assuming a file Y.H86 exists on drive B containing Intel hex records with no interspersed segment information, the command

```
        A>gencmd b:y data[b30,m20] extra[b50] stack[m40] x1[m40]
```

produces the file Y.CMD on drive B by selecting records beginning at address 0000h, and less than 0300h, for the Code Segment, with records starting at 0300h, and less than 0500h, allocated to the Data Segment. The Extra Segment is filled from records beginning at 0500h and higher, while the Stack and Auxiliary Segment #1 are uninitialized areas requiring a minimum of 0400h bytes each. In this example, the data area requires a minimum of 0200h bytes. Note again that the B value need not be included if the Digital Research ASM-86 assembler is used.

4.3 Intel hexadecimal file format

GENCMD input must be in Intel hexadecimal format produced by both the Digital Research ASM-86 assembler and the standard Intel OH86 utility program. (Refer to Intel document #9800639-03 entitled "MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users".) The CMD file produced by GENCMD contains a Header Record defining the memory model and memory size requirements for loading and executing the CMD file.

An Intel hexadecimal file consists of the traditional sequence of ASCII records, where the beginning of the record is marked by an ASCII colon (":"), and each subsequent digit position contains an ASCII hexadecimal digit in the range 0-9 or A-F.

There are four kinds of hexadecimal record formats. The Start Address Record specifies the starting address of the execution file. The Extended Address Record specifies the bits 4-19 of the Segment Base Address, where bits 0-3 of the SBA are zero. The Data Record contains a string of hexadecimal ASCII code that represents a portion of the 8086 memory image. The End-Of-File Record specifies the end of the object file.

Figure 4-5 shows the four record formats, their fields, and the contents of these fields. The fields are defined in Table 4-2.

Starting Address Record

```
+---+----+------+----+------+---+
| : | 04 | 0000 | 03 | HHHH | B |
+-+-+-+--+--+---+-+--+--+---+-+-+
  |   |     |     |     |     |
  |   |     |     |     |     +--> Checksum
  |   |     |     |     +--------> C-Seg
  |   |     |     +--------------> Rec Type
  |   |     +--------------------> Zeroes
  |   +--------------------------> Rec Len
  +------------------------------> Rec Mark
```

Extended Address Record

```
+---+----+------+----+------+---+
| : | 02 | 0000 | 02 | HHHH | B |
+-+-+-+--+--+---+-+--+--+---+-+-+
  |   |     |     |     |     |
  |   |     |     |     |     +--> Checksum
  |   |     |     |     +--------> USBA
  |   |     |     +--------------> Rec Type
  |   |     +--------------------> Zeroes
  |   +--------------------------> Rec Len
  +------------------------------> Rec Mark
```

Data Record

```
+---+----+------+----+------+---+
| : | HH | HHHH | 00 | DATA | B |
+-+-+-+--+--+---+-+--+--+---+-+-+
  |   |     |     |     |     |
  |   |     |     |     |     +--> Checksum
  |   |     |     |     +--------> Data bytes
  |   |     |     +--------------> Rec Type
  |   |     +--------------------> Ld Addr
  |   +--------------------------> Rec Len
  +------------------------------> Rec Mark
```

End-Of-File Record

```
+---+----+------+----+---+
| : | 00 | 0000 | 01 | B |
+-+-+-+--+--+---+-+--+-+-+
  |   |     |     |    |
  |   |     |     |    +---------> Checksum
  |   |     |     +--------------> Rec Type
  |   |     +--------------------> Zeroes
  |   +--------------------------> Rec Len
  +------------------------------> Rec Mark
```

Figure 4-5. Intel hexadecimal file format

Table 4-2. Intel hexadecimal field definitions

Format: Field
Contents

Rec Mark
Specifies start of record.

Rec Len
Record Length 00-FF (0-255 in decimal).

Zeroes
Extended Address Record: 0000h
Starting Address Record: 0000h
End-Of-File Record: 0000h

Ld Addr
Data Record: SBA offset defining address of byte 0 of data.

Rec Type
00 = Data Record
01 = End-Of-File Record
02 = Extended Address Record
03 = Starting Address Record

The following are output from ASM-86 only:

81 = same as 00, data belongs to Code Segment
82 = same as 00, data belongs to Data Segment
83 = same as 00, data belongs to Stack Segment
84 = same as 00, data belongs to Extra Segment

85 = paragraph address for absolute Code Segment
86 = paragraph address for absolute Data Segment
87 = paragraph address for absolute Stack Segment
88 = paragraph address for absolute Extra Segment

(85, 86, 87, and 88 are Digital Research extensions.)

C-Seg
Four hexadecimal digits specifying the Code Segment address. The high-order and low-order digits are the 10th and 13th characters of the record, repesctively.

USBA
Four hexadecimal digits specifying the Upper Segment Base Address. The high-order and low-order digits are the 10th and 13th characters of the record, respectively.

Data
Pairs of hexadecimal digits representing the ASCII code for each data byte. The high-order digit is the first digit of each pair.

Checksum
Extended Address Record: Checksum of Rec Len, Zeroes, Rec Type, and USBA fields.
Starting Address Record: Checksum of Rec Len, Zeroes, Rect Type, C-Seg, and IP fields.
Data Record: Checksum of Rec Len, Ld Addr, Rec Type, and data fields.

Ed-Of-File Record: Contains ASCII code 4646h, checksum of Rec Len, Zeroes, and Rec Type fields.

All characters preceding the colon (":") for each record are ignored. See "MCS-86 Absolute Object File Formats", published by Intel, for additional information on hexadecimal file record format.

EOF

CCPMPRG5.WS4 (Concurrent CP/M Programmer's Reference Guide, Chapter 5)

(Retyped by Emmanuel ROCHE.)

Section 5: Resident System Process generation

5.1 Introduction to RSPs

Resident System Process are programs that become part of the Concurrent CP/M operating system. They can be useful in several ways: to create a turnkey system, autoloading programs when Concurrent CP/M is booted; to build customized user interfaces or shells at the consoles, for monitoring hardware not supported in the XIOS; and to avoid disk loading time for frequently-used commands.

The source code for the ECHO RSP is included in Appendix D. Study this listing carefully while reading this section. The discussion of the P_CREATE system call in Section 6, "System calls", is also helpful in understanding RSPs.

Resident System Processes are included in Concurrent CP/M during system generation. GENCCPM searches the directory for all files with the filetype RSP, and prompts the user to choose whether it is to be included in the generated system file, CCPM.SYS. An RSP file is created by generating a CMD file and renaming it with an RSP filetype. The GENCCPM program is documented in the "Concurrent CP/M System Guide".

5.2 RSP memory models

Under Concurrent CP/M, there are two basic memory models for RSPs. They are similar to the 8080 Model and the Small Model of transient programs. However, several important distinctions exist between the transient program and RSP memory models. The RSP has no equivalent to the Base Page of the transient program's Data Segment. The RSP is responsible for its own Process Descriptor (PD) and User Data Area (UDA). The RSP must also allocate an additional 96 bytes at the end of the User Data Area if 8087 processing is required. The system creates and initializes these data structures for the transient programs automatically at load time. RSPs, on the other hand, must initialize these structures within their own Data Segments (see P_CLI and P_CREATE system calls for PD and UDA descriptions).

Note that Concurrent CP/M does not support Compact Model RSPs. Extra and Stack Segments must be part of the Data Segment.

Although there is no Base Page in an RSP, there is an RSP Header that must exist at offset 0000h of the Data Segment. In the 8080 Model, this implies that the RSP Header is in the Code Segment. The RSP Header and the associated data structures are discussed in Section 5.4, "Creating an initializing an

RSP".

5.2.1 8080 Model RSP

The 8080 Model consists of mixed code and data. When the system gives control of the CPU to an 8080 Model RSP, it initializes the Code, Data, Extra, and Stack Segment registers to the same value. Use GENCCPM with the 8080 option to generate an 8080 Model RSP. GENCCPM assumes the 8080 Model if the CMD file Header Record of the RSP has a single Code Group Descriptor, and no other Group Descriptors (refer to Section 3.2, "Command file format"). When discussing an 8080 Model RSP, any reference to the Data Segment also refers to the Code Segment.

5.2.2 Small Model RSP

The Small Model RSP implies separate Code and Data Segments. Before the system gives control of the CPU to a Small Model RSP, it initializes the Data, Extra, and Stack Segment Registers to the Data Segment address, while the Code Segment register is initialized to the Code Segment address. There is no guarantee where GENCCPM will place the Code Segment in memory relative to the Data Segment. The CMD file Header Record for this kind of RSP must have both Data and Code Group Descriptors.

```
                                   Small Model
                                   +------------+ <-- High
                                   |            |
               8080 Model          |    Data    |
               +------------+      |            |
               |   Mixed    |      +------------+
               |    Code    |      | RSP Header |
               |    and     |  DS --> +------------+
               |    Data    |      |            |
               +------------+      |    Code    |
               | RSP Header |      |            |
     CS, DS --> +------------+  CS --> +------------+ <-- Low
```

Figure 5-1. 8080 and Small RSP Models

5.3 Multiple copies of RSPs

At system generation, GENCCPM can make up to 255 extra copies of an RSP, such that each copy generates a separate process running under Concurrent CP/M. GENCCPM accomplishes this by making multiple copies of the RSP, and initializing each to be a separate RSP. The number of copies made by GENCCPM can be fixed, or made dependent on a byte value in the System Data Area. To determine the number of copies to make, GENCCPM looks at two fields in the RSP Header. The format of the RSP Header is shown in Figure 5-2.

```
     Byte: 00h     02h       04h   05h       10h
           +---+---+----+----+-----+----------+
           | LINK  | SDATVAR | NCP | RESERVED |
           +---+---+----+----+-----+----------+
```

Figure 5-2. RSP Header format

If the SDATVAR field is non-zero, it is used as an offset of a byte value in the System Data Area, which contains the number of copies to be generated. The offset should indicate a value that is set by the user during system generation. The TMP RSP uses this feature by placing the offset of the NVCNS (Number of Virtual CoNSoles) field into the SDATVAR field. This way, a TMP is generated for each System Console specified by the user. If SDATVAR is 0, then the NCP byte in the RSP Header is used as the number of extra copies to make. If both of these fields in the RSP Header are 0, then no extra copies are made, and only a single RSP is created. The ECHO RSP is an example of the latter.

If the number of extra copies is determined by GENCCPM to be greater than 0, each copy of the RSP is given a unique copy number. The copy number is placed in the NCP field, and the ASCII equivalent is appended to the end of the Process Descriptor NAME field of each copy. If there is not enough space for the number in the PD NAME, part of the PD NAME is over written. For the example TMP RSP, GENCCPM makes the specified number of copies, and changes the NAME field in each copy to be TMP0, TMP1, TMP2, ..., and sets the NCP field to 0, 1, 2, ..., respectively.

5.3.1 8080 Model

When GENCCPM makes copies of an 8080 Model RSP, the CS, DS, ES, and SS fields in each copy's User Data Area are set to the paragraph address where the RSP is in memory after loading.

5.3.2 Small Model

If multiple copies of a Small Model RSP are to be generated, GENCCPM copies both the Code and Data Groups of the RSP, if the MEM field of the Process Descriptor is 0. See the P_CREATE system call for a description of the Process Descriptor format. GENCCPM sets the UDA fields CS to the Code Segment of the RSP, and DS, ES, and SS to the Data Segment of the RSP.

5.3.3 Small Model with Shared Code

If a Small Model RSP has a non-zero MEM field in its Process Descriptor, the Code Segment is assumed to be re-entrant. When copies are made of this type of

RSP, only the Data Group is copied. GENCCPM sets the UDA CS field for each copy to the paragraph address of the one Code Segment for the RSPs. The DS, ES, and SS, in each copied Data Segment, are set by GENCCPM to the paragraph address of the Data Segment for that particular copy.

5.4 Creating and initializing an RSP

An RSP that is to be invoked from a console, or through the P_CLI system call, must create a special queue called an "RSP Command Queue". Such an RSP is called a "Command RSP". This type of RSP usually performs some initialization routine, and then goes into a loop. The initialization routine consists of creating and opening an RSP Command Queue, as well as changing the priority to the default transient process priority. (Priority values with regard to RSPs are discussed below.)

The first step of the loop reads a message from the RSP Command Queue. The process that writes the message to the RSP Command Queue activates the associated RSP. After the RSP returns from the Q_READ system call, it obtains the system resources it needs, such as the calling process' console. Typically, the RSP process is assigned the console process by the CLI after the CLI has succeeded in writing the command tail to the RSP Queue. This is only true if the RSP Process Descriptor name matches the RSP Command Queue name. Refer to the P_CLI (Call Command Line Interpreter) system call description for information about how the CLI handles a command.

When the RSP completes its activities for the given command, it releases any system resources it has acquired, including the console, and restarts the loop by reading from its RSP Command Queue. A Command RSP is a single process, and is a serially reusable resource; in other words, the RSP acts on one message at a time. When several processes attempt to invoke a single Command RSP, they wait as described in the Q_READ and Q_CREAD system call in Section 6, "System calls". Refer to these, and to the Q_WRITE and Q_CWRITE system call for further details.

Note: It is certainly possible to create RSPs that are invoked differently.

The format of the RSP Command Queue Message is shown below.

```
     Byte: 00h        02h                     082h
          +-----------+--------------------------+
          | PDADDRESS | COMMAND TAIL (129 bytes) |
          +-----------+--------------------------+
```

Figure 5-3. RSP Command Queue Message

The PDADDRESS is the offset relative to the System Data Area segment of the Process Descriptor of the process calling the RSP. A program that wants to invoke an RSP and is forming an RSP Command Queue Message, can find its Process Descriptor address by calling the P_PADR system call. The COMMAND TAIL usually contains what the TMP sends to the CLI, minus the command name, and is terminated with a zero byte.

When a command is entered at a console, the TMP performs a P_CLI system call. The P_CLI system call attempts to open a queue that has the RSP Flag ON, and has the same name as the command sent to the CLI. If the Q_OPEN is successful, the P_CLI system call attempts to assign the calling process' console to a process with the same name as the command. The P_CLI system call then creates an RSP Command Queue Message with the command tail sent to the CLI from the TMP, and writes it to the RSP Command Queue (refer to the discussion of the P_CLI and Q_WRITE system calls in Section 6, "System calls"). A transient program can use a Command RSP in the same manner, by writing directly to the appropriate RSP Command Queue. An advantage of using the P_CLI system call is that it looks for an RSP first, and only searches on disk for a CMD file if the RSP is not found.

When an RSP reads an RSP Command Queue Message, it often needs information about the calling process, such as which console, list device, drive, or user number to use. If an RSP is invoked through the P_CLI system call, the RSP is assigned the calling process' console, but if the RSP Command Queue is written to directly, the calling process might or might not assign its console to the RSP. A Command RSP can use the PD address in the Command RSP Message to find out what the default devices of the calling process are. The RSP should release any resources it assigns to itself when it is finished.

The beginning of the RSP Data Segment has a fixed formatr starting at offset 0. This data structure is the RSP Header. Note that, in the 8080 Model, the RSP Header is also in the Code Segment. After the RSP Header is a Process Descriptor starting at offset 0010h. A User Data Area and a stack must also be within the Data Segment, with the UDA placed at a paragraph boundary relative to the beginning of the Data Segment. If system calls assuming a default DMA Buffer are used, a 128-byte DMA Buffer must also exist. The DMA OFFSET field in the User Data Area should be set to the address of the DMA Buffer. When the process is created by Concurrent CP/M, the DMA SEGMENT field is initialized to the same value as the DS register. The DMA SEGMENT and OFFSET can also be set by calling F_DMASEG and F_DMAOFF once the RSP is running. The beginning of the RSP Data Segment is shown in Figure 5-4.

```
              :               :
              : Program Data  :
              : and RSP Stack :
              :               :
              +---------------+ 01A0h
              | Optional 8087 |
              | UDA extension |
              +---------------+ 0140h
              | User Data Area|
              +---------------+ 0040h
              | Process Descr.|
              +---------------+ 0010h
              |  RSP Header   |
        DS --> +---------------+ 0000h
```

Figure 5-4. RSP Data Segment

The RSP Header must be located at offset zero in the RSP Data Segment, the RSP Process Descriptor must be at offset 0010h, and the RSP User Data Area must begin on an even paragraph boundary.

5.4.1 The RSP Header

As discussed in Section 5.2, "RSP memory models", the number of copies made of an RSP is dependent on the values of the SDATVAR and NCP fields in the RSP Header. If no copies are desired, these fields must be zero. As a convenience, when Concurrent CP/M creates the RSP process, the LINK field in the RSP Header is set to the paragraph address of the System Data Area. The System Data Area can always be obtained by an RSP or transient program with the S_SYSDAT system call.

5.4.2 The RSP Process Descriptor

The RSP Process Descriptor should be initialized to zeroes, except for the PRIORITY, FLAGS, NAME, and USA SEGMENT fields. The PRIORITY field is usually initialized to 190. This is higher than transient programs and TMPs (200 and 198, respectively), but lower than the INIT process, which has a priority of 1. The description of the P_PRIORITY system call in Section 6, "System calls", contains more information about system priority assignments.

Starting an RSP at a priority of 190 ensures that the RSP is able to create and open an RSP Command Queue before it can be invoked through a TMP. RSPs such as ECHO usually set their priority to 200 after creating and opening their RSP Command Queue, and before attempting to read from the queue.

Note: There are no guarantees about the order in which the RSP processes are created by the Concurrent CP/M operating system. If one RSP must run before another, it must have a higher priority. Such is the case when one RSP uses a resource created by a second RSP; the second must run (at least during initialization) with a priority higher than the first.

The Process Descriptor SYS and KEEP Flags can be initialized in the RSP Data Segment (refer to P_CREATE in Section 6, "System calls", for further flag details). The SYS Flag allows a process to read and write to and from restricted system queues. This is discussed below with regard to RSP Command Queues. The KEEP Flag signals to the operating system that this process cannot be terminated. This flag is necessary if an RSP is not to be terminated when a Ctrl-C is typed on a console being used by the RSP. The 8087 Flag tells the system that a process is actively using the 8087 processor.

The NAME field of the RSP's Process Descritor is 8 bytes long. It is assumed to be left-justified and padded with blanks on the right. If an RSP Command Queue is going to be used to invoke the RSP through the CLI, the PD must have the same uppercase name as the Command Queue. The UDA field in the Process Descriptor must be the offset in paragraphs of the UDA relative to the RSP

Data Segment. GENCCPM restores the UDA field in the Process Descriptor to the actual UDA paragraph address when the system is generated.

If the PD field name is not the same as the Command Queue, the console is not assigned to the RSP by the CLI.

5.4.3 The RSP User Data Area

The User Data Area must have its SP field set to the offset of a three-word IRET structure, in the RSP's Data Segment. The offset is relative to the beginning of the Data Segment. The first of the three words is the offset of the code entry point for the RSP, relative to the beginning of the RSP Code Segment. Concurrent CP/M executes an IRET instruction to start the RSP using these three words for the IP, CS, and Flag registers, respectively. The CS value on the stack is initialized to be the CS field of the UDA, while the Flag value is set to 0200h (interrupts ON). The RSP stack must come immediately before these three words.

The initial values of the AX, BX, CX, DX, DI, SI, and BP registers are taken from the appropriate fields in the UDA.

The DMA OFFSET field should be set to the offset of the DMA Buffer in the RSP's Data Segment. Except for the SP and DMA OFFSET fields, and possibly the AX, BX, CX, DX, DI, SI, and BP fields, the remainder of the UDA fields should be initialized to 0. The CS, DS, ES, and SS fields are set by GENCCPM, as discussed in Section 5.3, "Multiple copies of RSPs".

If you include the 8087 extension in the UDA, you must initialize the CW field (Control Word) to 03FFh, and the SW (Status Word) field to 0 before system generation.

5.4.4 The RSP Stack

The RSP must reserve space for its stack, which is assumed to lie within the RSP's Data Segment. This stack must be large enough to accommodate what the RSP code needs, plus four levels (eight bytes) to handle possible hardware interrupts. We highly recommend that you reserve more than four extra levels of stack.

The SP field in the RSP's UDA points to the top of this stack; the top contains the three-word IRET instruction discussed above.

5.4.5 The RSP Command Queue

The RSP's Command Queue contains information that determines when it begins execution, and to which console it is attached. If an RSP is to be accessible from a console via the TMP, the Command Queue name must be in uppercase. The FLAGS field in the RSP Command Queue Descriptor must have the RSP bit ON. If this flag is not ON, the CLI will not write a message to the RSP Command QUeue, and instead attempts to load a transient program. The KEEP Flag should be set ON to protect the RSP QUEUE from inadvertent use of the Q_DELETE system call.

The RESTRICTED Flag (refer to the Q_MAKE system call in Section 6, "System calls") makes a queue accessible only by privileged processes. Privileged process have the SYS Flag ON in their Process Descriptor. If the RESTRICTED Flag is ON in an RSP Command Queue, then only privileged processes can invoke the related RSP. A lowercase letter in the RSP Command Queue name and the RESTRICTED Flag provides two methods of filtering access to an RSP QUEUE.

The Queue Descriptor of the RSP Command Queue must have a message length of 131 bytes. The format of this message is shown above. The number of messages is usually 1. If the Queue Descriptor is within 64 KB of the beginning of the System Data Area, buffer space for the Queue Descriptor must be allocated in the RSP. The BUFFER field in the Queue Descriptor must be the offset of this buffer, relative to the beginning of the RSP's Data Segment. The buffer size is the message length times the number of messages, usually 131 bytes.

Note: The queue buffer should be before the Queue Descriptor within the RSP Data Segment.

An RSP can certainly create other queues, besides the RSP Command Queue used with Command RSPs. However, any queue an RSP creates that lie within 64 KB of the System Data Area must have a buffer area pointed to by the BUFFER field in its Queue Descriptor. To be safe, the buffer should come before the Queue Descriptor in the RSP's Data Segment. It is assumed that the BUFFER field points to a buffer that is also within 64 KB of the System Data Area. If the Queue Descriptor is farther than 64 KB from the System Data Area, Concurrent CP/M uses buffer space in the System Data Area. Refer to the Q_MAKE system call in Section 6, "System calls", for further details.

In order to open the RSP Command Queue and subsequently read from it, a Queue Parameter Block and its associated buffer must be allocated in the RSP's Data Segment. These structures are treated just as in a transient process. For any queues created by an RSP, it is stressed that the queue buffer areas associated with the Queue Descriptor and the Queue Parameter Block are separate, distinct areas of storage.

5.4.6 Multiple processes within an RSP

An RSP can create child processes by calling the P_CREATE system call. Note that, if the Process Descriptor of the process being created is within 64 KB of the beginning of the System Data Area, the PD structure is used directly by Concurrent CP/M. Otherwise, the PD structure is copied into the PD table in the System Data Area.

5.5 Developing and debugging an RSP

The first RSP that you attempt should be very simple, on the order of complexity of the ECHO RSP listed in Appendix D. New RSPs should be developed and debugged as if they were transient processes, such as Concurrent CP/M CMD utilities, then converted into RSPs.

An RSP debugging session should proceed like an XIOS debugging session: first, load CP/M-86, then invoke DDT-86, and then bring up Concurrent CP/M. The "Concurrent CP/M System Guide" provides more information about running Concurrent CP/M under CP/M-86.

After reading in the CCPM.SYS file under DDT-86, find the RSPREG field of the System Data Segment (SYSDAT). The paragraph address of the SYSDAT is found in the A_BASE field of the Data Group Descriptor in the CCPM.SYS command file Header Record. The CMD file Header Record is described in Section 3.2, "Command file format", and the SYSDAT area is described in the S_SYSDAT system call in Section 6, "System calls". The RSPSEG field contains the paragraph address of the Data Segment of the first RSP in a linked list of the RSPs included by GENCCPM.

By using the Display Memory (D) command of DDT-86 to show memory at the segment RSPSEG, the name of the first RSP can be identified in the RSP's Process Descriptor. The LINK field in the RSP Header, which will be the first word in the RSPSEG segment, is the paragraph value of the next RSP's Data Segment. A zero in the LINK field means the end of the list of RSPs. Note that linkage information is lost once Concurrent CP/M is initialized. The LINK field of the RSP Header contains the System Data Segment once an RSP begins execution.

Once the RSP to be debugged is located, the initial code entry point can also be found. A discussed previously, the SP field in the RSP's UDA is the offset from the beginning of the RSP's Data Segment of the three-word IRET structure. The first word of the IRET structure contains the initial value of the IP register when Concurrent CP/M creates the RSP process. The initial value of the CS register is in the CS field also in the RSP's UDA. Once this is done, you can set break points in the RSP, similar to setting break points in XIOS system calls.

EOF

CCPMPRG6.WS4   (Concurrent CP/M Programmer's Reference Guide, Section 6)

(Retyped by Emmanuel ROCHE.)

Section 6: System calls

This section describes the Concurrent CP/M system calls in tabular form. It is intended both as an introduction to the calls, and as a reference for use during programming. You should be familiar with the material in Sections 1 through 5 before proceeding.

The first table, Table 6-1, describes the categories of Concurrent CP/M system call and their general uses. Table 6-2 summarizes the Concurrent CP/M system calls. Use it as a quick reference to find the system call that you need while programming. The system calls are broken down into functional groups. Immediately following is Table 6-3, a cross-reference showing the system calls in numerical order. Table 6-4 is an index providing the page numbers and fugure titles of commonly used data structures. Table 6-5 lists the error codes returned in register CX.

Table 6-1. System call categories

Format: Category
      Use

"C_" = Console System Calls
The Console System Calls handle I/O operations for virtual consoles on a character, string, and line basis, attach and detach consoles from processes, and return or change the number corresponding to the default virtual console.

"DEV_" = Device System Calls
The Device System Calls deal with flags and polling in managing system resources.

"DRV_" = Disk Drive System Calls
The Disk Drive System Calls manage Concurrent CP/M logical drives.

"F_" = File-Access System Calls
The File-Access System Calls include calls that operate on files within a directory, calls that operate on records within files, and miscellaneous system calls related to file I/O.

"L_" = List Device System Calls
The List Device System Calls write characters or strings to the default list device, attach or detach the default list device from calling processes, and return or change the number corresponding to the default list device.

"M_" = MP/M-86 Memory Management System Calls
The "M_" Memory Management System Calls are included for compatibility with MP/M-86. These calls allocate and free memory segements according to the MP/M-

86 segmentation algorithm.

"MC_" = CP/M-86 Memory Management System Calls
The "MC_" Memory Management System Calls allocate and free memory segments according to the CP/M-86 segmentation algortihm.

"P_" = Process/Program System Calls
The Process/Program System Calls create and terminate processes, call other processes, and perform other operations on processes.

"Q_" = Queue Management System Calls
The Queue Management System Calls create, delete, open, read from, and write to queues.

"S_" = System Calls
The System Calls return various types of system data, such as version numbers and addresses.

"T_" = Time System Calls
The Time System Calls set the system calendar and clock, and return the time from them in hours and minutes, or in hours, minutes, and seconds.

Table 6-2. Concurrent CP/M system calls

Dec	Mnemonic	Definition
Console I/O system calls		
149	C_ASSIGN	Assign default virtual console to another process.
146	C_ATTACH	Establish ownership of the default virtual console to the calling process; suspend process until console becomes available.
162	C_CATTACH	Conditionally establish ownership of the default virtual console by the calling process; return an error message if the device is unavailable.
110	C_DELIMIT	Set or return current String Output Delimiter. Used with C_WRITESTR.
147	C_DETACH	Detach default virtual console from the calling process.
153	C_GET	Return the virtual console number of the calling process.
109	C_MODE	Set or return Console Mode.
6	C_RAWIO	Perform Raw mode I/O with the default virtual console.
1	C_READ	Read a character from the default virtual console.

10	C_READSTR	Read an edited line from the default virtual console.
148	C_SET	Set or change the default virtual console for the calling process.
11	C_STAT	Obtain the input status of the default virtual console.
2	C_WRITE	Write a character to the default virtual console.
111	C_WRITEBLK	Write a specified number (block) of characters to the default virtual console.
9	C_WRITESTR	Write a string to the default virtual console, until delimiter.

Device system calls

131	DEV_POLL	Poll a non-interrupt-driven device.
133	DEV_SETFLAG	Set a system flag.
132	DEV_WAITFLAG	Wait for a system flag to be set before restoring the current process.

Disk drive system calls

38	DRV_ACCESS	Indicate access to specified drives.
27	DRV_ALLOCVEC	Get the address of the disk Allocation Vector.
13	DRV_ALLRESET	Reset all disk drives.
31	DRV_DPB	Return the segment and offset address of the Disk Parameter Block for the default disk of the calling process.
48	DRV_FLUSH	Write internal pending blocking/deblocking data buffers to disk.
39	DRV_FREE	Relinquish access to specified drives.
25	DRV_GET	Return the default drive of the calling process.
101	DRV_GETLABEL	Return the directory label data byte for the specified drive.
24	DRV_LOGINVEC	Return bit map of logged-in disk drives.
37	DRV_RESET	Reset the specified drives.
29	DRV_ROVEC	Return bit map vector of drives set to Read-Only.

14	DRV_SET	Set default drive of calling process.
100	DRV_SETLABEL	Set or update a directory label.
28	DRV_SETRO	Set the default drive to Read-Only.
46	DRV_SPACE	Return unallocated space on the specified drive.

Disk file system calls

30	F_ATTRIB	Set file attributes.
16	F_CLOSE	Close file.
19	F_DELETE	Delete file.
52	F_DMAGET	Return segment and offset address of Direct Memory Address buffer.
26	F_DMAOFF	Set the Direct Memory Access offset address.
51	F_DMASEG	Set Direct Memory Address buffer segment address.
45	F_ERRMODE	Set the BDOS Error mode.
42	F_LOCK	Lock record within file opened in Unlocked mode.
22	F_MAKE	Create file.
44	F_MULTISEC	Set the BDOS Multisector Count.
15	F_OPEN	Open file for record access.
152	F_PARSE	Parse an ASCII string, and initialize an FCB.
106	F_PASSWD	Set the default password.
36	F_RANDREC	Set the Random Record Number field in the FCB from the sequential record position.
20	F_READ	Read record sequentially.
33	F_READRAND	Read random record.
23	F_RENAME	Rename file.
17	F_SFIRST	Search for first matching directory FCB that matches the specified FCB.
35	F_SIZE	Return the size of a file.
18	F_SNEXT	Search for next matching directory FCB that matches the FCB specified in the F_SFIRST system call.

```
102    F_TIMEDATE     Return file's data and time stamps, and password mode.

 99    F_TRUNCATE     Truncate file to the specified Random Record Number.

 43    F_UNLOCK       Remove record locks.

 32    F_USERNUM      Set  or return the default user number of the  calling
                      process.

 21    F_WRITE        Write record sequentially.

 34    F_WRITERAND    Write random records.

103    F_WRITEXFCB    Create or update file's XFCB.

 40    F_WRITEZF      Write  random  records, and zero fill  any  previously
                      unallocated data blocks.

List device system calls
------------------------
158    L_ATTACH       Establish ownership of the default list device by  the
                      calling process; suspend the process until the  device
                      is available.

161    L_CATTACH      Conditionally establish ownership of the default  list
                      device  by the calling process; return error  code  if
                      the device is unavailable.

159    L_DETACH       Relinquish ownership of the default list device.

164    L_GET          Return  the default list device number of the  calling
                      process.

160    L_SET          Change  the  default  list  device  for  the   calling
                      process.

  5    L_WRITE        Write a character to the default list device.

112    L_WRITEBLK     Write  the specified number of characters  (block)  to
                      the default list device.

MP/M-86 compatible memory allocation system calls
-------------------------------------------------
128    M_ALLOC        Allocate   the  memory  segment  between   the   sizes
                      specified in the Memory Parameter Block to the calling
129    (same as 128)  process.

130    M_FREE         Free the specified memory segment.

CP/M-86 compatible memory allocation system calls
-------------------------------------------------
 56    MC_ABSALLOC    Allocate  a  specified amount of RAM,  as  above,  but
                      beginning at a specific address.
 54    MC_ABSMAX      Allocate  the  maximum amount of RAM  available  at  a
                 specified address.

 58    MC_ALLFREE     Free all memory owned by the calling process.

 55    MC_ALLOC       Allocate a segment of RAM, as specified in the  Memory
                 Control Block, to the calling process.

 57    MC_FREE        Free an area of RAM beginning at a specified  address,
                 and  extending to the end of the previously  allocated
                 memory area.

 53    MC_MAX         Allocate  the maximum amount of RAM available  in  the
                 system.

Process/Program system calls
----------------------------
157    P_ABORT        Terminate  a  process  specified by  name  or  Process
                 Descriptor address.

 47    P_CHAIN        Load, initialize, and jump to the program specified in
                 the DMA Buffer.

150    P_CLI          Interpret  and execute the specified command  line  by
                 calling the Command Line Interpreter (CLI).

144    P_CREATE       Create a subprocess.

141    P_DELAY        Suspend the calling process for a specified number  of
                 system clock ticks.

142    P_DISPATCH     Force  a dispatch operation; give up the CPU  resource
                 to the highest priority process ready to run.

 59    P_LOAD         Load the specified CMD file in memory; return its Base
                 Page segment address.

156    P_PDADR        Return  the address of the Process Descriptor  of  the
                 calling process.

145    P_PRIORITY     Set the priority of the calling process.

151    P_RPL          Invoke  a  system  call  from  a  Resident   Procedure
                 Library.

143    P_TERM         Terminate the calling process.

  0    P_TERMCPM      Terminate calling process unconditionally, release all
                 owned resources.

Queue system calls
------------------
138    Q_CREAD        Conditionally  read  a message from  a  system  queue;
                 return error code if a message is not available.
```

140   Q_CWRITE   Conditionally write a message to a system queue;
               return an error code if space is not available.

136   Q_DELETE   Delete a system queue.

134   Q_MAKE   Create a system queue.

135   Q_OPEN   Open a system queue for subsequent queue operations.

137   Q_READ   Read a message from a system queue; suspend calling
               process until message is available.

139   Q_WRITE   Write a message to a system queue; suspend calling
               process until space becomes available.

System information system calls
----
 12   S_BDOSVER   Return BDOS version number, CPU and operating system
               type.

 50   S_BIOS   Call specified CP/M-86 BIOS Character I/O routine.

163   S_OSVER   Return type and version number of Concurrent CP/M.

107   S_SERIAL   Return the Concurrent CP/M system serial number.

154   S_SYSDAT   Return address of the System Data Segment (SYSDAT).

Time system calls
----
105   T_GET   Obtain the system calendar and clock (hours and
               minutes only).

155   T_SECONDS   Return current system date and time; hours, minutes,
               seconds.

104   T_SET   Set internal system calendar and clock to specified
               value.

6.1 System call summary

Table 6-3 lists the Concurrent CP/M system calls in summary form, including the parameters that a process must pass when calling the system call, and the values that the system returns to the process.

Appendix A, "System calls summary by function number", lists the Concurrent CP/M system calls by function number, and includes all the information in Table 6-3.

Table 6-3. System call summary by name

Mnemonic	Dec	Input parameters	Returned values
C_ASSIGN	149	DX = .ACB	AX = Rtn Code
C_ATTACH	146	none	none
C_CATTACH	162	none	AX = Rtn Code
C_DELIMIT	110	DL = Out Delim	none
		= 0FFFFh	AL = Out Delim
C_DETACH	147	none	none
C_GET	153	none	AL = console #
C_MODE	109	DX = Con Mode	none
		= 0FFFFh	AX = Con Mode
C_SET	148	DL = console #	none
C_RAWIO	6	see def	see def
C_READ	1	none	AL = char
C_READSTR	10	DX = .Buffer	see def
C_STAT	11	none	AL = 1 if ready, 0 if not
C_WRITE	2	DL = char	none
C_WRITEBLK	111	DX = .CHCB	none
C_WRITESTR	9	DX = .Buffer	none
DEV_POLL	131	DL = Device	none
DEV_SETFLAG	133	DL = Flag	AX = Rtn Code
DEV_WAITFLAG	132	DL = Flag	AX = Rtn Code
DRV_ACCESS	38	DX = Drive Vector	none
DRV_ALLOCVEC	27	none	ES:AX = Alloc Addr
DRV_ALLRESET	13	none	see def
DRV_DPB	31	none	ES:AX = DPB Addr
DRV_FLUSH	48	none	see def
DRV_FREE	39	DX = Drive Vector	none
DRV_RESET	25	none	AL = Cur Drive
DRV_GETLABEL	101	DX = Drive #	AL = Label Data Byte
DRV_LOGINVEC	24	none	AX = Login Vector
DRV_RESET	37	DX = Drive Vector	AL = Err Code
DRV_ROVEC	29	none	AX = R/O Vector
DRV_SET	14	AL = Drive #	see def
DRV_SETLABEL	100	DX = .FCB	AL = Dir Code
DRV_SETRO	28	none	see def
DRV_SPACE	46	DL = Drive #	see def
F_ATTRIB	30	DX = .FCB	see def
F_CLOSE	16	DX = .FCB	AL = Dir Code
F_DELETE	19	DX = .FCB	AL = Dir Code
F_DMAGET	52	none	ES:AX = DMA Addr
F_DMAOFF	26	DX = .DMA	none
F_DMASEG	51	DX = .DMA Seg	none
F_ERRMODE	45	DL = Error Mode	none
F_LOCK	42	DX = .FCB	AL = Err Code
F_MAKE	22	DX = .FCB	AL = Dir Code
F_MULTISEC	44	DL = # of Records	AL = Rtn Code
F_OPEN	15	DX = .FCB	AL = Dir Code
F_PARSE	152	DX = .PFCB	see def
F_PASSWD	106	DX = .Password	none
F_RANDREC	36	DX = .FCB	R0,R1,R2

F_READ	20	DX = .FCB	AL = Err Code
F_READRAND	33	DX = .FCB	AL = Err Code
F_RENAME	23	DX = .FCB	AL = Dir Code
F_SFIRST	17	DX = .FCB	AL = Dir Code
F_SIZE	35	DX = .FCB	R0,R1,R2 & AL = Dir Code
F_SNEXT	18	none	AL = Dir Code
F_TIMEDATE	102	DX = .XFCB	AL = Dir Code
F_TRUNCATE	99	DX = .FCB	see def
F_UNLOCK	43	DX = .FCB	AL = Err Code
F_USERNUM	32	DL = 0FFh (Get)	AL = User #
		= User # (Set)	none
F_WRITE	21	DX = .FCB	AL = Err Code
F_WRITERAND	34	DX = .FCB	AL = Err Code
F_WRITEXFCB	103	DX = .XFCB	AL = Dir Code
F_WRITEZF	40	DX = .FCB	AL = Err Code
L_ATTACH	158	none	none
L_CATTACH	161	none	AX = Rtn Code
L_DETACH	159	none	none
L_GET	164	none	AL = List #
L_SET	160	DL = List #	none
L_WRITE	5	DL = char	none
L_WRITEBLK	112	DX = .CHCB	none
M_ALLOC	128	DX = .MPB	AX = Rtn Code
M_ALLOC	129	Same as above	Same as above
M_FREE	130	DX = .MPB	none
MC_ABSALLOC	56	DX = .MCB	see def
MC_ABSMAX	54	DX = .MCB	see def
MC_ALLFREE	58	none	none
MC_ALLOC	55	DX = .MCB	see def
MC_FREE	57	DX = .MCB	see def
MC_MAX	53	DX = .MCB	see def
P_ABORT	157	DX = .ABP	AX = Rtn Code
P_CHAIN	47	see def	none
P_CLI	150	DX = .CLBUF	none
P_CREATE	144	DX = .PD	none
P_DELAY	141	DX = # ticks	none
P_DISPATCH	142	none	none
P_LOAD	59	DX = .FCB	AX = BP Addr
P_PDADR	156	none	ES:AX = PD Addr
P_PRIORITY	145	DL = Priority	none
P_RPL	151	DX = .CPB	AX = result
P_TERM	143	DL = Term Code	AX = Rtn Code
P_TERMCPM	0	none	AX = Rtn Code
Q_CREAD	138	DX = .QPB	AX = Rtn Code
Q_CWRITE	140	DX = .QPB	AX = Rtn Code
Q_DELETE	136	DX = .QPB	AX = Rtn Code
Q_MAKE	134	DX = .QD	none
Q_OPEN	135	DX = .QPB	AX = Rtn Code
Q_READ	137	DX = .QPB	none

Q_WRITE	139	DX = .QPB	
S_BDOSVER	12	none	AX = Version #
S_BIOS	50	DX = .BD	AX = BIOS Rtn
S_OSVER	163	none	AX = Version #
S_SERIAL	107	DX = .serial #	serial #
S_SYSDAT	154	none	ES:AX = Sys Data Addr
T_GET	105	DX = .TOD	AL = seconds
T_SECONDS	155	DX = .TOD	TOD filled in
T_SET	104	DX = .TOD	none

Note: System calls, 3, 4, 7, and 8 are not supported by Concurrent CP/M.

Conventions used in Table 6-3:

#	= Number
ACB	= Assigned Control Block
APB	= Abort Parameter Block
Addr	= Address
BD	= BIOS Descriptor
BP	= Base Page
Char	= ASCII Character
CHCB	= CHaracter Control Block
CLBUF	= Command Line BUFfer
CPB	= Call Parameter Block
Con	= Console
Cur	= Current
Delim	= Delimiter
Dir	= Directory
DMA	= Direct Memory Address
Err	= Error
FCB	= File Control Block
MCB	= Memory Control Block
MPB	= Memory Parameter Block
Num	= Number
Out	= Output
PD	= Process Descriptor
PFCB	= Parse Filename Control Block
QD	= Queue Descriptor
QPB	= Queue Parameter Block
Rec	= Record
Rtn	= Return
Sys	= System
Term	= Termination
TOD	= Time Of Day
Vect	= Vector

Uppercase mnemonics refer to Data structures; see the function definition. A "." before a Data Structure means the byte offset of the Data Structure. A Return Code is either 0 for success, or 0FFFFh to indicate failure. When the Return Code in AX is 0FFFFh, CX is the Error Code (see Table 6-5). An Error Code returned in AL is specific to the BDOS system call that was made.

Table 6-4. Data structures index

Figure	Title	Page
2-1	FCB -- File Control Block	
2-2	Directory Label Format	
2-3	XFCB -- Extended File Control Block	
2-4	Directory Record with SFCB	
2-5	SFCB Subfields	
2-6	Disk System Reset	
3-1	CMD file Header Record format	
3-2	Group Descriptor format	
3-3	Concurrent CP/M Base Page values	
4-1	Initial program stack	
4-2	Concurrent CP/M 8080 Memory Model	
4-3	Concurrent CP/M Small Memory Model	
4-4	Concurrent CP/M Compact Memory Model	
4-5	Intel hexadecimal file format	
5-1	8080 and Small RSP Models	
5-2	RSP Header format	
5-3	RSP Command Queue Message	
5-4	RSP Data Segment	
6-1	ACB -- Assign Control Block	
6-2	Console Buffer format	
6-3	Drive, R/O, or Login Vector structure	
6-4	DPB -- Disk Parameter Block	
6-5	Disk Free Space Field format	
6-6	PFCB -- Parse Filename Control Block	
6-7	MCB -- Memory Control Block	
6-8	MPB -- Memory Parameter Block	
6-9	MFPB -- M_FREE Parameter Block	
6-10	APB -- Abort Parameter Block	
6-11	CLI Command Line Buffer	
6-12	PD -- Process Descriptor	
6-13	UDA -- User Data Area	
6-14	CPB -- Call Parameter Block	
6-15	QPB -- Queue Parameter Block	
6-16	QD -- Queue Descriptor	
6-17	BDOS Version Number format	
6-18	BIOS Descriptor format	
6-19	Operating Systems Version Number format	
6-20	SERIAL Number format	
6-21	SYSDAT Table	
6-22	TOD Time-Of-Day Structure	

Table 6-5. CX error code reports

Dec	Hex	Error report
0	00	No error
1	01	System call not implemented
2	02	Illegal system call number
3	03	Cannot find memory
4	04	Illegal flag number
5	05	Flag overrun
6	06	Flag underrun
7	07	No unused Queue Descriptors
8	08	No free queue buffer
9	09	Cannot find queue
10	0Ah	Queue in use
12	0Ch	No free Process Descritors
13	0Dh	No queue access
14	0Eh	Empty queue
15	0Fh	Full queue
16	10h	CLI queue missing
17	11h	No 8087 in system
18	12h	No unused Memory Descriptors
19	13h	Illegal console number
20	14h	No Process Descriptor match
21	15h	No console match
22	16h	No CLI process
23	17h	Illegal disk number
24	18h	Illegal filename
25	19h	Illegal filetype
26	1Ah	Character not ready
27	1Bh	Illegal Memory Descriptor
28	1Ch	Bad return from BDOS load
29	1Dh	Bad return from BDOS read
30	1Eh	Bad return from BDOS open
31	1Fh	Null command
32	20h	Not owner of resource
33	21h	No CSEG in load file
34	22h	Process Descriptor exists on Thread Root
35	23h	Could not terminate process
36	24h	Cannot attach to process
37	25h	Illegal list device number
38	26h	Illegal password
40	28h	External termination occurred
41	29h	Fixup error upon load
42	2Ah	Flag set ignored

6.2 Concurrent CP/M system calls

This section presents detailed information on the Concurrent CP/M system calls. Read the entire section through before attempting to use the system calls in a program, as many of them interact with one another.

6.2.1 Console I/O system calls

C_ASSIGN

Assign default virtual console to another process.

Entry Parameters:
    Register CL: 149 (95h)
            DX: ACB Address -- Offset
            DS: ACB Address -- Segment

Returned Values:
    Register AX: 0000h if assign OK,
               0FFFFh on Failure
            BX: Same as AX
            CX: Error Code

```
          +-------+-------+-------+-------+
      00h |  CNS  | MATCH |      PD       |
          +-------+-------+-------+-------+-------+-------+-------+-------+
      04h |                          NAME                                 |
          +-------+-------+-------+-------+-------+-------+-------+-------+
```

      Figure 6-1. ACB -- Assign Control Block

Table 6-6. ACB field definitions

Format: Field
       Definitions

CNS
Console to assign.

MATCH
Boolean; if 0FFh, the process being assigned the console must have the CNS as
its default console for a successful assign. If 00h, no check is made.

PD
Process ID of the process being assigned the console. If this field is zero, a
search is made of the Thread List for a process whose name is NAME. This field
must be either zero or a valid Process ID. If this value is not a valid ID? an
error occurs.

NAME
8-byte process name to search for. An error occurs if a process by this name
does not exist.

The C_ASSIGN system call directly assigns the specified console to a specified
process. This system call overrides the normal mechanism of the C_ATTACH and
C_DETACH system calls. The system call returns an error code if a process

other than the calling process owns the console. The system call ignores other processes waiting to attach to the specified console, and they continue to wait until the current owner either calls the C_DETACH system call, or terminates.

Refer to Table 6-5 for a list of error codes returned in CX.

C_ATTACH

Establish ownership of the default virtual console to the calling process; suspend process until console becomes available.

Entry Parameters:
    Register CL: 146 (92h)

The C_ATTACH system call attaches the default console to the caling process. If the console is already owned by the calling process, or if it is not owned by another process, the C_ATTACH system call immediately returns with ownership established and verified. If another process owns the console, the calling process waits until the console becomes available.

Refer to Table 6-5 for a list of error codes returned in CX.

C_CATTACH

Conditionally establish ownership of the default virtual console by the calling process; return an error message if the device is unavailable.

Entry Parameters:
    Register CL: 162 (0A2h)

```
Returned  Values:
    Register AX:  0000h if attach OK,
                  0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The C_CATTACH system call attaches the default console of the calling process, only if the console is currently un-attached.

If the console is currently attached to another process, the system call returns a value of 0FFh, indicating that the console could not be attached. The system call returns a value of 0 to indicate that either the console is already attached, or that it was unattached and a successful attach operation was made.

Refer to Table 6-5 for a list of error codes returned in CX.

C_DELIMIT

Set or return current String Output Delimiter. Used with C_WRITESTR.

```
Entry Parameters:
    Register CL: 110  (6Eh)
             DX: 0FFFFh (Get)
          or DL: Output Delimiter (Set)

Returned  Values:
    Register AL: Output Delimiter (no value if Set)
             BL: Same as AL
```

A program can set or interrogate the current Output Delimiter by calling C_DELIMIT. If register DX = 0FFFFh, then the current Output Delimiter is returned in register AL. Otherwise, C_DELIMIT sets the Output Delimiter to the value in register DL.

C_DELIMIT sets the string delimiter for C_WRITESTR. When a new process is created, the default delimiter value is set to a dollar sign ("$"). The default delimiter is not inherited from the parent process.

C_DETACH

Detach default virtual console from the calling process.

```
Entry Parameters:
    Register CL: 147  (93h)

Returned  Values:
    Register AX:  0000h if detach OK,
                  0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The C_DETACH system call detaches the default console from the calling process. If the default console is not attached to the calling process, no action is taken. If other processes are waiting to attach to the console, the process with the highest priority attaches the console. If there is more than one process with the same priority waiting for the console, it is given to the queue writing processes on a first-come, first-serve basis.

Refer to Table 6-5 for a list of error codes returned in CX.

C_GET

Return the virtual console number of the calling process.

```
Entry Parameters:
    Register CL: 153  (99h)
```

Returned Values:
    Register AL: Console number
           BL: Same as AL

The C_GET system call returns the default console number of the calling process.

C_MODE

Set or return Console Mode.

Entry Parameters:
    Register CL: 109 (6Dh)
           DX: 0FFFFh (Get)
            or Console Mode (Set)

Returned Values:
    Register AX: Console Mode (or no value)
           BX: Same as AX

A process can set or interrogate the Console Mode by calling C_MODE. If register DX = 0FFFFh, then the current Console Mode is returned in register AX. Otherwise, C_MODE sets the Console Mode to the value contained in register DX.

The Console Mode is a 16-bit system parameter that determines the action of certain Console I/O functions. Note that the Console Mode bits are numbered from right to left. The Console Mode is set to zero when a new process is created; it is not inherited from its parent. The definition of the Console Mode is:

```
      Bit 0 = 0 --> Normal status for C_STAT.
              1 --> Ctrl-C only status for C_STAT.

      Bit 1 = 0 --> Enable stop scroll, start scroll support.
              1 --> Disable stop scroll (Ctrl-S), start scroll (Ctrl-Q)
              support.

      Bit 2 = 0 --> Normal console output mode.
              1 --> Raw console output mode. Disables tab expansion for
              C_WRITE, C_WRITESTR, and C_WRITEBLK. Also disables printer
              echo (Ctrl-P) support.

      Bit 3 = 0 --> Enable Ctrl-C program termination.
              1 --> Disable Ctrl-C program termination.

      Bit 7 = 0 --> Enable Ctrl-O console output byte bucket.
              1 --> Disable Ctrl-O console output byte bucket.
```

C_RAWIO

Perform Raw mode I/O with the default virtual console.

```
Entry Parameters:
    Register CL: 6  (06h)
           DL: 0FFh (Input/Status)
            or 0FEh (Status)
            or 0FDh (Input)
            or Character (Output)

Returned  Values:
    Register AL: (Input/Status)
              00h (if no character)
              Character
              (Status)
               00h = No character
              0FFh = Ready
              (Input)
              Character
              (Output)
              No return value
           BL: Same as AL
```

The C_RAWIO system call allows the calling process to do raw I/O to its default console. Concurrent CP/M verifies that the calling process owns its default console before allowing any I/O.

A process calls the C_RAWIO system call by passing one of three different values shown in Table 6-7.

Table 6-7. C_RAWIO calling values

Value	Description
0FFh	Console input status command (if no character is ready, a 00h is returned, else the character is returned).
0FEh	Console status command (on return, register AL contains 00h if no character is ready; otherwise, it contains 0FFh.
0FDh	Console input command (if no character is ready, the calling process waits until one is typed). Input characters are not echoes to the screen.
ASCII character	If the parameter is less than 0FDh, the C_RAWIO system call assumes register DL contains a valid ASCII character, and sends it to the console.

The C_RAWIO system call places the calling process in Raw mode. The Ctrl-C, Ctrl-P, Ctrl-S, and Ctrl-O characters are not acted on by the PIN (Physical Input Process), but are passed on to the calling process when C_RAWIO is used.

Note: If the virtual console is in Ctrl-S mode, and the process that owns the virtual console then performs a C_RAWIO call, the Ctrl-S state is reset. Characters read with C_RAWIO are not echoed on the screen, thus allowing passwords and so forth to be entered in a secure manner.

C_READ

Read a character from the default virtual console.

```
Entry Parameters:
    Register CL: 1  (01h)

Returned  Values:
    Register AL: Character
             BL: Same as AL
```

The C_READ system call reads a character from the default console of the calling process. Before attempting the read, Concurrent CP/M internally verifies the ownership of the console. If the calling process does not own the console, it relinquishes the CPU resource until the calling process can attach to the console. Typically, a process that is created through the P_CLI system call owns its default console when it begins execution.

C_READ echoes characters read from the console. This includes the carriage return, line feed, and backspace characters. It expands tab characters (Ctrl-I) in columns of eight characters.

C_READ ignores the termination character (Ctrl-C) if the calling process cannot terminate (refer to the P_TERM system call). C_READ does not return until a character is typed on the console. The system suspends the calling process until a character is ready.

C_READSTR

Read an edited line from the default virtual console.

```
Entry Parameters:
    Register CL: 10  (0Ah)
             DX: BUFFER Address -- Offset
             DS: BUFFER Address -- Segment
```

The C_READSTR system call reads characters from the calling process' default console, and places them into the specified buffer. The format of the buffer is shown in Figure 6-2. C_READSTR performs line-editing system calls on the line as it is read from the console; it completes a line and return upon receiving a terminator character (carriage return or line feed) from the console, or when the maximum number of characters is reached. As in the C_READ system call, C_READSTR echoes all graphic characters read from the console. Concurrent CP/M verifies that the calling process owns its default console before allowing I/O to begin.

```
     0     1       2               MAX+2
     +-----+-------+-----------...--+
     | MAX | NCHAR | CHARACTERS...  |
     +-----+-------+-----------...--+
```

Figure 6-2. Console buffer format

Table 6-8. Console buffer field definition

Format: Field	Definition
MAX	Maximum number of characters that can be read into the buffer. This value must be initialized before calling the C_READSTR system call.
NCHAR	Actual number of characters read into the buffer as filled in by the C_READSTR system call.
CHARACTERS	Actual characters read from the console, as filled in by the C_READSTR system call.

C_READSTR recognizes a number of special characters used in editing the input line, as well as a set of special characters that actually control the calling process.

Table 6-9. C_READSTR line-editing characters

Format: Character	Function
RUB/DEL	Removes the last character from the line, and echoes it.
(Ctrl-E)	Echoes new line, a carriage return (Ctrl-M) and a line feed (Ctrl-J) to the screen, but does not affect the line buffer.
BACKSPACE (Ctrl-H)	Removes the last character from the line, and backspaces over that character.
TAB (Ctrl-I)	Echoes enough spaces to place the next character position at a tab stop. Tab stops are fixed at every eight character of the physical line.
LINE FEED (Ctrl-J)	Terminates the input line. The C_READSTR system call does not echo a terminating character, nor does it place the character in the line buffer.
CARRIAGE RETURN (Ctrl-M) "ENTER"	

Terminates the input line.

REDRAW (Ctrl-R)
Retypes the current line after echoing a new line.

(Ctrl-U)
Removes all of the current line from the line buffer, echoes a new line, and starts all over again.

(Ctrl-X)
Removes all of the current line from the line buffer, and echoes enough backspaces to return to the beginning of the line.

C_SET

Set or change the default virtual console for the calling process.

```
Entry Parameters:
    Register CL: 148  (94h)
           DL: Console Number

Returned  Values:
    Register AX:  0000h if successful,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code
```

The C_SET system call changes the calling process' default console to the value specified. If the console number specified is not one supported by this particular implementation of Concurrent CP/M, the system call return an error code, and does not change the default console.

Refer to Table 6-5 for a list of error codes returned in CX.

C_STAT

Obtain the input status of the default virtual console.

```
Entry Parameters:
    Register CL: 11  (0Bh)

Returned  Values:
    Register AX: 01h if character ready,
              00h if not ready
           BL: Same as AL
```

The C_STAT system call checks to see if a character has been typed at the default console. If the calling process is not attached to its default console, the C_STAT system call causes a dispatch to occur and return 00h (the Not Ready condition).

This system call sets the console to the Non-raw mode, allowing recognition of special control characters, such as the terminate character, Ctrl-C. Use C_RAWIO to obtain console status in Raw mode.

Note: If bit 0 is set in the Console Mode word, using the C_MODE function call, C_STAT only returns AL = 01h when a Ctrl-C is typed on the default console.

```
Entry Parameters:
    Register CL: 149  (95h)
           DX: ACB Address -- Offset
           DS: ACB Address -- Segment

Returned  Values:
    Register AX:  0000h if assign OK,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code
```

C_WRITE

Write a character to the default virtual console.

```
Entry Parameters:
    Register CL: 2  (02h)
           DL: ASCII character
```

The C_WRITE system call writes the specified character to the calling process' default console. As in the C_READ system call, Concurrent CP/M verifies that the calling process owns its default console before performing the operation. On output, C_WRITE expands tabs in columns of eight characters.

C_WRITEBLK

Write a specified number (block) of characters to the default virtual console.

```
Entry Parameters:
    Register CL: 111  (6Fh)
           DX: CHCB Address -- Offset
```

C_WRITEBLK sends the character string located by the CHaracter Control Block, CHCB, addressed in register pair DX to the console. If the Console Mode is in the Default state, C_WRITEBLK expands tab characters, Ctrl-I, in columns of eight characters.

The CHCB format is:

```
      bytes 0-1 : Offset  of character string
      bytes 2-3 : Segment of character string
      bytes 4-5 : Length  of character string to print
```

```
C_WRITESTR
----------

Write a string to the default virtual console, until delimiter.

Entry Parameters:
    Register CL: 9  (09h)
             DX: String Address -- Offset
             DS: String Address -- Segment
```

The C_WRITESTR system call prints an ASCII string starting at the indicated string address, and continuing until it reaches a dollar sign ("$") character (24h). "$" is the default string delimiter, and can be changed by the C_DELIMIT system call. C_WRITESTR writes this string to the calling process' default console.

Concurrent CP/M verifies that the calling process owns the console before writing the string. C_WRITESTR sets the console to a Non-raw state, and expands tabs in columns of eight characters, as does the C_WRITE system call.

Use the C_WRITESTR system call whenever possible, rather than the single-character system calls. The CPU overhead involved in handling the first character is the same as that for a single-character system call, but subsequent characters require as little as one-fifth the CPU overhead.

```
6.2.2 Device system calls
-------------------------

DEV_POLL
--------

Poll a non-interrupt-driven device.

Entry Parameters:
    Register CL: 131  (83h)
             DL: Device number

Returned  Values:
    Register AX:  0000h on Success,
                0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The DEV_POLL system call is used by the XIOS to poll non interrupt-driven devices. It should be used whenever the XIOS is waiting for a non interrupt event. The calling process relinquishes the CPU, and allows Concurrent CP/M to poll the device at every dispatch. The XIOS contains routines for each polling device number. These routines are called through the DEV_POLL system call, and they return whether the device is ready or not. When the device is ready, DEV_POLL restores the calling process to the Run state and returns. Upon return, the calling process knows the device is ready.

Refer to Table 6-5 for a list of error codes returned in CX.

DEV_SETFLAG

Set a system flag.

```
Entry Parameters:
    Register CL: 133  (85h)
             DL: Flag number

Returned  Values:
    Register AX:  0000h on Success,
                 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The DEV_SETFLAG system call is used by interrupt routines to notify the system that a logical interrupt has occurred. A process waiting for this flag is placed back into the Run state. If there is no process waiting, then the next process to wait for this flag returns successfully, without relinquishing the CPU. The system call detects an error if the flag has already been set, and no process has done a DEV_WAITFLAG call to reset it.

Note: If a process waiting for a specific flag to be set is aborted, the next DEV_SETFLAG call is ignored and an error code is returned in CX. In this case, the interrupt handler should continue to set call DEV_SETFLAG until it successfully sets the flag IP, and AX = 0 on return.

Refer to Table 6-5 for a list of error codes returned in CX.

DEV_WAITFLAG

Wait for a system flag to be set before restoring the current process.

```
Entry Parameters:
    Register CL: 132  (84h)
             DL: Flag number

Returned  Values:
    Register AX:  0000h on Success,
                 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The DEV_WAITFLAG system call is used by a process to wait for an interrupt. The process relinquishes the CPU until an interrupt routine calls the DEV_SETFLAG system call, which places the waiting process in the Run state. When the DEV_WAITFLAG returns to the calling process, the interrupt has occurred, or an error has occurred. An error occurs when a process is already waiting for the flag. If the flag was set before DEV_WAITFLAG was called, the routine returns successfully without relinquishing the CPU. This routine is usually used by the XIOS. The mapping between types of interrupts and flag numbers is maintained in the XIOS, although Concurrent CP/M reserves flags 0, 1, 2, and 3 for system use.

Refer to Table 6-5 for a list of error codes returned in CX.

6.2.3 Disk drive system calls

The Drive Vector, Read-Only Vector, and Login Vector are referenced or returned by several Concurrent CP/M Disk Drive system calls. The Drive, Read-Only, or Login vectors are 16-bit values specifying one or more drives, where the least significant bit corresponds to drive A, and the high-order bit corresponds to the sixteenth drive, labeled P. The format of the Drive, Read-Only, and Login vectors is illustrated below:

```
        +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    DRV | P   O   N   M   L   K   J   I   H   G   F   E   D   C   B   A |
        +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    Bit:  15  14  13  12  11  10  9   8   7   6   5   4   3   2   1   0
```

Figure 6-3. Drive, Read-Only, or Login vectors structure

DRV_ACCESS

Indicate access to specified drives.

```
Entry Parameters:
    Register CL: 38  (26h)
             DX: Drive Vector

Returned  Values:
    Register AL: Return Code
             AH: Extended Error
             BX: Same as AX
```

The DRV_ACCESS system call inserts a special open file item into the system Lock List for each specified drive. While the item exists in the Lock List, the drive cannot be reset by another process. The calling process passes the drive vector in register DX. The format of the drive vector is discussed at the beginning of Section 6.2.3, "Disk drive system calls".

The DRV_ACCESS system call inserts no items if insufficient free space exists in the Lock List to support all the new items, or if the number of items to be inserted puts the calling process over the Lock List open file maximum. If the BDOS Error mode is in the default mode (refer to the F_ERRMODE system call), the file system displays a message at the console identifying the error, and terminates the calling process. Otherwise, DRV_ACCESS returns to the calling process with register AL set to 0FFh, and register AH set to one of the following hexadecimal values:

```
       0Ah = Open file limit exceeded
       0Bh = No room in system Lock List
```

On successful calls, DRV_ACCESS returns with register AL set to 00h.

DRV_ALLOCVEC

Get the address of the disk Allocation Vector.

```
Entry Parameters:
    Register CL: 27  (1Bh)

Returned  Values:
    Register AX: Alloc Address -- Offset
             BX: Same as AX
             ES: Alloc Address -- Segment
```

Concurrent CP/M maintains an allocation vector in memory for each active disk drive. Some programs use the information provided by the allocation vector to determine the amount of free data space on a drive. Note, however, that the allocation information can be inaccurate if the drive has been marked Read-Only.

The DRV_ALLOCVEC system call returns the address of the allocation vector for the currently selected drive. If a physical error is encountered when the BDOS Error mode is in one of the return modes (refer to the F_ERRMODE system call), DRV_ALLOCVEC returns the value 0FFFFh in AX.

You can use the DRV_SPACE system call to directly return the number of free 128-byte records on a drive. The Concurrent CP/M utility, SHOW, finds a drive's free space by using the DRV_SPACE system call.

DRV_ALLRESET

Reset all disk drives.

```
Entry Parameters:
    Register CL: 13  (0Dh)

Returned  Values:
    Register AL:  00h if Successful,
                 0FFh on Failure
             BL: Same as AL
```

The DRV_ALLRESET system call restores the file system to a reset state, where all the disk drives are set to Read-Write (refer to the DRV_SETRO and DRV_ROVEC system calls), the default disk is set to drive A, and the default DMA address is reset to offset 0080h, relative to the current DMA segment address. This system call can be used, for example, by an application program that requires disk changes during operation. You can also use the DRV_RESET system call for this purpose.

This system call is conditional under Concurrent CP/M. If another process has a file open on any of the drives to be reset, and the drive is also Read-Only or removable, the DRV_ALLRESET system call is denied, and none of the specified drives are reset (see Section 2.17, "Reset, access, and free drive").

Upon return, if the reset operation is successful, DRV_ALLRESET sets register AL to 00h. Otherwise, it sets register AL to 0FFh. If the BDOS is not in one of the return error modes (refer to the F_ERRMODE system call), the file system displays an error message at the console identifying the process owning the first open file that caused the DRV_ALLRESET to be denied.

DRV_DPB

Return the segment and offset address of the Disk Parameter Block for the default disk of the calling process.

Entry Parameters:
    Register CL: 31 (1Fh)

Returned Values:
    Register AX: DPB Address -- Offset (0FFFFh on Physical Error)
             BX: Same as AX
             ES: DPB Address -- Segment

DRV_DPB returns the address of the XIOS-resident Disk Parameter Block (DPB) for the currently selected drive. The calling process can use this address to extract the disk parameter values.

If a physical error is encountered when the BDOS Error mode is one of the Return Error modes (refer to the F_ERRMODE system call), DRV_DPB returns the value 0FFFFh.

The Disk Parameter Block (DPB) contains the parameters that define the actual disk.

```
        +-----+-----+-----+-----+-----+-----+-----+-----+
    00h |    SPT    | BSH | BLM | EXM |    DSM    | DRM..
        +-----+-----+-----+-----+-----+-----+-----+-----+
    08h ..DRM | AL0 | AL1 |    CKS    |    OFF    | PSH |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    10h | PRM |
        +-----+
```

Figure 6-4. DPB -- Disk Parameter Block

Table 6-10. DPB field definitions

Format: Field
Definition

SPT -- Sectors Per Track
The number of Sectors Per Track equals the total number of physical sectors per track. Physical sector size is defined by PSH and PRM, described below.

BSH -- Allocation Block Shift Factor
BLM -- Allocation Block Mask
The data allocation block size determines the values of the data allocation Block Shift Factor and the allocation Block Mask. The Block Shift factor equals the the logarithm base two of the block logical size in 128-byte records, or BSH = LOG2 (BLS). The Block Mask equals the number of 128-byte records in an allocation Block minus 1, or BLM = (2**BSH-1). Refer to the "Concurrent CP/M System Guide" for valid block sizes, and BSH and BLM values.

EXM -- Extent Mask
The data block allocation size and the number of disk allocation blocks determine the value of the Extent Mask. The Extent Mask determines the maximum number of 16 KB extents that can be contained in a directory entry. It is equal to the maximum number of 16 KB extents per directory entry minus one. Refer to the "Concurrent CP/M System Guide" for EXM values.

DSM -- Disk Storage Maximum
The Disk Storage Maximum defines the total storage capacity of the drive. This is equal to the total number of allocation blocks minus one for the drive. DSM must be less than or equal to 7FFFh. If the disk uses 1024 bytes blocks (BSH = 3, BLM = 7), DSM must be less than or equal to 00FFh.

DRM -- Directory Maximum
The Directory Maximum defines the total number of directory entries for the drive. This is equal to the total number of directory entries minus one, that can be kept on this drive. The directory requires 32 bytes of disk per entry. The maximum directory allocation is 16 blocks, where the block size is determined by BSH and BLM.

AL0 -- Allocation Vector 0
AL1 -- Allocation Vector 1
The Directory Allocation Vectors determine the reserved directory allocation blocks.

CKS -- Checksum Vector Size
The Checksum Vector Size determines the required length of the directory checksum vector and the number of directory entries that the BDOS will checksum. The Checksum Vector Size is equal to the number of directory entries divided by 4, or CKS = (DRM+1)/4. If the media is fixed, CKS might be zero, no storage needs to be reserved, and the BDOS does not calculate directory checksums for the drive.

The high-bit of CKS (that it, >= 8000h) is set if the referenced drive is considered to be a non-removable media drive. Note that this modifies the rules for resetting the drive. For more information, refer to Section 2.15, "File byte counts".

OFF -- Track Offset
The Track Offset is the number of reserved tracks at the beginning of the disk. OFF is equal to the track number on which the directory starts.

PSH -- Physical Record Shift Factor
The Physical Record Shift Factor ranges from 0 to 5, corresponding to physical record sizes of 128, 256, 512, 1 KB, 2 KB, or 4 KB. It is equal to the logarithm base two of the physical record size divided by 128, or LOG2 (sector_size/128).

PRM -- Physical Record Mask
The Physical Record Mask ranges from 0 to 31, corresponding to physical record sizes of 128, 256, 512, 1 KB, 2 KB, or 4 KB. It is equal to the physical sector size divided by 128 minus 1, or (sector_size/128)-1.

For more information on DPB parameters, refer to the "Concurrent CP/M System Guide", Section 5.4, "Creating and initializing an RSP".

DRV_FLUSH

Write internal pending blocking/deblocking data buffers to disk.

```
Entry Parameters:
    Register CL: 48  (30h)
             DL: Purge Flag

Returned  Values:
    Register AL: Error Flag
             AH: Permanent Error
             BX: Same as AX
```

The DRV_FLUSH system call forces the write of any write-pending records contained in internal blocking/deblocking buffers. If register DL is set to 0FFh, DRV_FLUSH also purges all active data buffers after performing the writes. Programs that provide write with read verify support needed to purge internal buffers, to ensure that verifying reads actually access the disk, instead of returning data resident in internal data buffers. The Concurrent CP/M PIP utility is an example of such a program.

Upon return, the system call sets register AL to 00h if the flush operation is successful. If a physical error is encountered, DRV_FLUSH performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the calling process. Otherwise, it returns to the calling process with register AL set to 0FFh, and register AH set to one of the following physical error codes:

```
     01h = Disk I/O Error : permanent error
     02h = Read-Only Disk
```

DRV_FREE

Relinquish access to specified drives.

```
Entry Parameters:
    Register CL: 39  (27h)
             DX: Drive Vector
```

The DRV_FREE system call purges the system Lock List of all file and locked record items that belong to the calling process on the specified drives. DRV_FREE passes the drive vector in register DX.

DRV_FREE does not close files associated with purged open file Lock List items. In addition, if a process references a purged file with a BDOS system call requiring an open FCB, the system call returns a checksum error. A file that has been written to should be closed before making a DRV_FREE call to the file's drive, or data can be lost. Refer to Section 2.17, "Reset, acces, and free drive", for more information on this system call.

DRV_GET

Return the default drive of the calling process.

```
Entry Parameters:
    Register CL: 25  (19h)

Returned  Values:
    Register AL: Drive Number
             BL: Same as AL
```

The DRV_GET system call returns the calling process' currently selected default disk number. The disk numbers range from 0 through 15, corresponding to drives A through P.

DRV_GETLABEL

Return the directory label data byte for the specified drive.

```
Entry Parameters:
    Register CL: 101  (65h)
             DL: Drive

Returned  Values:
    Register AL: Directory Label Data Byte
             AH: Physical Error
             BX: Same as AX
```

The DRV_GETLABEL system call returns the directory label data byte for the specified drive. The calling process passes the drive number in register DL with 0 for drive A, 1 for drive B, continuing through 15 for drive P in a full

16-drive system. The format of the directory label data byte is shown below:

```
bit 7 : Require passwords for password protected files
    6 : Perform access time and date stamping
    5 : Perform update time and date stamping
    4 : Perform create time and date stamping
    0 : Directory label exists on drive
```

(Bit 0 is the least significant bit.)

DRV_GETLABEL returns the directory label data byte to the calling process in register AL. Register AL equal to 00h indicates that no directory label exists on the specified drive. If the system call encounters a physical error when the BDOS Error mode is in one of the return error modes (refer to the F_ERRMODE system call), it returns with register AL set to 0FFh, and register AH set to one of the following:

```
01h = Disk I/O Error : permanent error
04h = Invalid Drive : drive select error
```

DRV_LOGINVEC

Return bit map of logged-in disk drives.

```
Entry Parameters:
    Register CL: 24  (18h)

Returned  Values:
    Register AX: Login Vector
             BX: Same as AX
```

The DRV_LOGINVEC system call returns the Login Vector in register AX. The Login Vector is a 16-bit value with the least significant bit corresponding to drive A, and the high-order bit corresponding to the 16th drive, drive P. A 0 bit indicates that the drive is not logged-in, while a 1 bit indicates that the drive is logged-in. Refer to the beginning of Section 6.2.3, "Disk drive system calls", for a complete description of the Login Vector.

DRV_RESET

Reset the specified drives.

```
Entry Parameters:
    Register CL: 37  (25h)
             DL: Drive Vector

Returned  Values:
    Register AL: Return Code
             BL: Same as AL
```

The DRV_RESET system call is used to programmatically restore specified removable media drives to the reset state (a reset drive is not logged in, and is in Read-Write status). The passed parameter in register DX is a 16-bit vector of drives to be reset, where the least significant bit corresponds to drive A, and the high-order bit corresponds to the sixteenth drive, labeled P. Bit values of 1 indicate that the specified drive is to be reset. Refer to Section 2.17, "Disk drive system calls", for more information regarding the use of this system call.

This system call is conditional under Concurrent CP/M. If another process has a file open on any of the drives to be reset, the DRV_RESET system call is denied, and none of the drives are reset.

Upon return, if the reset operation is successful, DRV_RESET sets register AL to 00h. Otherwise, it sets register AH to 0FFh. If the BDOS Error mode is not in Return Error mode (refer to the F_ERRMODE system call), the system displays an error message at the console, identifying the process owning the first open file that caused the DRV_RESET request to be denied.

DRV_ROVEC

Return bit map vector of drives set to Read-Only.

```
Entry Parameters:
    Register CL: 29  (1Dh)

Returned  Values:
    Register AX: Read-Only Vector
             BX: Same as AX
```

The DRV_ROVEC system call returns a bit vector indicating which drives have the temporary Read-Only bit set. The Read-Only bit can only be set by a DRV_SETRO call.

Note: When the file system detects a change in the media on a drive, it automatically logs in the drive, and sets it to Read-Write.

The format of the Read-Only Vector is analogous to that of the Login Vector. The least significant bit corresponds to drive A; the most significant bit corresponds to drive P. For a complete description of the Read-Only Vector, refer to the beginning of this section.

DRV_SET

Set default drive of calling process.

```
Entry Parameters:
    Register CL: 14  (0Eh)
             DL: Selected Disk
```

Returned Values:
    Register AL: Error Flag
             AH: Physical Error
             BX: Same as AX

The DRV_SET system call designates the specified disk drive as the default disk for subsequent BDOS file operations. Set the DL register to 0 for drive A, 1 for drive B, continuing through 15 for drive P. DRV_SET also logs in the designated drive if it is currently in the reset state. Logging in a drive activates the drive's directory for file operations.

FCBs that specify drive code zero (DR = 00h) automatically reference the currently selected default drive. FCBs with drive code values between 1 and 16, however, ignore the selected default drive, and directly reference drives A through P.

Upon return, register AL equal to 00h indicates that the select operation was successful. If a physical error is encountered, DRV_SET performs different actions depending on the BDOS Error mode (refer to the F_ERRMODE system call).

If the BDOS Error mode is in the default mode, the system displays a message at the console, identifying the error, and terminates the calling process. Otherwise, DRV_SET returns to the calling process with register AL set to 0FFh, and register AH set to one of the following physical error codes:

      01h = Disk I/O Error : permanent error
      04h = Invalid Drive : drive select error

DRV_SETLABEL

Set or update a directory label.

Entry Parameters:
    Register CL: 100 (64h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned Values:
    Register AL: Directory Code
             AH: Physical or Extended Error
             BX: Same as AX

The DRV_SETLABEL system call creates a directory label, or updates the existing directory label for the specified drive. The calling process passes the address of an FCB containing the name, type, and extent fields to be assigned to the directory label. The name and type fields of the referenced FCB are not used to locate the directory label in the directory; they are simply copied into the updated or created directory label. Byte 12 of the FCB contains the user's specification of the directory label data byte.

The definition of the directory label data byte is:

```
bit 7 : Require passwords for password protected files
    6 : Perform access time and date stamping
    5 : Perform update time and date stamping
    4 : Perform create time and date stamping
    0 : Assign a new password to the directory label

(Bit 0 is the least significant bit.)
```

If the currently directory label is password protected, the correct password must be placed in the first 8 bytes of the current DMA, or have been previously established as the default password (refer to the F_PASSWD system call). If bit 0 of the directory label data byte is set to 1, it indicates that a new password for the directory label has been placed in the second 8 bytes of the current DMA.

The DRV_SETLABEL system call also requires that the referenced directory contains SFCBs in order to activate date and time stamping on the drive. If an attempt is made to activate date and time stamping when no SFCBs exist, the DRV_SETLABEL system call returns an error code, and performs no action. The Concurrent CP/M INITDIR untility initializes a directory for date and time stamping by placing an SFCB in every fourth entry of the directory.

Upon return, the DRV_SETLABEL system call returns a directory code in register AL with the value 00h if the directory label create or update was successful, or 0FFh if no space existed in the referenced directory to create a directory label. It also returns 0FFh if date and time stamping was requested, and the referenced directory did not contain SFCBs. Register AH is set to 00h in all of these cases.

If a physical or extended error is encountered, the DRV_SETLABEL system call performs different actions, depending on the BDOS Erro mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the file system displays a message at the console identifying the error, and terminates the calling process. Otherwise, the DRV_SETLABEL system call returns to the calling process with register AL set to 0FFh, and register AH set to one of the following physical or extended error codes:

```
01h = Disk I/O Error : permanent error
02h = Read-Only Disk
04h = Invalid Drive : drive select error
07h = Password Error
```

DRV_SETRO

Set the default drive to Read-Only.

```
Entry Parameters:
    Register CL: 28  (1Ch)

Returned  Values:
    Register AL: Return Code
             BL: Same as AL
```

The DRV_SETRO system call provides temporary write protection for the currently selected disk, by marking the drive as Read-Only. No process can write to a disk that is in the Read-Only state. You must perform a successful DRV_RESET operation to restore a Read-Only drive to the Read-Write state (refer to the DRV_ALLRESET and DRV_RESET system calls).

The DRV_SETRO system call is conditional under Concurrent CP/M. If another process has an open file on the drive, the operation is denied, and the system call returns the value 0FFh to the calling process. Otherwise, it returns a 00h. If the BDOS Error mode is not in Return Error mode (refer to the F_ERRMODE system call), the file system displays an error message at the console, identifying the process owning the first open file that caused the DRV_SETRO request to be denied.

Note that a drive in the Read-Only state cannot be reset by a process if another process has an open file on the drive.

DRV_SPACE

```
---------
```

Return unallocated space on the specified drive.

```
Entry Parameters:
    Register CL: 46  (2Eh)
           DL: Drive

Returned  Values:
    Register AL: Error Flag
           AH: Physical Error
           BX: Same as AX
    First 3 bytes of DMA Buffer filled in.
```

The DRV_SPACE system call determines the number of free sectors (128-byte records) on the specified drive. The calling process passes the drive number in register DL, with 0 for drive A, 1 for B, continuing through 15 for drive P. DRV_SPACE returns a binary number in the first 3 bytes of the current DMA Buffer. This number is returned in the format shown in Figure 6-5.

```
      +-----+-----+-----+
      | FS0 | FS1 | FS2 |
      +-----+-----+-----+

      FS0 = low    byte
      FS1 = middle byte
      FS2 = high   byte
```

Figure 6-5. Disk free space field format

Note that the returned free space might be inaccurate if the drive has been marked Read-Only.

Upon return, DRV_SPACE sets register AL to 00h, indicating that the operation was successful. However, if the BDOS Error mode is one of the return modes (refer to the F_ERRMODE system call), and a physical error occurs, it sets register AL to 0FFh, and register AH to one of the following values:

```
    01h = Disk I/O Erro : permanent error
    04h = Invalid Drive : drive select error
```

6.2.4 Disk file system calls

Most file-access system calls reference a File Control Block (FCB). This data structure is illustrated in Figure 2-1, "FCB -- File Control Block". Refer to Section 2.4, "File Control Block definition", for a comprehensive explanation of the FCB data structure, its initialization, and usage.

F_ATTRIB

Set file attributes.

```
Entry Parameters:
   Register CL: 30  (1Eh)
            DX: FCB Address -- Offset
            DS: FCB Address -- Segment

Returned  Values:
   Register AL: Directory Code
            BL: Same as AL
```

By calling the F_ATTRIB system call, a process can modify a file's attributes, and sets its last record byte count. Other BDOS system calls can interrogate these file parameters, but only F_ATTRIB can change them. The file attributes that can be set or reset by F_ATTRIB are F1' through F4', Read-Only (T1'), System (T2'), and Archive (T3'). The specified FCB contains a filename with the appropriate attributes set or reset. The calling process must ensure that it does not specify an ambiguous filename. Also, if the specified file is password protected, the correct password must be placed in the first eight bytes of the current DMA Buffer, or have been previously established as the default password (refer to the F_PASSWD system call).

Interface attribute F5' specified whether an extended file lock is to be maintained after the F_ATTRIB call. Interface attribute F6' specifies if the specified file's byte count is to be set. The interface attribute definitions are listed below:

```
    F5' = 0 : Do not maintain an extended file lock (default)
        = 1 : Maintain an extended file lock
    F6' = 0 : Do not set byte count (default)
        = 1 : Set byte count
```

If F5' is set and the referenced FCB specifies a file with an extended file lock, the calling process maintains the lock on the file. Otherwise, the file becomes available to other processes on the system. Section 2.11, "Extended file locking", describes extended file locking in detail.

If interface attribute F6' is set, the calling process must set the CR field of the referenced FCB to the new byte count value. A process can access a file's byte count value with the BDOS F_OPEN, F_SFIRST, and F_SNEXT system calls. File byte counts are described in Section 2.15, "File byte counts".

F_ATTRIB searches the FCB specified directory for an entry belonging to the current user number that matches the FCB specified name and type fields. The system call then updates the directory to contain the selected indicators and, if interface attribute F6' is set, the specified byte count value. Note that the last record byte count is maintained in the byte 13 of a file's directory FCB.

File attributeS T1', T2', and T3' are defined by Concurrent CP/M as described in Section 2.4.2, "File attributes". Attributes F1' through F4' of command files are defined as Compatibility Attributes, as described in Section 2.12, "Compatibility attributes". However, for all other files, attributes F1' through F4' are available for definition by the user. Attributes F5' through F8' are reserved as Interface Attributes, and cannot be used as file attributes. Interface attributes are described in Section 2.4.3, "Interface attributes".

An F_ATTRIB system call is not performed if the referenced FCB specifies a file currently open for another process. It is performed, however, if the referenced file is open by the calling process in Locked mode. However, the file's lock entry is purged when this is done, and the file system prevents continued read and write operations on the file. F_ATTRIB does not set the attributes of a file currently open in Read-Only or Unlocked mode for any process.

Making an F_ATTRIB system call for an open file can adversely affect the performance of the calling process. For this reason, you should close an open file before you call the F_ATTRIB system call.

Upon return, F_ATTRIB returns a directory code in register AL with the value 00h if the system call is successful, or 0FFh if the file specified by the referenced FCB is not found. Register AH is set to 00h in both cases.

If a physical or extended error is encountered, the F_ATTRIB system call performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the file system displays a message at the console identifying the error, and terminates the process. Otherwise, it returns to the calling process with register AL set to 0FFH, and register AH set to one of the following physical or extended error codes:

```
     01h = Disk I/O Error : permanent error
     02h = Read-Only Disk
     04h = Invalid Drive : drive select error
     05h = File open by another process
     07h = Password Error
     09h = Illegal ? in FCB
```

F_CLOSE

Close file.

```
Entry Parameters:
    Register CL: 16  (10h)
           DX: FCB Address -- Offset
           DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
           AH: Physical or Extended Error
           BX: Same as AX
```

The F_CLOSE system call performs the inverse operation of the F_OPEN system call. The referenced FCB must have been previously activated by a successful F_OPEN or F_MAKE system call. Interface attributes F5' and F6'specify how the file is to be closed, as shown below:

```
      F5' = 0, F6' = 0 --> Default close
         = 0,     = 1 --> Extend File Lock
         = 1,     = 0 --> Partial Close
         = 1,     = 1 --> Partial Close
```

The F_CLOSE system call performs the following steps, regardless of the interface attribute specification. First, it verifies that the referenced FCB has a valid checksum. If the checksum is invalid, F_CLOSE performs no action, and returns an error code.

If the checksum is valid and the referenced FCB contains new information because of write operations to the FCB, F_CLOSE permanently records the new information in the directory. If the FCB does not contain new information, the directory update step is bypassed. However, F_CLOSE always attempts to locate the FCB's corresponding entry in the directory, and returns an error code if the directory entry cannot be found.

If the F_CLOSE system call successfully performs the above steps, it performs different actions, depending on how the interface attributes are set. In default close operations, F_CLOSE decrements the file's open count, which is maintained in the file's system Lock List entry. If the open count decrements to zero, it indicates that the number of default close operations for the file matches the number of open operations.

If the open count decrements to zero, F_CLOSE permanently closes the file by performing the following steps. First of all, it removes the file's item from the system Lock List. If the FCB is opened in Unlocked mode, it also purges all record locks belonging to the file from the system Lock List. In addition, F_CLOSE invalidates the FCB's checksum, to ensure that the referenced FCB is not subsequently used with BDOS system calls that require an open FCB (for example, F_WRITE).

If the open count does not decrement to zero, F_CLOSE simply returns to the calling process, and the file remains open.

For partial close operations, F_CLOSE does not decrement the file's open count, and returns to the calling process. The file always remains open following a partial close request.

Closing a file with an extended file lock modifies the way F_CLOSE performs a permanent close. F_CLOSE only honors an extended lock request on a permanent close of a file opened in Locked mode. If these conditions are satisfied, F_CLOSE invalidates the FCB's checksum, but maintains the lock item. Thus, although the file is permanently closed, other processes cannot access the file. Section 2.11, "Extended file locking", describes extended file locking in detail.

Upon return, the F_CLOSE system call returns a directory code in register AL with the value 00h if the close operation is successful, or 0FFh if the file is not found. Register AH is set to 00h in both of these cases.

If a physical or extended error is encountered, the F_CLOSE system call performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the file system displays a message identifying the error at the console, and terminates the calling process. Otherwise, the F_CLOSE system call returns to the calling process with register AL set to 0FFH, and register AH set to one of the following physical or extended error codes:

```
      01h = Disk I/O Error : permanent error
      02h = Read-Only Disk
      04h = Invalid Drive : drives select error
      06h = Close Checksum Error
```

F_DELETE

Delete file.

```
Entry Parameters:
    Register CL: 19  (13h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
             AH: Physical or Extended Error
             BX: Same as AX
```

The F_DELETE system call removes files and/or XFCBs that matches the FCB addressed in register DX. The filename and filetype fields can contain wildcard file specifications (question marks in bytes 1 through 11), but byte 0 cannot be a wilcard, as it can be in the F_SFIRST and F_SNEXT system calls. Interface attribute F5' specifies the type of delete operation to be performed, as shown below:

F5' = 0 : Standard delete (Default mode)
    = 1 : Delete only XFCB's, and maintain an extended file lock

If any of the files specified by the referenced FCB are password protected, the correct password must be placed in the first eight bytes of the current DMA Buffer, or it must have been previously established as the default password (refer to the F_PASSWD system call).

For standard delete operations, the F_DELETE system call removes all directory entries belonging to files that match the referenced FCB. All disk directory and data space owned by the deleted files is returned to free space, and becomes available for allocation to other files. Directory XFCBs that were owned by the deleted files are also removed from the directory. If interface attribute F5' of the FCB is set to 1, F_DELETE deletes only the directory XFCBs matching the referenced FCB.

Note: If any of the files matching the input FCB specification fail the password check, are Read-Only, or are currently open by another process, then F_DELETE deletes no files or XFCBs. This applies to both types of delete operations.

Interface attribute F5' also specifies whether an extended file lock is to be maintained after the F_DELETE call. If F5' is set and the referenced FCB specifies a file with an extended lock, the calling process maintains the lock on the file. Section 2.11, "Extended file locking", describes extended file locking in detail.

A process can delete a file, that it currently has open, if the file is opened in locked mode. However, the BDOS returns a checksum error if the process makes a subsequent reference to the file with a BDOS system call requiring an open FCB. A process cannot delete files open in Read-Only or Unlocked mode.

Deleting an open file can adversely affect the performance of the calling process. For this reason, you should close an open file before you delete it.

Upon return, the F_DELETE system call returns a directory code in register AL with the value 00h if the delete is successful, or 0FFh if no file matching the referenced FCB is found. Register AH is set to 00h in both of these cases.
If a physical or extended error is encountered, F_DELETE performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call).

If the BDOS Error mode is in the default mode, the file system displays a message identifying the error at the console, and terminates the calling process. Otherwise, it returns to the calling process with register AL set to 0FFH, and register AH set to one of the following physical or extended error codes:

01h = Disk I/O Error : permanent error
02h = Read-Only Disk
03h = Read-Only File
04h = Invalid Drive : drive select error
05h = File opened by another process,
    or open in Read-Only or Unlocked mode.

```
    07h = Password Error
```

F_DMAGET

Return segment and offset address of Direct Memory Address buffer.

```
Entry Parameters:
   Register CL: 52  (34h)

Returned  Values:
   Register AX: DMA Address -- Offset
            BX: Same as AX
            ES: DMA Address -- Segment
```

F_DMAGET returns the current DMA Base Segment address in ES, with the current DMA Offset in AX.

F_DMAOFF

Set the Direct Memory Access offset address.

```
Entry Parameters:
   Register CL: 26  (1Ah)
            DX: DMA Address -- Offset
```

DMA is an acronym for "Direct Memory Access", which is often used with disk controllers that directly access the memory of the computer to transfer data to and from the disk subsystem. Under Concurrent CP/M, the current DMA is usually defined as the buffer in memory where a record resides before a disk write, and after a disk read operation. If the BDOS Multisector Count is equal to one (refer to the F_MULTISEC system call), the size of the buffer is 128 bytes. However, if the BDOS Multisector Count is greater than one, the size of the buffer must equal N * 128, where N equals the Multisector Count.

Some BDOS system calls also use the current DMA to pass parameters, and to return values. For example, BDOS system calls that check and assign file passwords require that the password be placed in the current DMA Buffer. As another example, DRV_SPACE returns its results in the first 3 bytes of the current DMA. When the current DMA is used in this context, the size of the buffer is memory is determined by the specific requirements of the system call.

When the P_CLI system call initiates a transient progra, it sets the DMA offset to 0080h, and the DMA Segment, or Base, to its initial Data Segment. DRV_ALLRESET also sets the DMA offset to 0080h. The F_DMAOFF system call can change this default value to another memory address. The DMA address remains at its current value until it is changed by an F_DMASEG, F_DMAOFF, or DRV_ALLRESET call.

```
F_DMASEG
--------

Set Direct Memory Address buffer segment address.

Entry Parameters:
    Register CL: 51  (33h)
           DX: DMA Segment Address

F_DMASEG  set  the segment value of the current DMA Buffer address.  The  word
parameter in DX is a paragraph address, and is used with the DMA offset  value
to specify the 20-bit address of the DMA Buffer. Refer to the F_DMAOFF  system
call for additional information.

Note  that, upon initial program loading, the default DMA base is set  to  the
address  of  the user's data segment (the initial value of DS),  and  the  DMA
offset  is  set to 0080h (which provides access to the default buffer  in  the
Base Page).


F_ERRMODE
---------

Set the BDOS Error mode.

Entry Parameters:
    Register CL: 45  (2Dh)
           DL: BDOS Error Mode

The BDOS Error mode is a system parameter maintained for each running process,
that  determines  how the file system handles physical  and  extended  errors.
Physical  and  extended  errors are described in  Section  2.18,  "BDOS  Error
handling". The BDOS Error mode has three states: the Default mode, the  Return
Error mode, and the Return and Display mode.

If  a  physical or extended error occurs when the BDOS Error mode  is  in  the
default  mode, the BDOS displays a system message at the  console  identifying
the error, and terminates the calling process.

If  a physical or extended error occurs when the BDOS Error mode is in  Return
Error  mode,  the  BDOS  sets  register AL  to  0FFH,  places  an  error  code
identifying the physical or extended error in register AH, and returns to  the
calling process.

If  a physical or extended error occurs when the BDOS Error mode is in  Return
and Display mode, the BDOS displays the system message before returning to the
calling process, and sets registers AH and AL as in the Return Error mode.

The F_ERRMODE system call sets the BDOS Error mode for the calling process  to
the mode specified in register DL. If register DL is set to 0FFh, the mode  is
set  to Return Error mode. If register DL is set to 0FEh, the mode is  set  to
Return and Display mode. If register DL is set to any other value, the mode is
set to the default mode.
```

F_LOCK

Lock record within file opened in Unlocked mode.

```
Entry Parameters:
    Register CL: 42  (2Ah)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
             AH: Physical Error
             BX: Same as AX
```

The F_LOCK system call allows a process to establish temporary ownership to particular records within a file. This system call is only supported for files open in Unlocked mode. If it is called for a file open in Locked or Read-Only mode, no locking action is performed, and a successful result is returned. This provides compatibility between Concurrent CP/M and CP/M-86.

The calling process passes the address of an FCB in which the random record field is filled with the Random Record Number of the first record to be locked. The number of records to be locked is determined by the BDOS Multisector Count (refer to the F_MULTISEC system call). The current DMA must also contain the 2-byte File ID returned by F_OPEN or F_MAKE when the referenced FCB was opened. Note that the File ID is only returned by the F_OPEN and F_MAKE system call when the Open mode is Unlocked.

Interface attribute F5' specifies the type of lock to perform. Interface attribute F6' specifies whether records have to exist in order to be locked. The F_LOCK interface attribute definitions are listed below:

```
      F5' = 0 : Exclusive lock (default)
          = 1 : Shared lock
      F6' = 0 : Lock existing records only (default)
          = 1 : Lock logical records
```

These options are described in detail in Section 2.14, "Concurrent file locking".

F_LOCK verifies that a locking conflict with another process does not exist for each of the records to be locked. However, if F_LOCK is called with attribute F6' reset, it also verifies that each record number to be locked exists within the specified file. Both tests are made before any records are locked.

Most F_LOCK requests require a new entry in the BDOS system Lock List. If there is insufficient space in the system Lock List to satisfy the lock request, or if the process record lock limit is exceeded, then F_LOCK does not lock any records, and returns an error code to the calling process.

Upon return, the F_LOCK system call sets register AL to 00h if the lock operation is successful. Otherwise, register AL contains one of the following error codes:

```
     01h = Reading unwritten data
     03h = Cannot close current extent
     04h = Seek to unwritten extent
     06h = Random Record Number out of range
     08h = Record locked by another process
     0Ah = FCB Checksum error
     0Bh = Unlocked file verification error
     0Ch = Process record Lock List limit exceeded
     0Dh = Invalid File ID
     0Eh = No Room in system Lock List
    0FFh = Physical error : refer to register AH
```

The system call returns error code 01h when it accesses a data block that has not been previously written.

The system call returns error code 03h when it cannot close the current extent prior to moving to a new extent.

The system call returns error code 04h when it accesses an extent that has not been created.

The system call returns error code 06h when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 08h if the specified record is locked by another process with an incompatible lock type.

The system call returns error code 0Ah if the referenced FCB failed the FCB Checksum test.

The system call returns error code 0Bh if the BDOS cannot locate the referenced FCB's directory entry when attempting to verify that the FCB contains current information.

The system call returns error code 0Ch if performing the lock request would require that the process consume more than the maximum allowed number of system Lock List entries.

The system call returns error code 0Dh when an invalid File ID is placed at the beginning of the current DMA.

The system call returns error code 0Eh when the system Lock List is full and performing the lock request would require at least one new entry.

The system call returns error code 0FFh if a physical error is encountered, and the BDOS Error mode is either Return Error mode, or Return and Display Error mode (refer to the F_ERRMODE system call). If the Error mode is in the default mode, the system displays a message at the console identifying the physical error, and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

```
    01h = Disk I/O Error : permanent error
    04h = Invalid Drive : drive select error
```

F_MAKE

Create file.

```
Entry Parameters:
    Register CL: 22  (16h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
             AH: Physical or Extended Error
             BX: Same as AX
```

The F_MAKE system call creates a new directory entry for a file under the current user number. It also creates an XFCB for the file if the referenced drive has a directory label that enables password protection on the drive, and the calling process assigns a password to the file.

The calling process passes the address of the FCB with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 set to extent number. Byte 12, the EX field, is usually set to 00h. Byte 32 of the FCB, the CR field, must be initialized to 00h, before or after the F_MAKE call, if the intent is to write sequentially from the beginning of the file.

Interface attribute F5' specifies the mode in which the file is to be opened. Interface attribute F6' specifies whether a password is to be assigned to the created file. The interface attributes are summarized below:

```
    F5' = 0 : Open in Locked mode (default)
        = 1 : Open in Unlocked mode
    F6' = 0 : Do not assign password (default)
        = 1 : Assign password to created file
```

When attribute F6' is set to 1, the calling process must place the password in the first 8 bytes of the current DMA Buffer, and set byte 9 of the DMA Buffer to the password mode. Note that F_MAKE only interrogates attribute F6' if the referenced drive's directory label has enabled password support. The XFCB Password mode is summarized below:

```
    XFCB Password mode
    ------------------
    Bit 7 = Read mode
        6 = Write mode
        5 = Delete mode
```

The F_MAKE system call returns with an error code if the referenced FCB names a file that currently exists in the directory under the current user number. If there is any possibility of duplication, an F_DELETE call should precede the F_MAKE call.

If the make file operation is successful, it activates the referenced FCB for record operations (opens the FCB) and initializes both the directory entry and the referenced FCB to an empty file. It also computes a checksum and assigns it to the FCB. BDOS system calls that require an open FCB (for example, F_WRITE) verify that the FCB Checksum is valid before performing their operation. If the file is opened in Unlocked mode, F_MAKE also sets bytes R0 and R1 in the FCB to a two-byte value called the "File ID". The File ID is a required parameter for the BDOS Lock Record and Unlock Record system calls. Note that the F_MAKE system call initializes all file attributes to 0.

The BDOS file system also creates an open file item in the system Lock List to record a successful F_MAKE oeration. While this item exists, no other process can delete, rename, truncate, or set the file attributes of this file.

A creation and/or update stamp is made for the created file if the referenced drive contains a directory label that enables creation and/or update time and date stamping, and the FCB extent number is equal to 0.

F_MAKE also creates an XFCB for the created file if the referenced drive conains a directory label that enables password protection, interface attribute F6' of the FCB is 1, and the FCB is an extent zero FCB. In addition, F_MAKE also assigns the password and password mode placed in the first nine bytes of the DMA to the XFCB.

Upon return, the F_MAKE system call returns a directory code in register AL with the value 00h if the make operation is successful, or 0FFh if no directory space is available. Register AH is set to 00h in both cases.

If a physical or extended error is encountered, the F_MAKE system call performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the process. Otherwise, it returns to the calling process with register AL set to 0FFh, and register AH set to one of the following physical or extended error codes:

01h	= Disk I/O Error : permanent error
02h	= Read-Only Error
04h	= Invalid Drive : drive select error
08h	= File Already Exists
09h	= Illegal ? in FCB
0Ah	= Open File Limit Exceeded
0Bh	= No Room in system Lock List

F_MULTISEC

Set the BDOS Multisector Count.

```
Entry Parameters:
    Register CL: 44  (2Ch)
             DL: Numbers of Sectors

Returned  Values:
    Register AL: Return Code
             BL: Same as AL
```

The F_MULTISEC system call provides logical record blocking under Concurrent CP/M. It enables a process to read and write from 1 to 128 logical records of 128 bytes at a time during subsequent BDOS read and write system calls. It also specifies the number of 128-byte records to be locked or unlocked by the F_LOCK and F_UNLOCK system calls.

F_MULTISEC sets the Multisector Count value for the calling process to the value passed in register DL. Once set, the specified Multisector Count remains in effect until the calling process makes another F_MULTISEC system call, and changes the value. Note that the P_CLI system call sets the Multisector Count to one when it initializes a transient process.

The Multisector COunt affects BDOS error reporting for the BDOS read and write system calls. With the exception of physical errors, if an error occurs during these system calls and the Multisector Count is greater than one, the system returns the number of records successfully processed in register AH.

Upon return, the system call sets register AL to 00h if the specified value is in the range of 1 to 128. Otherwise, it sets register AL to 0FFh.

F_OPEN

Open file for record access.

```
Entry Parameters:
    Register CL: 15  (0Fh)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
             AH: Physical or Extended Error
             BX: Same as AX
```

The F_OPEN system call activates the FCB for a file that exists in the disk directory under the currently active user number or user zero. The calling process passes the address of the FCB, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and byte 12 specifying the extent. Byte 12 is usually set to zero.

Interface attributes F5' and F6' of the FCB specify the mode in which the file is to be opened, as shown below:

```
      F5' = 0, F6' = 0 --> Open in Locked mode (Default mode)
```

```
         = 0,     = 1 --> Open in Read-Only mode
         = 1,     = 0 --> Open in Unlocked mode
         = 1,     = 1 --> Open in Read-Only mode
```

If the file is password protected in Read mode, the correct password must be placed in the first eight bytes of the current DMA, or have been previously established as the default password (refer to the F_PASSWD system call). If the current record field of the FCB, CR, is set to 0FFh, the F_OPEN system call returns the byte count of the last record of the file in the CR field. The last record byte count for a file can be set using the F_ATTRIB system call.

Note: The calling process must set the CR field of the FCB to 00h if the file is to be accessed sequentially from the first record.

The F_OPEN system call performs the following steps for files opened in Locked or Read-Only mode. If the current user is non-zero and the file to be opened does not exist under the current user number, the F_OPEN system call searches user 0 for the file. If the file exists under user 0 and has the system attribute (T2') set, the file is opened under user 0. The Open mode is automatically set to Read-Only when this is done.

The F_OPEN system call also performs the following action for files opened in locked mode. If the file has the Read-Only attribute (T1') set, the Open mode is automatically set to Read-Only. Note that Read-Only mode implies that the file can be concurrently accessed by other processes if they also open the file in Read-Only mode.

If the open operation is successful, F_OPEN activates the user's FCB for record operations as follows: F_OPEN copies the relevant directory information from the matching directory FCB into bytes D0 through D15 of the FCB. It also computes a checksum and assigns it to the FCB. All BDOS system calls that require an open FCB (for example, F_READ) verify that the FCB Checksum is valid before performing their operation.

If the file is opened in Unlocked mode, the F_OPEN system call sets bytes R0 and R1 of the FCB to a two-byte value called the "File ID". The File ID is a required parameter for the F_LOCK and F_UNLOCK system calls. If the Open mode is forced to Read-Only, F_OPEN sets interface attribute F8' to 1 in the user's FCB. In addition, the system call sets attribute F7' to 1 if the eferenced file is password protected in Write mode and the correct password was not passed in the DMA or did not match the default password. The BDOS does not support write operations for an activated FCB if interface attribute F7' or F8' is set to 1.

The BDOS file system also creates an open file item in the system Lock List to record a successful open file operation. While this item exists, no other process can delete, rename, or modify the file's attributes. In addition, this item prevents other processes from opening the file if the file is opened in Locked mode. It also requires that other processes match the file's Open mode if the file is opened in Unlocked or Read-Only mode. This item remains in the system Lock List until the file is permanently closed or until the process that opened the file terminates.

When the open operation is successful, the F_OPEN system call also makes an access time and date stamp for the opened file when the following conditions are satisfied: the referenced drive has a directory label that requests access date and time stamping, the FCB extent field is equal to zero, and the referenced drive is Read-Write.

Upon return, F_OPEN returns a directory code in register AL with the value 00h if the open is successful, of 0FFh if the file is not found. Register AH is set to 0 in both of these cases. If a physical or extended error is encountered, the F_OPEN system call performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message identifying the error at the console, and terminates the process. Otherwise, F_OPEN returns to the calling process with register AL set to 0FFh, and register AH set to one of the following physical or extended error codes:

```
        01H = Disk I/O Error : permanent error
        04h = Invalid Drive : drive select error
        05h = File is open by another process, or by the current process in an
               incompatible mode.
        07h = Password Error
        09h = Illegal ? in FCB
        0Ah = Open File Limit Exceeded
        0Bh = No Room in system Lock List
```

F_PARSE

Parse an ASCII string, and initialize an FCB.

```
Entry Parameters:
    Register CL: 152  (98h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AX: 0FFFFh if error
                 0000h if end of filename string
                 0000h if end of lineaddress of next item to parse
             BX: Same as AX
             CX: Error Code
```

```
        +----------+--------+
        | FILENAME | FCBADR |
        +----------+--------+
```

Figure 6-6. PFCB -- Parse Filename Control Block

Table 6-11. PFCB field definitions

Format: Field
        Description

FILENAME
Offset of an ASCII file specification to parse. The offset is relative to the same Data Segment as the FCB.

FCBADR
Offset of a File Control Block to initialize. The offset is relative to the same Data Segment as the PFCB.

The F_PARSE system call parses an ASCII file specification (FILENAME) and prepares a File Control Block (FCB). The calling process passes the address of a data structure called the Parse Filename Control Block (PFCB) in registers DX and DS. The PFCB contains the offset of the ASCII filename string, followed by the offset of the target FCB.

F_PARSE assumes the file specification to be in the following form:

    {D:}FILENAME{.TYP}{;PASSWORD}

where those items enclosed in curly brackets are optional.

The F_PARSE system call parses the first file specification it finds in the input string. First of all, it eliminates leading blanks and tabs. F_PARSE then assumes the file specification ends on the first delimiter it encounters that is out of context with the specific field it is parsing. For instance, if it finds a colon (":"), and it is not the second character of the file specification, the colon delimits the whole file specification.

The F_PARSE system call recognizes the following characters as delimiters:

space	
tab	
carriage return ("ENTER")	
null	(00h)
;	(semicolon) -- except before password field
=	(equal)
<	(less than)
>	(greater than)
.	(period) -- except after filename, and before filetype
:	(colon) -- except before filename, and after drive
,	(comma)
|	(vertical bar)
[	(left square bracket)
]	(right square bracket)

If the F_PARSE system call encounters a non-graphic character in the range 1 through 31 not listed above, it treats the character as an error.

The F_PARSE system call initializes the specified FCB as shown in Table 6-12.

Table 6-12. FCB initialization

Format: Byte number
    Explanation

Byte 0
The drive field is set to the specified drive. If the drive is not specified, the default value is used. 0=default, 1=A, 2=B, etc.

Bytes 1-8
The name is set to the specified filename. All letters are converted to uppercase. If the name is not eight characters long, the remaining bytes in the filename field are padded with blanks. If the filename has an asterisk ("*"), all remaining bytes in the filename field are filled with question marks ("?"). The system call returns an error if the filename is more than eight bytes long.

Bytes 9-11
The type is set to the specified filetype. If no type is specified, the type field is initialized to blanks. All letters are converted to uppercase. If the type is not three characters long, the remaining bytes in the filetype field are padded with blanks. If an asterisk is encountered, all remaining bytes are filled in with question marks. The system call returns an error if the type field is more than 3 bytes long.

Bytes 12-15
Filled in with zeros.

Bytes 16-23
The password field is set to the specified password. If no password is specified, this field is initialized to blanks. If the password is not eight characters long, remaining bytes are padded with blanks. All letters are converted to uppercase. The system call returns an error if the password field is more than eight bytes long.

Bytes 24-31
Reserved for system use.

If an error occurs, F_PARSE returns 0FFFFh in register AX indicating the error.

On a successful parse, the F_PARSE system call checks the next item in the FILENAME string. It scans for the first character that follows trailing balnks and tabs. If the character is a line feed (0Ah), a carriage return (0Dh), or a null character (00h), it returns a 0 indicating the end of the FILENAME string. If the next character is a delimiter, it returns the address of the delimiter. If the next character is not a delimiter, it returns the address of the first trailing blank or tab.

If the F_PARSE system call is to be used to parse a subsequent filename in the FILENAME string, the returned address should be advanced over the delimiter before placing it in the PFCB.

Refer to Table 6-5 for a list of error codes returned in CX.

F_PASSWD

Set the default password.

Entry Parameters:
    Register CL: 106 (6Ah)
             DX: Password Address -- Offset
             DS: Password Address -- Segment

The F_PASSWD system call allows a process to specify a password value before a file protected by the password is accessed. When the file system accesses a password protected file, it checks the current DMA and the default password for the correct value. If either value matches the file's password, full access to the file is allowed.

Concurrent CP/M maintains a default password for each process running on the system. A new process inherits its initial default password from its parent, the process creating the new process.

Note: Changing the default password does not affect other processes currently running on the system.

To make an F_PASSWD call, the calling process passes the adress of an 8-byte field containing the password.

F_RANDREC

Set the Random Record Number field in the FCB from the sequential record position.

Entry Parameters:
    Register CL: 36 (24h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned Values:
        Random Record field of FCB set

The F_RANDREC system call returns the Random Record Number of the next record to be accessed from a file that has been read or written sequentially to a particular point. The system call returns this value in the Random Record field, bytes R0, R1, and R2, of the addressed FCB. The F_RANDREC system call can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file, to extract the positions of various key fields. As each key is encountered, F_RANDREC is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record number minus one is placed into a table with the key for later retrieval.

After scanning the entire file and tabularizing the keys and their record numbers, you can move directly to a particular record by performing a random read using the corresponding Random Record Number that was saved earlier. The scheme is easily generalized when variable record lengths are involved, because the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

F_RANDREC can also be used when switching from a sequential read or write to a random read or write. A file is sequentially accessed to a particular point in the file, F_RANDREC is called to set the record number, and subsequent random read and write operations continue from the next record in the file.

F_READ

Read record sequentially.

```
Entry Parameters:
    Register CL: 20  (14h)
           DX: FCB Address -- Offset
           DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
           AH: Physical Error
           BX: Same as AX
```

The F_READ system call reads the next 1 to 128 128-byte records from a file into memory, beginning at the current DMA address. The BDOS Multisector Count (refer to the F_MULTISEC system call) determines the number of records to be read. The default is one record. The addressed FCB must have been previously activated by an F_OPEN or F_MAKE system call.

F_READ reads each record from the current record (CR) field in the FCB, relative to the current extent, then automatically increments the CR field to the next record position. If the CR field overflows, then F_READ automatically opens the next logical extent, and resets the CR field to zero for the next read operation. The calling process must set the CR field to 00h following the open call, if the intent is to read sequentially from the beginning of the file.

Upon return, the F_READ system call sets register AL to zero if the read operation is successful. Otherwise, register AL contains an error code identifying the error, as shown below:

```
      01h = Reading unwritten data (end-of-file)
      08h = Record locked by another process
      09h = Invalid FCB
      0Ah = FCB Checksum error
      0Bh = Unlocked file verification error
     0FFh = Physical error, refer to register AH
```

The system call returns error code 01h if no data exists at the next record position of the file. The no data situation is usually encountered at the end of a file. However, it can also occur if you try to read a data block that has not been previously written, or an extent that has not been created. These situations are usually restricted to files created or appended with the BDOS random write system calls (F_WRITERAND and F_WRITEZF).

The system call returns error code 08h if the calling process attempts to read a record locked by another process with an exclusive lock. This error code is only returned for files opened in Unlocked mode.

The system call returns error code 09h if the FCB is invalidated by a previous F_CLOSE system call that returned an error.

The system call returns error code 0Ah if the referenced FCB failed the FCB Checksum test.

The system call returns error code 0Bh if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call only returns this error for files opened in Unlocked mode.

The system call returns error code 0FFh if a physical error is encountered and the BDOS Error mode is in one of the return modes (refer to the F_ERRMODE system call). If the Error mode is in the default mode, the system displays a message at the console identifying the physical error, and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

```
      01h = Disk I/O Error : permanent error
      04h = Invalid Drive : drive select error
```

On all error returns, except for physical error returns (AL = 0FFh), F_READ sets register AH to the number of records successfully read before the error was encountered. This value can range from 0 to 127, depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.

F_READRAND

Read random record.

```
Entry Parameters:
    Register CL: 33  (21h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
             AH: Physical Error
             BX: Same as AX
```

The F_READRAND system call is similar to the F_READ system call, except that the read operation takes place at a particular Random Record Number, selected by the 24-bit value constructed from the three-byte, R0, R1, R2, field beginning at position 33 of the FCB. Note that the sequence of 24 bits is storead with the least signifcant byte first, R0, the middle byte next, R1, and the high byte last, R2. The Random Record Number can range from 0 to 262,143. This corresponds to a maximum value of 3 in byte R2.

To read a file with the F_READRAND system call, the calling process must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. The base extent might or might not contain any allocated data.

The F_READRAND system call reads the record specified by the random record field into the current DMA address. F_READRAND automatically sets the FCB extent and current record number values, EX and CR, but, unlike the F_READ system call, it does not advance the current record number. Thus, a subsequent F_READRAND call re-reads the same record. After a random read operation, a file can be accessed sequentially, starting from the current randomly accessed position. However, the last randomly accessed record is re-read or re-written when switching from random to sequential mode.

If the BDOS Multisector Count is greater than one (refer to the F_MULTISEC system call), F_READRAND reads multiple consecutive records into memory beginning at the current DMA. F_READRAND automatically increments the R0, R1, R2 fields of the FCB to read each record. However, it restores the FCB's Random Record Number to the first record's value upon return to the calling process.

Upon return, F_READRAND sets register AL to 00h if the read operation is successful. Otherwise, register AL contains one of the following error codes:

```
       01h = Reading unwritten data
       03h = Cannot close current extent
       04h = Seek to unwritten extent
       06h = Random Record Number out of range
       08h = Record locked by another process
       0Ah = FCB Checksum error
       0Bh = Unlocked file verification error
      0FFh = Physical error : refer to register AH
```

The system call returns error code 01h when it accesses a data block not previously written. This may indicate an end-of-file (EOF) condition.

The system call returns error code 03h when it cannot close the current extent prior to moving to a new extent.

The system call returns error code 04h when a read random operation accesses an extent that has not been created.

The system call returns error code 06h when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 08h if the calling process attempts to read a record locked by another process with an exclusive lock. This error code is only returned for files opened in Unlocked mode.

The system call returns error code 0Ah if the referenced FCB failed the FCB Checksum test.

The system call returns error code 0Bh if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call only returns this error for files open in Unlocked mode.

The system call returns error code 0FFh if a physical error is encountered and the BDOS Error mode is in one of the Return modes (refer to the F_ERRMODE system call). If the Error mode is in the default mode, the file system displays a message at the console identifying the physical error, and terminates the calling process. When a physical error is returned to the calling process, it is identified by the four low-order bits of register AH, as shown below:

```
    01h = Disk I/O Error : permanent error
    04h = Invalid Drive : drive select error
```

On all error returns, except for physical error returns, AL = 0FFh, F_READRAND sets register AH to the number of records successfully read before the error was encountered. This value can range from 0 to 127, depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.

F_RENAME

Rename file.

```
Entry Parameters:
    Register CL: 23  (17h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
             AH: Physical or Extended Error
             BX: Same as AX
```

The F_RENAME system call uses the referenced FCB to change all directory entries of the file specified by the drive and filename in bytes 0 to 11 of the FCB to the filename specified in bytes 17 through 27.

If the file specified by the first filename is password protected, the correct password must be placed in the first eight bytes of the current DMA Buffer, or have been previously established as the default password (refer to the F_PASSWD system call).

The calling process must also ensure that the filenames specified in the FCB are valid and un-ambiguous, and that the new filename does not already exist on the drive. F_RENAME uses the drive code at byte 0 of the FCB to select the drive. The drive code at byte 16 of the FCB is ignored.

Interface attribute F5' specifies whether an extended file lock is to be maintained after the F_ATTRIB call, as shown below:

```
        F5' = 0 --> Do not maintain an extended file lock (default)
            = 1 --> Maintain an extended file lock
```

If F5' is set and the referenced FCB specifies a file with an extended file lock, the calling process maintains the lock on the file. Otherwise, the file becomes available to other processes on the system. Section 2.11, "Extended file locking", describes extended file locking in detail.

A process can rename a file that it has open if the file is open in Locked mode. However, the BDOS returns a checksum error if the process subsequently references the file with a system call requiring an open FCB. A file open in Read-Only or Unlocked mode cannot be renamed by any process.

Renaming an open file can adversely affect the performance of the calling process. For this reason, you should close an open file before you rename it.

Upon return, the F_RENAME system call returns a directory code in register AL with the value 00h if the rename is successful, or 0FFh if the file named by the first filename in the FCB is not found. Register AH is set to 00h in both of these cases.

If a physical or extended error is encountered, the F_RENAME system call performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the process. Otherwise, it returns to the calling process with register AL set to 0FFh, and with register AH set to one of the following physical or extended error codes:

```
        01h = Disk I/O Error : permanent error
        02h = Read-Only Error
        03h = Read-Only File
        04h = Invalid Drive : drive select error
        05h = File open by another process
        07h = Password Error
        08h = File Already Exists
        09h = Illegal ? in FCB
```

F_SFIRST

Search for first matching directory FCB that matches the specified FCB.

```
Entry Parameters:
     Register CL: 17  (11h)
              DX: FCB Address -- Offset
              DS: FCB Address -- Segment
```

Returned Values:
    Register AL: Directory Code
            AH: Physical or Extended Error
            BX: Same as AX

The F_SFIRST system call scans the directory for a match with the referenced FCB. Two types of searches can be performed. For standard searches, the calling process initializes bytes 0 through 12 of the referenced FCB, with byte 0 specifying the drive directory to be searches, bytes 1 through 11 specifying the file or files to be searched for, and byte 12 specifying the extent. Byte 12 is usually set to 00h. An ASCII question mark (63, or 3Fh) in any of the bytes 1 through 12 matches all entries on the directory in the corresponding position. This facility, called "ambiguous file reference", can be used to search for multiple files on the directory. When called in the standard mode, F_SFIRST scans for the first file entry in the specified directory that matches the FCB and belongs to the current user number.

The F_SFIRST system call also initializes the F_SNEXT system call. After the F_SFIRST system call has located the first directory entry matching the referenced FCB, F_SNEXT can be called repeatedly to locate all remaining matching entries. In terms of execution sequence, however, the F_SNEXT call must follow either a F_SFIRST or F_SNEXT call with no other intervening BDOS file-access system calls.

If byte 0 of the referenced FCB is set to a question mark, F_SFIRST ignores the remainder of the referenced FCB, and locates the first directory entry residing on the current default drive. All remaining directory entries can be located by making multiple F_SNEXT calls. This type of search operation is not usually made by application programs, but it does provide complete flexibility to scan all directory entries. Note that this type of search operation must be performed to access a drive's directory label.

Upon return, the F_SFIRST system call returns a directory code in register AL with the value 0 to 3 if the search is successful, or 0FFh if a matching directory entry is not found. Register AH is set to zero in both of these cases. For successful searches, the current DMA is also filled with the directory record containing the matching entry, and the relative starting position is AL * 32. The directory information can be extracted from the buffer at this position.

If the directory has been initialized for date and time stamping, then an FCB resides in every fourth directory entry, and successful directory codes are restricted to the values 0 to 2. For successful searches, if the matching directory record is an extent zero entry, and if an SFCB resides at offset 96 within the current DMA Buffer, then the contents of (DMA Address + 96) = 21h, and the SFCB contains the time and date stamp information, and password mode, for the file. This information is located at the relative starting position of 97 + (AL * 10) within the current DMA, in the following format:

0-3	Create or Access Date and Time Stamp field
4-7	Update Date and Time Stamp field
8	Password Mode field

Refer to Section 2.8, "File date and time stamps: SFCBs", for more information about SFCBs.

If a physical error is encountered, the F_SFIRST system call performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the process. Otherwise, it returns to the calling process with register AL set to 0FFh, and with register AH set to one of the following physical error codes:

```
      01h = Disk I/O Error : permanent error
      04h = Invalid Drive : drive select error
```

F_SIZE

Return the size of a file.

```
Entry Parameters:
    Register CL: 35  (23h)
            DX: FCB Address -- Offset
            DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
            AH: Physical or Extended Error
            BX: Same as AX
    Random Record field of FCB set
```

The F_SIZE system call determines the virtual file size. This is the address of the record immediately following the end of the file. The virtual size of a file corresponds to the physical size if the file is written sequentially. If the file is written in random mode, gaps might exist in the allocation, and the file might contain fewer records than the indicated size. For example, if a single record with record number 262,143, the Concurrent CP/M maximum, is written to a file using the F_WRITERAND system call, then the virtual size of the file is 262,144 records, even though only one data block is actually allocated.

To compute file size, the calling process passes the address of an FCB with bytes R0, R1, and R2 present. The F_SIZE system call sets the random record field of the FCB to the Random Record Number + 1 of the last record in the file. If the R2 byte is set to 04h, and R0 and R1 are both zero, then the file contains the maximum record count, 262,144.

A process can append data to the end of an existing file, by calling F_SIZE to set the random record position to the end of the file, and then performing a sequence of random writes.

Note: The file need not be open in order to use F_SIZE. However, if the file is open in Locked mode and it has been extended by the calling process, the file must be closed before F_SIZE is called. Otherwise, F_SIZE returns an incorrect file size. F_SIZE returns the correct size for files open in Unlocked mode and Read-Only mode.

Upon return, F_SIZE returns a 00h in register AL if the file specified by the referenced FCB is found, or a 0FFh in register AL if the file is not found. Register AH is set to 00h in both cases.

If a physical or extended error is encountered, F_SIZE performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the process. Otherwise, F_SIZE returns to the calling process with register AL set to 0FFh, and with register AH set to one of the following physical or extended error codes:

```
        01h = Disk I/O Error : permanent error
        04h = Invalid Drive : drive select error
        09h = Illegal ? in FCB
```

F_SNEXT

Search for next matching directory FCB that matches the FCB specified in the F_SFIRST system call.

```
Entry Parameters:
     Register CL: 18  (12h)

Returned  Values:
     Register AL: Directory Code
              AH: Physical or Extended Error
              BX: Same as AX
```

The F_SNEXT system call is identical to F_SFIRST, except that the directory scan continues from the last entry that was matched. F_SNEXT returns a directory code in register AL, analogous to F_SFIRST.

Note: In execution sequence, a F_SNEXT call must follow either an F_SFIRST or another F_SNEXT with no other intervening BDOS file-access system calls.

F_TIMEDATE

Return file's data and time stamps, and password mode.

```
Entry Parameters:
     Register CL: 102  (66h)
              DX: FCB Address -- Offset
              DS: FCB Address -- Segment

Returned  Values:
     Register AL: Directory Code
              AH: Physical Error
              BX: Same as AX
```

The F_TIMEDATE system call returns the time and date stamp information, and password mode, for the specified file in byte 12, and bytes 24 through 31, of the specified FCB. The calling process passes the address of an FCB in which the drive, filename, and type fields have been defined.

If F_TIMEDATE is successful, it sets the following fields in the referenced FCB:

```
        Byte 12 -- Password mode field
        ------------------------------
        Bit 7 = Read mode
            6 = Write mode
            5 = Delete mode
```

Byte 12 equal to 0 indicates that the file has not been assigned a password.

```
        Byte 24-27 = XFCB Create or Access time stamp field
        Byte 28-31 = XFCB Update time stamp field
```

Upon return, F_TIMEDATE returns a directory code in register AL with the value 00h if the operation is successful, or 0FFh if the specified file is not found. Register AH is set to 00h in both of these cases.

If a physical or extended error is encountered, F_TIMEDATE performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the process. Otherwise, F_TIMEDATE returns to the calling process with register AL set to 0FFh, and with register AH set to one of the following physical error codes:

```
        01h = Disk I/O Error : permanent error
        04h = Invalid Drive : drive select error
        09h = Illegal ? in FCB
```

F_TRUNCATE

Truncate file to the specified Random Record Number.

```
Entry Parameters:
     Register CL: 99  (63h)
              DX: FCB Address -- Offset

Returned  Values:
     Register AL: Directory Code
              AH: Physical or Extended Error
              BX: Same as AX
```

The F_TRUNCATE system call sets the last record of the Random Record Number contained in the referenced FCB. The calling program passes the address of the FCB in register DX, with byte 0 of the FCB specifying the drive, bytes 1 through 11 specifying the filename and filetype, and bytes 33 through 35 (R0, R1, and R2) specifying the last record of the file. The last record number is a 24-bit value, stored with the least significant byte first (R0), the middle byte next (R1), and the high byte last (R2). This value can range from 0 to 262,143 (3FFFFh).

If the file specified by the referenced FCB is password protected, the correct password must have been placed in the first eight bytes of the current DMA Buffer, or have been previously established as the default password (refer to the F_PASSWD system call).

Interface attribute F5' specifies whether an extended file lock is to be maintained after the F_TRUNCATE call, as shown below:

```
F5' = 0 --> Do not maintain an extended file lock (default)
    = 1 --> Maintain an extended file lock
```

If F5' is set and the referenced FCB specifies a file with an extended file lock, the calling process maintains the lock on the file. Otherwise, the file becomes available to other processes on the system. Section 2.11, "Extended file locking", describes extended file locking in detail.

F_TRUNCATE requires that the Random Record Number field of the referenced FCB specify a value less than the current file size. In addition, if the file is sparse, the random record field must specify a region of the file where data exists.

A process can truncate a file that it currently has open if the file is opened in Locked mode and the file has not been extended during the open session. However, the BDOS returns a checksum error if the process makes a subsequent reference to the file with a BDOS system call requiring an open FCB. A process cannot truncate files open in Read-Only or Unlocked mode.

Truncating an open file is not recommended under Concurrent CP/M. F_TRUNCATE truncates a file based on the file's state in the directory. If a process attempts to truncate at a region of the file that has been allocated in memory but has not been recorded in the directory, F_TRUNCATE returns an error. Even when successful, an open file truncate can adversely affect the performance of the calling process. For these reasons, you should close an open file before you truncate it.

After completion, F_TRUNCATE returns a directory code in register AL with the value 00h if the operation is successful, or 0FFh if the file is not found or if the record number is invalid. In both cases, register AH is set to 00h.

If a physical or extended error is encountered, F_TRUNCATE performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, F_TRUNCATE returns to the calling program with register AL set to 0FFh, and register AH set to one of the following physical or extended error codes:

```
01H = Disk I/O Error : permanent error
```

```
        02h = Read/Only Disk
        03h = Read/Only File
        04h = Invalid Drive : drive select error
        05h = File Currently Open
        06h = Close Checksum Error
        07h = Password Error
        08h = File Already Exists
        09h = Illegal ? in FCB
        0Ah = Open File Limit Exceeded
        0Bh = No Room in system Lock List
```

F_UNLOCK

Remove record locks.

```
Entry Parameters:
    Register CL: 43  (2Bh)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
             AH: Physical Error
             BX: Same as AX
```

The F_UNLOCK system call unlocks one or more consecutive records previously locked by the F_LOCK system call. This system call is only supported for files open in Unlocked mode. If it is called for a file open in Locked or Read-Only mode, no unlocking action occurs, and a successful result is returned. Record locking and unlocking is described in detail in Section 2.14, "Concurrent file locking".

The calling process passes the address of an FCB in which the Random Record field if filled with the Random Record Number of the first record to be unlocked. The number of records to be unlocked is determined by the BDOS Multisector Count (refer to the F_MULTISEC system call). The current DMA must contain the 2-byte File ID returned by the F_OPEN or F_MAKE system call when the referenced FCB was opened. Note that the File ID is only returned by F_OPEN or F_MAKE when the file open mode is Unlocked.

If interface attribute F5' is set to 1, F_UNLOCK unlocks all locked records belonging to the calling process. The F_UNLOCK interface attribute definition is listed below:

```
        F5' = 0 --> Unlock records specified by Random Record Number and
                    BDOS Multisector Count (default)
            = 1 --> Unlock all locked records
```

F_UNLOCK ignores the FCB Random Record field and the BDOS Multisector Count when F5' is set.

F_UNLOCK does not unlock a record that is currently locked by another process.

However, the system call does not return an error if a process attempts to do that. Thus, if the Multisector Count is greater than one, F_UNLOCK unlocks all records locked by the calling process, skipping those records locked by other processes.

Some F_UNLOCK requests require a new entry in the BDOS system Lock List. If there is insufficient space in the system Lock List to satisfy the F_UNLOCK request, or if the process record Lock List limit is exceeded, then F_UNLOCK does not unlock any records, and returns an error code to the calling proces.

Upon return, F_UNLOCK sets register AL to 00h if the unlock operation was successful. Otherwise, register AL contains one of the following error codes:

```
     01h = Reading unwritten data
     03h = Cannot close current extent
     04h = Seek to unwritten extent
     06h = Random Record Number out of range
     0Ah = FCB Checksum error
     0Ch = Process record Lock List limit exceeded
     0Dh = Invalid File ID
     0Eh = No Room in system Lock List
    0FFh = Physical error : refer to register AH
```

The system call returns error code 01h when it accesses a data block which has not been previously written.

The system call returns error code 03h when it cannot close the current extent prior to moving to a new extent.

The system call returns error code 04h when it accesses an extent that has not been created.

The system call returns error code 06h when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 0Ah if the referenced FCB failed the FCB Checksum test.

The system call returns error code 0Ch if performing the unlock request would require that the process consume more than the maximum allowed number of system Lock List entries.

The system call returns error code 0Dh when an invalid File ID is placed at the beginning of the current DMA.

The system call returns error code 0Eh when the system Lock List is full and performing the unlock request would require at least one new entry.

The system call returns error code 0FFh if a physical error is encountered and the BDOS Error mode is one of the return modes (refer to the F_ERRMODE system call). If the Error mode is the Default mode, the system displays a message at the console identifying the physical error, and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

```
    01h = Disk I/O Error : permanent error
    04h = Invalid Drive : drive select error
```

F_USERNUM

Set or return the default user number of the calling process.

```
Entry Parameters:
    Register CL: 32  (20h)
             DL: 0FFh (Get)
              or User Number (to Set)

Returned  Values:
    Register AL: Current User Number (if Get)
             BL: Same as AL
```

A process can change or interrogate its current default user number by calling F_USERNUM. If register DL = 0FFh, then the system call returns the value of this user number in register AL. The value can range from 0 to 0Fh. If register DL is not 0FFh, then the system call changes the default user number to the value in DL, modulo 10h (the high nibble of DL is masked off).

Under Concurrent CP/M, a new process inherits its initial default user number from its parent, the process creating the new process. Changing the default user number does not change the user code of the parent. On the other hand, all child processes of the calling process inherit the new user number.

This convention is demonstrated by the operation of the TMP. When a command is typed, a new process is created with the same user number as that of the TMP. If this new process changes its user number, the TMP is unaffected. Once the new process terminates, the TMP displays the same user number in its prompt that it displayed before the command was entered and the child process was created.

F_WRITE

Write record sequentially.

```
Entry Parameters:
    Register CL: 21  (15h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
             AH: Physical Error
             BX: Same as AX
```

The F_WRITE system call writes 1 to 128 128-byte data records beginning at the current DMA address into the file named by the specified FCB. The BDOS Multisector Count (refer to the F_MULTISEC system call) determines the number of 128-byte records that are written. The default is one record. An F_OPEN or F_MAKE system call must have previously activated the referenced FCB.

F_WRITE places the record into the file at the position indicated by the CR byte of the FCB, and then automatically increments the CR byte to the next record position. If the CR field overflows, the system call automatically opens or creates the next logical extent, and resets the CR field to 00h in preparation for the next write operation. If F_WRITE is used to write to an existing file, then the newly written records overlay those already existing in the file. The calling process must set the CR field to 00h following an F_OPEN or F_MAKE system call if the intent is to write sequentially from the beginning of the file.

F_WRITE makes an update date and time stamp for the file if the following conditions are met: the referenced drive has a directory label that requests update date and time stamping, and the file has not already been stamped for update by a previous F_MAKE or F_WRITE system call.

Upon return, the F_WRITE system call sets register AL to zero if the write operation is successful. Otherwise, register AL contains an error code identifying the error, as shown below:

```
      01h = No available directory space
      02h = No available data block
      08h = Record locked by another process
      09h = Invalid FCB
      0Ah = FCB Checksum error
      0Bh = Unlocked file verification error
     0FFh = Physical error : refer to register AH
```

The system call returns error code 01h when it attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

The system call returns error code 02h when it attempts to allocate a new data block to the file, and no un-allocated data blocks exist on the selected disk drive.

The system call returns error code 08h if the calling process attempts to write to a record locked by another process, or a record locked by the calling process in Shared mode. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 09h if the FCB is invalidated by a previous F_CLOSE system call that returned an error.

The system call returns error code 0Ah if the referenced FCB failed the FCB Checksum test.

The system call returns error code 0Bh if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call returns this error only for files open in

Unlocked mode.

The system call returns error code 0FFh if a physical error was encountered and the BDOS is in Return mode, or Return and Display Error mode (refer to the F_ERRMODE system call). If the Error mode is in the Default mode, the system displays a message at the console identifying the physical error, and terminates the calling process. When the system call returns a physical error to the calling process, it is identified by register AH as shown below:

```
      01h = Disk I/O Error : permanent error
      02h = Read/Only Disk
      03h = Read/Only File, or
           File Opened in Read/Only Mode, or
           File password protected in Write mode
      04h = Invalid Drive : drive select error
```

On all error returns, except for physical error returns (AL = 0FFh), F_WRITE sets register AH to the number of records successfully written before the error was encountered. This value can range from 0 to 127, depending on the current BDOS Multisector Count. It is always set to zero when the Multisector Count is equal to one.

F_WRITERAND

Write random records.

```
Entry Parameters:
    Register CL: 34  (22h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
             AH: Physical Error
             BX: Same as AX
```

The F_WRITERAND system call is analogous to the F_READRAND system call, except that data is written to the disk from the current DMA address. If the disk extent and/or data block where the data is to be written is not already allocated, the BDOS automatically performs the allocation before the write operation continues.

In order to write to a file using the F_WRITERAND system call, the calling process must first open the base extent, extent 0. This ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, the calling process must create the base extent with the F_MAKE system call before an F_WRITERAND system call. The base extent might or might not contain data, but it records the file in the directory, so that it can be displayed by the Concurrent CP/M DIR utility. If a process does not open extent 0 and allocates data to some other extent, the file is invisible to the DIR utility.

The F_WRITERAND system call sets the logical extent and current record positions to correspond with the random record being written, but does not change the Random Record Number. Ths, sequential read or write operations can follow a random write, with the current record being re-read or re-written as the calling process switches from random to sequential mode.

F_WRITERAND makes an update date and time stamp for the file if the following conditions are met: the referenced drive has a directory label that requests update date and time stamping, and the file has not already been stamped for update by a previous F_MAKE or F_WRITE system call.

Upon return, the F_WRITERAND system call sets register AL to 00h if the write operation is successful. Otherwise, register AL contains one of the following error codes:

```
     02h = No available data block
     03h = Cannot close current extent
     05h = No available directory space
     06h = Random Record Number out of range
     08h = Record locked by another process
     0Ah = FCB Checksum error
     0Bh = Unlocked file verification error
    0FFh = Physical error : refer to register AH
```

The system call returns error code 02h when it attempt to allocate a new data block to the file. No un-allocated data blocks exist on the selected disk drive.

The system call returns error code 03h when it cannot close the current extent before moving to a new extent.

The system call returns error code 05h when it attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

The system call returns error code 06h when byte 35 (R2) of the referenced FCB is greater than 3.

The system call returns error code 08h if the calling process attempts to write to a record locked by another process, or a record locked by the calling process in Shared mode. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 0Ah if the referenced FCB failed the FCB Checksum test.

The system call returns error code 0Bh if the BDOS cannot locate the FCB's directory entry when attempting to verify that the referenced FCB contains current information. The system call returns this error only for files open in Unlocked mode.

The system call returns error code 0FFh if a physical error is encountered and the BDOS Error mode is in one of the Return modes (refer to the F_ERRMODE system call). If the Error mode is in the default mode, the system displays a

message at the console identifying the physical error, and terminates the
calling process. When a physical error is returned to the calling process, it
is identified by register AH, as shown below:

```
      01h = Disk I/O Error : permanent error
      02h = Read/Only Disk
      03h = Read/Only File, or
            File Opened in Read/Only Mode, or
            File password protected in Write mode
      04h = Invalid Drive : drive select error
```

On all error returns, except for physical error returns, AL = 0FFh,
F_WRITERAND sets register AH to the number of records successfully written
before the error was encountered. This value can range from 0 to 127,
depending on the current BDOS Multisector Count. It is always set to zero when
the Multisector Count is equal to one.

F_WRITEXFCB

Create or update file's XFCB.

```
Entry Parameters:
    Register CL: 103  (67h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Directory Code
             AH: Physical or Extended Error
             BX: Same as AX
```

The F_WRITEXFCB system call creates a new XFCB, or updates the existing XFCB
for the specified file. The calling process passes the address of an FCB in
which the drive, name, type, and extent fields have been defined. The FCB
extent field, if set, specifies the password mode and whether a new password
is to be assigned to the file. The format of the extent field byte is shown
below:

```
      FCB byte 12 (EX) -- XFCB password mode
      -------------------------------------
      Bit 7 = Read mode
          6 = Write mode
          5 = Delete mode
          0 = Assign new password to the file
```

If the FCB is currently password protected, the correct password must reside
in the first 8 bytes of the current DMA, or have been previously established
as the default password (refer to the F_PASSWD system call). If bit 0 is set
to 1, the new password must reside in the second 8 bytes of the current DMA.

Note: The F_WRITEXFCB system call does not create or update an XFCB if the
XFCB specifies a file open by another process. However, a process can update

or create an XFCB for a file that it has open in Locked mode.

Upon return, F_WRITEXFCB returns a directory code in register AL with the value 00h if the XFCB create or update was successful. F_WRITEXFCB returns 0FFh in register AL if no directory label existed on the specified drive, or the file specified in the FCB was not found, or no space existed in the directory to create an XFCB, or if the drive is not password enabled. F_WRITEXFCB also returns 0FFh if passwords are not enabled by the specified drive's directory label. Register AH is set to 00h in all of these cases.

If a physical or extended error is encountered, F_WRITEXFCB performs different actions, depending on the BDOS Error mode (refer to the F_ERRMODE system call). If the BDOS Error mode is in the default mode, the system displays a message at the console identifying the error, and terminates the calling process. Otherwise, F_WRITEXFCB returns to the calling process with register AL set to 0FFh, and register AH set to one of the following physical or extended error codes:

```
      01h = Disk I/O Error : permanent error
      02h = Read-Only Disk
      04h = Invalid Drive : drive select error
      05h = File open by another process, or open in Read/Only or
            Unlocked mode.
      07h = Password Error
      09h = Illegal ? in FCB
```

F_WRITEZF

Write random records, and zero fill any previously unallocated data blocks.

```
Entry Parameters:
    Register CL: 40  (28h)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AL: Error Code
             AH: Physical Error
             BX: Same as AX
```

The F_WRITEZF system call is similar to the F_WRITERAND system call, with the exception that it fills a previously un-allocated data blocks with zeros (00h) before writing the record. If this system call has been used to create a file, records accessed by an F_READRAND system call that contain all zeros identify un-written random records. Un-written random records in allocated data blocks of files created using the F_WRITERAND system call contain un-initialized data.

6.2.5 List device system calls

L_ATTACH

Establish ownership of the default list device by the calling process; suspend the process until the device is available.

Entry Parameters:
    Register CL: 158 (9Eh)

The L_ATTACH system call attaches the default list device of the calling process. If the list device is already attached to some other process, the calling process relinquishes the CPU until the other process detaches from the list device. When the list device becomes free, and the calling process is the highest priority process waiting for the list device, the attach operation occurs.

Refer to Table 6-5 for a list of error codes returned in CX.

L_CATTACH

Conditionally establish ownership of the default list device by the calling process; return error code if the device is unavailable.

Entry Parameters:
    Register CL: 161 (0A1h)

Returned Values:
```
    Register AX:  0000h if Attach OK,
                  0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The L_CATTACH system call attaches the default list device of the calling process only if the list device is currently available.

If the list device is currently attached to another process, the system call returns a value of 0FFh, indicating that the list device could not be attached. The system call returns a value of 00h to indicate that either the list device is already attached to the process, or that it was un-attached, and a successful attach operation was made.

Refer to Table 6-5 for a list of error codes returned in CX.

L_DETACH

Relinquish ownership of the default list device.

Entry Parameters:
    Register CL: 159 (9Fh)

Returned Values:
    Register AX: 0000h if Attach OK,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code

The L_DETACH system call detaches the default list device of the calling process. If the list device is not currently attached, no action takes place.

Refer to Table 6-5 for a list of error codes returned in CX.

L_GET

Return the default list device number of the calling process.

Entry Parameters:
    Register CL: 164 (0A4h)

Returned Values:
    Register AL: List Device Number
           BL: Same as AL

The L_GET system call returns the default list device number of the calling process.

L_SET

Change the default list device for the calling process.

Entry Parameters:
    Register CL: 160 (0A0h)
           DL: List Device Number

Returned Values:
    Register CX: Error Code

The L_SET system call sets the default list device for the calling process.

Refer to Table 6-5 for a list of error codes returned in CX.

L_WRITE

Write a character to the default list device.

Entry Parameters:
    Register CL: 5 (05h)
           DL: Character

The L_WRITE system call writes the specified character to the default list device of the calling process. Before writing the character, the system internally calls L_ATTACH to verify that the calling process owns its default list device.

L_WRITEBLK

Write the specified number of characters (block) to the default list device.

Entry Parameters:
```
    Register CL: 112  (70h)
             DX: CHCB Address -- Offset
```

L_WRITEBLK sends the character string specified in the CHaracter Control Block (CHCB) and addressed in register DX to the logical list device, LST:. The CHCB format is:

```
      Bytes 0-1 : Offset  of character string
            2-3 : Segment of character string
            4-5 : Length  of character string to print
```

6.2.6 Memory system calls

MP/M-86 compatible memory allocation system calls

There are two classes of Memory System Calls in Concurrent CP/M. The first class supports the MP/M-86 memory allocation scheme and contains two system calls, M_ALLOC and M_FREE. The second class contains six system calls, MC_ABSALLOC, MC_ABSMAX, MC_ALLFREE, MC_ALLOC, MC_FREE, and MC_MAX. These system calls support the CP/M-86 memory allocation scheme.

Note: The CP/M-86 memory calls are also supported under MP/M-86.

Many of the Memory system calls use the Memory Control Block (MCB) or the Memory Parameter Block (MPB) to pass parameters to and from the operating system. The format, structure and example programming equates for these data structures are presented below, along with example listings.

```
      +------+--------+-----+
      | BASE | LENGTH | EXT |
      +------+--------+-----+
```

      Figure 6-7. MCB -- Memory Control Block

Table 6-13. MCB field definitions

Format: Field
      Definition

BASE
The Segment Address of the beginning of the specified memory segment.

LENGTH
Length of the Memory Segment in paragraphs. The LENGTH field is set to the number of paragraphs wanted.

EXT
The EXT field is unused, but must be available.

```
;*****************************************************************
;*
;*      Memory Control Block definition
;*
;*****************************************************************

mcb_base        equu     word ptr 0
mcb_length      equ     word ptr mcb_base + word
mcb_ext         equ     byte ptr mcb_length + word

mcb_len         equ     mcb_ext + byte
```

Listing 6-1. Memory Control Block definition

```
+----+----+----+----+----+----+----+----+----+----+
|  START  |   MIN   |   MAX   | * 0000h | * 0000h |
+----+----+----+----+----+----+----+----+----+----+
```

Figure 6-8. MPB -- Memory Parameter Block

Table 6-14. MPB field definitions

Format: Field
Description

START
If non-00h, an absolute request at this paragraph.

MIN
Minimum memory needed (paragraphs)

MAX
Maximum memory wanted (paragraphs)

* 0000h
These fields must be 00h; they are used internally.

```
;*****************************************************************
;*
;*      Memory Parameter Block definition
;*
```

```
;************************************************************

mpb_start       equ     word ptr 0
mpb_min         equ     word ptr mpb_start + word
mpb_max         equ     word ptr mpb_min + word
mpb_pdadr       equ     word ptr mpb_max + word
mpb_flags       equ     word ptr mpb_pdadr + word

mpb_len         equ     mpb_flags + word

; MPB_FLAGS definition

mf_load         equ     0001h
mf_share        equ     0002h
mf_code         equ     0004h
```

Listing 6-2. Memory Parameter Block definition

M_ALLOC

Allocate the memory segment between the sizes specified in the Memory Parameter Block to the calling process.

```
Entry Parameters:
    Register CL: 128, 129  (80h, 81h)
             DX: MPB Address -- Offset
             DS: MPB Address -- Segment
    MPB filled in

Returned  Values:
    Register AX:  0000h on Success,
                  0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
    MPB_Start filled in
```

The M_ALLOC system call allows a program to allocate extra memory. A successful allocation allocates a contiguous memory segment, whose length is at least the MIN, and no more than the MAX, number of paragraphs specified in the MPB. The START field of the MPB is modified to be the starting paragraph of the memory segment. The MIN and MAX fields are modified to be the length of the memory segment in paragraphs. Memory Segments can be explicitly released through the M_FREE system call; Concurrent CP/M also releases all memory owned by a process at termination.

Note: MIN and MAX fields must be explicitly filled in. The MAX value must be greater than, or equal to, the MIN value.

Refer to Table 6-5 for a list of error codes returned in CX.

M_FREE

Free the specified memory segment.

```
Entry Parameters:
    Register CL: 130  (82h)
             DX: MFPB Address -- Offset
             DS: MFPB Address -- Segment

Returned  Values:
    Register AX:  0000h on Success,
                 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

```
      +---------+---------+
      |  START  | * 0000h |
      +---------+---------+
```

Figure 6-9. MFPB -- M_FREE Parameter Block

The M_FREE system call releases memory, starting at the START paragraph, to the end of a single previously allocated segment that contains the START paragraph. If the START paragraph is the same as that returned in the MPB of a memory allocation call, then M_FREE releases the whole memory segment. The "* 0000h" field must be initialized to zero.

Refer to Table 6-5 for a list of error codes returned in CX.

CP/M-86 compatible memory allocation system calls

MC_ABSALLOC

Allocate a specified amount of RAM, as above, but beginning at a specific address.

Allocate the maximum amount of RAM available at a specified address.

```
Entry Parameters:
    Register CL: 56  (38h)
             DX: MCB Address -- Offset
             DS: MCB Address -- Segment

Returned  Values:
    Register AL:  0000h on Success,
                 0FFFFh on Failure
             BL: Same as AL
             CX: Error Code
```

The MC_ABSALLOC system call allocates a memory area that starts at the address specified by the BASE field. The memory area's length is specified by the LENGTH field of the MCB. Upon return, register AL contains a 00h if the request was successful, and a 0FFh if the memory could not be allocated. If the calling process already owns the requested memory, no error is returned. This assures compatibility with CP/M-86.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_ABSMAX

Allocate the maximum amount of RAM available at a specified address.

```
Entry Parameters:
    Register CL: 54  (36h)
             DX: MCB Address -- Offset
             DS: MCB Address -- Segment
    MCB_Base filled in
    MCB_Length set to maximum number of paragraphs wanted

Returned  Values:
    Register AL:  0000h on Success,
                 0FFFFh on Failure
             BL: Same as AL
             CX: Error Code
    MCB_Length set to actual number of paragraphs allocated
```

In CP/M-86, system call 54 does not allocate memory but, under Concurrent CP/M, this system call allocates memory because other processes are competing for common memory. For compatibility with CP/M-86, MC_ABSALLOC (system call 56) does not return an error if there is a memory segment allocated at the absolute address.

MC_ABSMAX is used to allocate the largest possible region at the absolute paragraph boundary given by the BASE field of the MCB, for a maximum of LENGTH paragraphs. If the allocation is successful, the system call sets the LENGTH to the actual length. Upon return, register AL has the value 0FFh if no memory is available at the absolute address, and 00h if the request was successful.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_ALLFREE

Free all memory owned by the calling process.

```
Entry Parameters:
    Register CL: 58  (3Ah)
```

In the Concurrent CP/M environment, the MC_ALLFREE system call releases all of the calling process' memory, except the User Data Area (UDA). This system call is useful for system processes, and for sub-processes that share the memory of another process.

Note: This system call should not be used by processes running programs loaded into the Transient Program Areas (TPAs).

MC_ALLOC

Allocate a segment of RAM, as specified in the Memory Control Block, to the calling process.

```
Entry Parameters:
    Register CL: 55  (37h)
            DX: MCB Address -- Offset
            DS: MCB Address -- Segment
    MCB_Length filled in

Returned  Values:
    Register AL:  0000h on Success,
               0FFFFh on Failure
            BL: Same as AL
            CX: Error Code
    MCB_Base filled in
```

The MC_ALLOC system call allocates a memory area whose size is the LENGTH field of the MCB. MC_ALLOC returns the base paragraph address of the allocated region in the user's MCB. Upon return, register AL contains a 00h if the request was successful, and a 0FFh if the memory could not be allocated.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_FREE

Free an area of RAM beginning at a specified address, and extending to the end of the previously allocated memory area.

```
Entry Parameters:
    Register CL: 57  (39h)
            DX: MCB Address -- Offset
            DS: MCB Address -- Segment
    MCB_Base and MCB_Ext filled in

Returned  Values:
    Register AL:  0000h on Success,
               0FFFFh on Failure
            BL: Same as AL
            CX: Error Code
```

The MC_FREE system call is used to release memory areas allocated to the program. The value of the EXT field of the MCB controls the operation of this system call. If EXT = 0FFh, then the system call releases all memory areas allocated by the calling program. If the EXT field is 00h, the system call releases the memory area beginning at the specified BASE and ending a the end of the previously allocated memory segment.

Refer to Table 6-5 for a list of error codes returned in CX.

MC_MAX

Allocate the maximum amount of RAM available in the system.

```
Entry Parameters:
    Register CL: 53  (35h)
            DX: MCB Address -- Offset
            DS: MCB Address -- Segment
    MCB_Length contains the maximum number of paragraphs wanted

Returned  Values:
    Register AL:  0000h on Success,
               0FFFFh on Failure
            BL: Same as AL
            CX: Error Code
    MCB_Base filled in
    MCB_Length set to actual number of paragraphs allocated
```

In CP/M-86, system call 53 does not allocate memory but, under Concurrent CP/M, this system call allocates memory because other processes are competing for common memory. For compatibility with CP/M-86, MC_ABSALLOC (system call 56) does not return as error if there is a memory segment allocated at the absolute address.

MC_MAX allocates the largest available memory region that is less than or equal to the LENGTH field of the MCB in paragraphs. If the allocation is successful, the system call sets the BASE to the base paragraph address of the available area, and LENGTH to the paragraph length. Upon return, register AL has the value 0FFh if no memory is available, and 00h if the request was successful. The system call sets the EXT to 1 if there is additional memory for allocation, and 0 if no additional memory is available.

Refer to Table 6-5 for a list of error codes returned in CX.

6.2.7 Process/Program system calls

P_ABORT

Terminate a process specified by name or Process Descriptor address.

```
Entry Parameters:
    Register CL: 157  (9Dh)
            DX: APB Address -- Offset
            DS: APB Address -- Segment
     APB filled in
```

Returned Values:

```
     Register AX:  0000h on Success,
               0FFFFh on Failure
            BX: Same as AX
            CX: Error Code
```

```
           +------+------+------+------+------+------+
       00h |     PD      |    TERM     | CNS  | *00h |
           +------+------+------+------+------+------+------+------+
       06h |                      NAME                             |
           +------+------+------+------+------+------+------+------+
```

Figure 6-10. APB -- Abort Parameter Block

Table 6-15. APB field definitions

Format: Field
Definition

PD
Process Descriptor offset of the process to be terminated. If this field is zero, a match is attempted with the NAME and CNS fields to find the process. If this field is non-zero, the NAME and CNS fields are ignored.

TERM
Termination Code. This field corresponds to the termination code of the P_TERM system call. If the low_order byte of TERM is 0FFh, P_ABORT can abort a specified system process; if the termination code is not 0FFh, the system call can only terminate a user process. (A system process is identified by the System flag in the Process Descriptor's FLAG field.)

CNS
Default console of process to be aborted. If the PD field is 0, the P_ABORT system call scans the Thread List for a PD with the same NAME and CNS fields as specified in the APB. P_ABORT only aborts the first process that it finds. Subsequent calls must be made to abort all processes with the same NAME and CNS.

*00h
This field is reserved for sustem use, and must be set to zero.

NAME
Name of the process to be aborted. Combined with the CNS field, the NAME field is used to find the process to be aborted. This is only used if the PD filed is 0.

The P_ABORT system call permits a process to terminate another specified process. The calling process passes the address of a data structure called an "Abort Parameter Block", initialized as described above.

If the Process Descriptor address is know, it can be filled in, and the process name and console can be omitted. Otherwise, the Process Descriptor address field should be a 00h, and the process name and console must be specified. In either case, the calling process must supply the termination code, which is the same parameter passed to the P_TERM system call.

Refer to Table 6-5 for a list of error codes returned in CX.

P_CHAIN

Load, initialize, and jump to the program specified in the DMA Buffer.

```
Entry Parameters:
    Register CL: 47  (2Fh)
     DMA Buffer: Command Line

Returned  Values:
    Register AX: 0FFFFh = Could not find Command
```

The P_CHAIN system call provides a means of chaining from one program to the next without operator intervention. Although there is no passed parameters for this call, the calling process must place a command line terminated by a 00h byte in the default DMA Buffer.

Under Concurrent CP/M, the P_CHAIN system call releases the memory of the calling process before executing the command. The command is processed in the same manner as the P_CLI system call. If the command warrants the loading of a CMD file and the memory released is large enough for the new program, Concurrent CP/M loads the new program into the same memory area as the old program. The new program is run by the same process that ran the old program. The name of the process is changed to reflect the new program being run.

Parameter passing between the old and new programs is accomplished through the use of disk files, queues, or the command line. The command line is parsed and placed in the Base Page of the new program, in the manner documented in the P_CLI system call.

The P_CHAIN system call returns an error if no CMD file is found. If a CMD file is found, and an error occurs after it is successfully opened, the calling process terminates, as its memory has been released.

P_CLI

Interpret and execute the specified command line by calling the Command Line Interpreter (CLI).

```
Entry Parameters:
    Register CL: 150  (96h)
             DX: CLBUF Address -- Offset
             DS: CLBUF Address -- Segment
```

Returned Values:
    Register AX: 0000h on Success,
              0FFFFh on Error
          CX: Error Code

```
      0      1           129
      +------+--------...-+------+
      | *00h | COMMAND... | *00h |
      +------+--------...-+------+
```

Figure 6-11. CLI command line buffer

Table 6-16. Command line buffer field definitions

Format: Field	Definition
*00h	Must be set to zero for system use.
COMMAND	1-128 ASCII characters terminated with a null (00h) character.

The P_CLI system call obtains an ASCII command from the Command Line Buffer (CLBUF), and then executes it. If the calling process is attached to its default virtual console, the P_CLI system call assigns the virtual console to either the newly created process, or to the Resident System Process (RSP) that acts on the command. The calling process must re-attach to its default virtual console before accessing it.

P_CLI calls F_PARSE to parse the command line. If an error occurs in F_PARSE, P_CLI returns to the calling process with the error code set to the same code that F_PARSE returned.

If there is no disk specification for the command, P_CLI tries to open a system queue with the same name as the command. If the open operation is successful, and the queue is an RSP-type queue, P_CLI then writes the command tail to the RSP queue. If the queue is full, the system call returns an error code to the calling process. The P_CLI function also attempts to assign the calling process' virtual console to a process with the same name as the RSP queue. If the RSP queue cannot be found, the CLI assumes that the command is on disk, and continues.

The P_CLI system call opens a file with the filename being the command, and the filetype being "CMD". If the command has an explicit disk specification, and the F_OPEN system call fails, P_CLI returns an error code to the calling process. If there is no disk specification with the command, P_CLI attempts to open the command file on the system disk. If the F_OPEN system call succeeds, P_CLI checks the file to verify that the System attribute is ON. This search order is discussed in Section 2.9.2, "File attributes", of the "Concurrent CP/M User's Guide". If this second F_OPEN fails, or if the Dir attribute is ON, P_CLI returns an error code to the calling process.

Once the P_CLI system call succeeds in opening the command file, it calls the P_LOAD system call. The P_LOAD system call finds, and then loads the file into an appropriate memory space. If P_LOAD encounters any errors, the P_CLI system call returns to the calling process with the error code set by the P_LOAD system call.

A successful load operation establishes the command file in memory, with its Base Page partially initialized. The P_CLI system call then continues parsing the command tail, to set up the Base Page values from 0050h to 00FFh.

P_CLI initializes an unused Process Descriptor from the internal PD Table, a UDA (expanded UDA if 8087 processing is required), and a 96-byte stack area. The UDA and stack are dynamically allocated from memory. P_CLI then calls the P_CREATE system call. If P_CLI encounters an error in any of these steps, it releases all memory segments allocated for the new command, as well as the Process Descriptor, and then returns with the appropriate error code set.

Once the P_CREATE system call returns successfuly, the P_CLI system call assigns the calling process' default virtual console to the new process, and then returns.

The calling process should set its priority to less than the TMP (198), if it wants to attach to the virtual console after the created process releases it. Once the calling process has succesfully re-attached, it should set its priority back to 200.

Refer to Table 6-5 for a list of error codes returned in CX.

P_CREATE

Create a subprocess.

```
Entry Parameters:
    Register CL: 144  (90h)
           DX: PD Address -- Offset
           DS: PD Address -- Segment
    PD filled in

Returned  Values:
    Register AX:  0000h on Success,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code
```

The P_CREATE system call allows a process to create a subprocess within its own memory area. The child process shares all memory owned by the calling process at the time of the P_CREATE call. If the Process Descriptor (PD) is outside of the operating system area, the system copies it into a PD from the internal PD Table. The system call returns an error code if there are no more unused PDs in the table.

The User Data Area (UDA) can be anywhere in memory, but is required to be on a paragraph boundary. The only time the system copies the PD is if it is not within 64 KB of the System Data Segment.

Process Descriptors, as well as Queue Descriptors and Queue Buffers, are required to be within the System Data Segment, because they are linked together on various system lists, or are used by more than one process. Because of this, they cannot be in the Transient Process Area (TPA), where they cannot be protected.

More than one process can be created by a single P_CREATE call if the LINK field of the PD is non-zero. In this case, it is assumed to point to another PD within the same Data Segment. After it creates the first process, the system call checks the Process Descriptor's LINK field. Using this linked list of PDs, a single P_CREATE call can create multiple processes.

Note: The P_CREATE system call does not check the validity of the PD addresses passed by the calling process. An invalid PD address can cause Concurrent CP/M to crash if no hardware memory protection is available on the system.

Refer to Table 6-5 for a list of error codes returned in CX.

```
     +------+------+------+------+------+------+------+------+
 00h |    LINK     |   THREAD    | STAT | PRIOR|    FLAG     |
     +------+------+------+------+------+------+------+------+
 08h |                       NAME                            |
     +------+------+------+------+------+------+------+------+
 10h |     UDA     | DISK | USER |  RESERVED   |     MEM     |
     +------+------+------+------+------+------+------+------+
 18h |              RESERVED                   |   PARENT    |
     +------+------+------+------+------+------+------+------+
 20h | CNS  |      RESERVED      | LIST | Rsrvd|    SFLAG    |
     +------+------+------+------+------+------+------+------+
 28h |                     RESERVED                          |
     +------+------+------+------+------+------+------+------+
```

Figure 6-12. PD -- Process Descriptor

Table 6-17. PD field definitions

Format: Field
Definition

LINK
Link field for insertion on current system list. If this field's initial value is non-zero, it is assumed to point to another PD. This field is used to create more than one process with a single Create Process call.

THREAD
Link field for insertion on Thread List. Initialized to be zero (0000h).

STAT
Current Process activity. Initialized to be zero (00h). Activity codes are listed below:

00h RUN
The process is ready to run. The STAT field is always in this state when a process is examining its own Process Descriptor. The PD is on the Ready List. The currently running process is always at the head of Ready List.

01h POLL
The process is polling a device. The PD is on the Poll List.

02h DELAY
The process is delaying for a specified number of system tacks. The PD is on the Delay List.

06h Read Queue
The process is waiting to read a message from a system queue that is empty. The PD is on the Read Queue List whose root is in the Queue Descriptor of the system queue involved.

07h Write Queue
The process is waiting to write a message to a system queue whose buffer is full. The PD is on the Write Queue List, whose root is in the Queue Descriptor of the system queue involved.

08h FLAGWAIT
The process is waiting for a system flag to be set. The PD is in the flag table entry of the flag it is waiting for.

09h CIOWAIT
The process is waiting to attach to a Character I/O device (console or list), while another process owns it. The PD in on CQUEUE list whose root is in the Character Control Block of the device in question.

PRIOR
Current priority. Process scheduling is done based on this field. Typical user programs run at a priority of 200. 0 is the best priority, and 255 is the worst priority. The following is a list of priority assignments used by most Concurrent CP/M systems. User processes priorities should be from 200-254.

1	Initialization Process
2-31	Interrupt Handlers
32-63	System Processes
64-190	Undefined
191-197	Undefined
198	Terminal Message Process
199	Undefines
200	Default Priority for Transients
201-254	User Processes
255	Idle Process

FLAG
Bit field of flags determining run-time characteristics of a process. Initialize as needed. All undocumented flags are used internally, or are reserved for system use.

0001h SYS
System Process. Has privileged access to various features of Concurrent CP/M. This process can only be terminated if the termination code is 0FFh. This process can access restricted system queues. This flag is turned off if the calling process is not a system process.

0002h KEEP
This process cannot be terminated. This flag is turned off if the calling process is not a system process.

0004h KERNEL
This process resides within the operating system. This flag is turned off if the PD is not within the operating system.

0010h TABLE
This PD is copied into the PD from the PD table. When this process terminates, the PD is recycled into the PD table.

8000h 8087
This process is n 8087-running process.

NAME
Process Name. Eight bytes, all eight bits of each byte are used for matching process names.

UDA
Segment address of this process' User Data Area. Initialized to be the number of paragraphs from the beginning of the calling process' Data Segment. The User Data Area contains process information that is not needed between processes. It also contains the System Stack of each process. Refer to the UDA description below.

DISK
Current default disk.

USER
Current default user number.

MEM
Root of linked list of Memory Segment Descriptors that are owned by this process. Initialized to zero, except for re-entrant or shared code RSPs.

SFLAG
Second Flag. If bit 0 of SFLAG (01h) is set, the system suspends this process whenever it is switched out to the background, and runs it only when it is switched in to the foreground.

PARENT
Process that created this process. The P_CREATE system call sets this value at process creation. The parent field is set to zero if the parent terminates before the child.

CNS
Current default console's number. Initialized to be the default console number.

LIST
Current default list device's number. Initialized to be the default list device number.

RESERVED
Reserved for internal use. These fields must be initialized to zero (00h).

```
         +------+------+------+------+------+------+------+------+
     00h |  RESERVED   |  DMA OFFSET |        RESERVED           |
         +------+------+------+------+------+------+------+------+
     08h |                    RESERVED                           |
         +------+------+------+------+------+------+------+------+
     10h |                    RESERVED                           |
         +------+------+------+------+------+------+------+------+
     18h |                    RESERVED                           |
         +------+------+------+------+------+------+------+------+
     20h |     AX      |     BX      |     CX      |     DX      |
         +------+------+------+------+------+------+------+------+
     28h |     DI      |     SI      |     BP      |  RESERVED   |
         +------+------+------+------+------+------+------+------+
     30h |        RESERVED           |     SP      |  RESERVED   |
         +------+------+------+------+------+------+------+------+
     38h |          INT 0            |          INT 1            |
         +------+------+------+------+------+------+------+------+
     40h |        RESERVED           |          INT 3            |
         +------+------+------+------+------+------+------+------+
     48h |          INT 4            |        RESERVED           |
         +------+------+------+------+------+------+------+------+
     50h |     CS      |     DS      |     ES      |     SS      |
         +------+------+------+------+------+------+------+------+
     58h |         INT 224           |         INT 225           |
         +------+------+------+------+------+------+------+------+
     60h |                    RESERVED                           |
         +------+------+------+------+------+------+------+------+
     68h |                                                       |
         :                  USER SYSTEM STACK                    :
    0F8h |                                                       | 0FFh
         +------+------+------+------+------+------+------+------+
    100h |     CW      |     SW      |        RESERVED           |
         +------+------+------+------+                           |
    108h |                                                       |
         |                    RESERVED                           |
         :              (Optional 8087 Extension)                :
    158h |                                                       | 15Fh
         +------+------+------+------+------+------+------+------+
```

Figure 6-13. UDA -- User Data Area

The length of the UDA is 256 bytes (352 bytes if 8087 processing is required), and it must begin on a paragraph boundary.

Table 6-18. UDA field definition

Format: Field
Definition

DMA OFFSET
The initial DMA OFFSET for the new process. The segment address of the DMA is assumed to be the same as the initial Data Segment (refer to DS below).

AX,BX,CX,DX,DI,SI,BP
The initial register values for the new process. These are typically set to zero.

SP
The initial stack pointer for the new process. The stack pointer is relative to the initial Stack Segment (refer to SS below). The initial stack of the new process must be initialized with the offset of the first instruction to be executed by the new process. The word that the stack pointer points to is the initial instruction pointer. Two words must follow the initial IP, which is filled in with the initial Code Segment (refer to CS below) and the initial flags. The initial flags are set to 0200h, which means that interrupts are ON, and all other flags are OFF. Concurrent CP/M starts a new process by executing an Interrupt Return instruction with the initial stack.

Note: This stack area is distinct from the User System Stack at the end of the UDA.

```
Stack Initialization Area
-------------------------
    Low Memory
    ...
    stack area
    ...
SS SP IP
    0  (CS)
    0  (Flags)
```

INT 0, INT 1, INT 3, INT 4
The initial interrupt vectors for the first five interrupts types can be set by filling in these fields. The first word of each field is the Instruction Pointer (IP), and the second word is the Code Segment (CS) for a list of the interrupt routine that services these interrupts. Those fields that are zero are initialized to be the same as the calling processes interrupt vectors. These fields are typically initialized to be 0.

CS, DS, ES, SS
The initial segment addresses for the new process are taken from these fields. Those fields that are zero are initialized to be the same as the calling process' Data Segment.

INT 224, INT 225
Interrupts 224 and 225 are used to communicate with Concurrent CP/M by typical

programs. These interrupt vectors are initialized to be the same as the
calling process if these values are zero. The ability to change these values
allows a run-time system to intercept Concurrent CP/M calls that its children
make. The suggested protocol is to keep INT 225 pointing to the Concurrent
CP/M entry point, and changing INT 224 to point to an internal routine. When a
child process does an INT 224, the internal routine can filter calls to
Concurrent CP/M, using INT 225 for the actual Concurrent CP/M call.

RESERVED
All reserved fields are used internally, and must be initialized to zero.

USER SYSTEM STACK
This is the stack area used by the process when it is in the operating system.
The SP variable in the UDA should not point to this area.

CW (*)
Control Word for 8087 processor. Processes bypassing the P_CLI or P_LOAD
system call must set this word to 03FFh.

SW (*)
Status Word for 8087 processor. Processes bypassing the P_CLI or P_LOAD system
call must set this word to 0000h.

(*) = Part of optional 8087 Extension. If the 8087 flag is set in the SFLAG
field, this 6-paragraph extension must be included for the 8087 environment.

P_DELAY

Suspend the calling process for a specified number of system clock ticks.

```
Entry Parameters:
    Register CL: 141  (8Dh)
           DX: Number of System Ticks
```

The P_DELAY system call causes the calling process to wait until the specified
number of system ticks has occurred. The P_DELAY system call avoids the
necessity of programmed delay loops. It allows other processes to use the CPU
resource, while the calling process waits.

The length of the system teck varies among installations. A typical system
tick is 60 Hz (16.67 milliseconds) in the USA. In Europe, it is likely to be
50 Hz (20 milliseconds). The exact length of the system tick can be obtained
by reading the TICKS/SEC value from the System Data Segment (refer to the
S_SYSDAT system call).

There is up to one tick of uncertainty in the exact amount of time delayed.
This is due to the P_DELAY system call being called asynchronously from the
actual time base. The P_DELAY system call is guaranteed to delay the calling
proces at least the number of ticks specified. However, when the calling
process is rescheduled to run, it might wait quite a bit longer if there are
higher priority processes waiting to run. The P_DELAY system call is used
primarily by programs that need to wait specific amounts of time for I/O

events to occur. Under these conditions, the calling process usually has a very high priority level. If a process with a high priority calls the P_DELAY system call, the actual delay is typically within a system tick of the amount of time wanted.

P_DISPATCH

Force a dispatch operation; give up the CPU resource to the highest priority process ready to run.

```
Entry Parameters:
    Register CL: 142  (8Eh)
```

The P_DISPATCH system call forces a reschedule of processes that are waiting to run. Normally, dispatches occur at every system tick interrupt (usually 60 times a second), and whenever a process releases a system resource. Dispatching also occurs whenever a process needs a system resource that is not currently available. A CPU-bound process runs for no more than one system tick before a dispatch is forced. The dispatch occurs at the next system tick.

The Concurrent CP/M Dispatcher is priority driven, with round-robin scheduling of equivalent-priority processes. When a process calls the P_DISPATCH system call, it is rescheduled, so that processes with higher or equivalent priorities are given the CPU before the calling process obtains it again. The calling process regains control of the CPU resource when it becomes the highest priority process again.

P_LOAD

Load the specified CMD file in memory; return its Base Page segment address.

```
Entry Parameters:
    Register CL: 59  (3Bh)
             DX: FCB Address -- Offset
             DS: FCB Address -- Segment

Returned  Values:
    Register AX: Base Page Address  (0FFFFh on Error)
             BX: Same as AX
             CX: Error Code
```

The P_LOAD system call loads a disk CMD-type file into memory. Upon entry, register DX contains the offset, relative to DS, of a successfully opened FCB that specifies the CMD file to load. Upon return, register AX has the value 0FFFFh if the program load failed. Otherwise, AX contains the paragraph address of the Base Page belonging to the loaded program. The paragraph address and length of each group loaded from the CMD file is found in the Base Page. See Sections 3.2, "Command file format", and 3.3, "Base Page initialization".

Note that, before calling P_LOAD, the calling process must establish the DMA address of where the CMD file to be loaded. This is accomplished with F_DMASEG and F_DMAOFF.

Note: Open the CMD file in Read-Only mode, and close it once the load is completed.

Refer to Table 6-5 for a list of error codes returned in CX.

P_PDADR

Return the address of the Process Descriptor of the calling process.

```
Entry Parameters:
    Register CL: 156  (9Ch)

Returned  Values:
    Register AX: PD Address -- Offset
             BX: Same as AX
             ES: PD Address -- Segment
```

The P_PDADR system call obtains the address of the calling process' Process Descriptor. For a description of the format of the Process Descriptor, refer to the P_CREATE system call.

P_PRIORITY

Set the priority of the calling process.

```
Entry Parameters:
    Register CL: 145  (91h)
             DL: Priority
```

The P_PRIORITY system call sets the priority of the calling process to the specified value. This system call is useful in situations where a process needs to have a high priority during an initialization phase, but afterwards can run at a lower priority.

The best or highest priority is 00h, while the worst or lowest priority is 0FFh. Transient processes are initialized to run at 0C8h (200) by the P_CLI system call.

P_RPL

Invoke a system call from a Resident Procedure Library.

```
Entry Parameters:
    Register CL: 151  (97h)
```

```
          DX: CPB Address -- Offset
          DS: CPB Address -- Segment

Returned  Values:
    Register AX: 0001h if RPL not found,
                 RPL return parameter
          BX: Same as AX
          CX: Error Code
          ES: RPL return segment if address

      +------+------+------+------+------+------+------+------+
      |                    NAME                    |
      +------+------+------+------+------+------+------+------+
      |   PARAM   |
      +------+------+
```

Figure 6-14. CPB -- Call Parameter Block

Table 6-19. CPB field definitions

Format: Field
Definition

NAME
Name of Resident Procedure, eight ASCII characters.

PARAM
Parameter to send to the Resident Procedure.

P_RPL permits a process to call a system call in an optional Resident Procedure Library (RPL).

P_RPL opens a system queue with the specified name. If the Q_OPEN system call succeeds, P_RPL checks the queue to verify that it is an RPL-type queue. If either the Q_OPEN fails, or if it is not an RPL-type queue, P_RPL returns to the calling process with an error code.

P_RPL reads a message from the queue that contains the address of the specified system call. It then places the PARAM field of the CPB in register DX, and places the calling process' Data Segment address in register DS. P_RPL performs a Far Call instruction to the address it obtains from the queue message. Upon return from the RPL, the system call copies the BX register to the AX register, and then returns to the calling process.

Note: The P_RPL system call does not write the address of the Resident Procedure back to the queue. The Resident Procedure itself must do this. If the Resident Procedure is to be re-entrant, it must write the message into the queue upon entry. If it is to be serially re-usable, the procedure must write the message just before returning.

Refer to Table 6-5 for a list of error codes returned in CX.

P_TERM

Terminate the calling process.

```
Entry Parameters:
    Register CL: 143  (8Fh)
             DL: Termination Code

Returned  Values:
    Register AX: 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The P_TERM system call terminates the calling process. If the termination code is not 0FFh, the system call can only terminate a user process. If the termination code is 0FFh, the system call can terminate the calling process, even though the process' System flag is ON. P_TERM cannot terminate a process with the Keep flag ON. If the termination is successful, the system call releases the mutual exclusion queues owned by the process. It also releases all memory segments owned by the process, and returns the Process Descriptor to the PD table.

A process can own one or more of the following resources: memory segments, consoles, printers, mutual exclusion messages, and system Lock List entries (that record open files and locked records). When a process terminates and releases its resources, these resources become available to other processes on the system. For example, if a terminating process releases a system console, the console is usually given back to the console's TMP. This occurs when the TMP is the highest priority process waiting for the console.

If the system call returns to the calling process, the P_TERM call has failed for one of two reasons. Either the process has the Keep flag ON, or it has the System flag ON, and the termination code is not 0FFh.

P_TERMCPM

Terminate calling process unconditionally, release all owned resources.

```
Entry Parameters:
    Register CL: 0  (00h)

Returned  Values:
    Register AX: 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The P_TERMCPM system call terminates the calling process, releasing all system resources owned by the process.

P_TERMCPM is implemented internally by calling P_TERM with the termination code set to 00h.

Under CP/M-86, the P_TERMCPM system call has a further argument that allows a process not to release its memory. This argument places a piece of code into memory that becomes an interface for later programs. Concurrent CP/M does not include this option. Memory segments are not recovered by the system until all processes that own the memory segment have released it.

Refer to Table 6-5 for a list of error codes returned in CX.

6.2.8 Queue system calls

Queue system calls under Concurrent CP/M use the Queue Parameter Block data structure to pass parameters to and from the operating system. Listing 6-3 shows the structure of the Queue Parameter Block and the equates for its fields.

```
        +------+------+------+------+------+------+------+------+
        |   * 0000h   |   QUEUEID   |   * 0000h   |   BUFFER    |
        +------+------+------+------+------+------+------+------+
        |                      NAME                      |
        +------+------+------+------+------+------+------+------+
```

Figure 6-15. QPB -- Queue Parameter Block

Table 6-20. QPB field definitions

Format: Field
Definition

Field	Definition
* 0000h	Reserved for internal use; must be initialized to zero.
QUEUEID	Queue number field; filled in by a Q_OPEN operation.
BUFFER	Offset address of Queue Message Buffer.
NAME	Name of Queue for Q_OPEN operation.

```
;*****************************************************************
;*
;*      QPB -- Queue Parameter Block definition
;*
;*          +----+----+----+----+----+----+----+----+
;*      00h |  0000h  | QUEUEID |  0000h  | BUFFER  |
;*          +----+----+----+----+----+----+----+----+
;*      08h |                NAME                   |
;*          +----+----+----+----+----+----+----+----+
;*
```

```
;*      QUEUEID -- Queue ID, address of QD
;*      BUFFER  -- Address to read/write into/from
;*      NAME    -- Name of queue (for open only)
;*
;*************************************************************

qpb_0           equ     word ptr 0
qpb_queueid     equ     word ptr qpb_0 + word
qpb_buffer      equ     word ptr qpb_queueid + 4
qpb_name        equ     byte ptr qpb_buffer + word

qpb_len         equ     qpb_name + qnamsiz
qnamsiz         equ     8
```

Listing 6-3. Queue Parameter Block definition

```
Q_CREAD
-------
```

Conditionally read a message from a system queue; return error code if a message is not available.

```
Entry Parameters:
    Register CL: 138  (8Ah)
            DX: QPB Address -- Offset
            DS: QPB Address -- Segment
    QPB_QueueID filled in by previous Q_OPEN
    QPB_Buffer set to message buffer offset

Returned  Values:
    Register AX:  0000h on Success,
               0FFFFh on Failure
            BX: Same as AX
            CX: Error Code
    Message in buffer
```

The Q_CREAD system call is analogous to the Q_READ system call, but it returns an error code if there are not enough messages to read, instead of waiting for another process to write to the queue.

Refer to Table 6-5 for a list of error codes returned in CX.

```
Q_CWRITE
--------
```

Conditionally write a message to a system queue; return an error code if space is not available.

```
Entry Parameters:
    Register CL: 140  (8Ch)
            DX: QPB Address -- Offset
            DS: QPB Address -- Segment
    QPB_QueueID filled in by previous Q_OPEN
    QPB_Buffer set to message buffer offset
    Message in current DMA Buffer

Returned  Values:
    Register AX:  0000h on Success,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code
```

The Q_CWRITE system call is analogous to the Q_WRITE system call, but it returns an error code if there is not enough system queue buffer space for the message to be written, instead of waiting for another process to read from the queue.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_DELETE

Delete a system queue.

```
Entry Parameters:
    Register CL: 136  (88h)
           DX: QPB Address -- Offset
           DS: QPB Address -- Segment
    QPB_QueueID filled in by a previous Q_OPEN call

Returned  Values:
    Register AX:  0000h on Success,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code
```

The Q_DELETE system call removes a system queue from the system. The system returns error codes if the queue cannot be deleted, or if the queue has not been opened prior to the Q_DELETE call.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_MAKE

Create a system queue.

```
Entry Parameters:
    Register CL: 134  (86h)
           DX: QD Address -- Offset
           DS: QD Address -- Segment
    QD filled in

Returned  Values:
    Register AX:  0000h on Success,
                 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code

      +------+------+------+------+------+------+------+------+
      |  * 0000h  |  * 0000h  |    FLAGS    |  NAME...
      +------+------+------+------+------+------+------+------+
                        ...NAME                |  MSGLEN   |
      +------+------+------+------+------+------+------+------+
      |   NMSGS    |  * 0000h  |  * 0000h  |  * 0000h  |
      +------+------+------+------+------+------+------+------+
      |  * 0000h  |  BUFFER    |
      +------+------+------+------+
```

Figure 6-16. QD -- Queue Descriptor

Table 6-21. Queue Descriptor field definition

Format: Field
Definition

* 0000h
For internal use. Must be initialized to zero.

FLAGS
Queue Flags. The bits are defined as follows:

```
      0001h = Mutual exclusion queue
      0002h = Cannot be deleted
      0004h = Restricted to system processes
      0008h = RSP message queue
      0010h = Used internally
      0020h = RPL address queue
      0040h = Used internally
      0080h = Used internally
```

Remaining flags reserved for future use.

NAME
8-byte queue name. All 8 bits of each character are matched on a Q_OPEN call.

MSGLEN
Number of bytes in each logical message.

NMSGS
Maximum number of logical messages to be supported. If the number of messages written to the queue equals this maximum, no more messages are allowed until a message is read.

BUFFER
Address of the queue buffer. This buffer must be (NMSGS * MSGLEN) bytes long. The address is an offset relative to the DS register. This field is unused if

the QD resides outside of the System Data Segment. Typically, this field is 0 if the queue is being created by a transient program. RSPs that creates queues must initialize this field to poin to a buffer. The Data Segment of an RSP's queue is considered part of the System Data Segment, unless it is beyond 64 KB of the beginning of the System Data Segment.

Every system queue under Concurrent CP/M is associated with a Queue Descriptor that resides within the Concurrent CP/M System Data Segment. In the Q_MAKE system call, the calling process passes the address of a Queue Descriptor. If this Queue Descriptor is within the Concurrent CP/M System Data Segment, the system uses it directly for the System Queue. If the Queue Descriptor is outside of the System Data Segment, the system obtains a Queue Descriptor from an internal Queue Descriptor table. If there are no unused Queue Descriptors in the internal table, the system call returns an error code.

Refer to Table 6-5 for a list of error codes returned in CX.

The buffer for a system queue must also reside within the System Data Area. For non-zero length buffers, resident buffers are used directly. The system obtains a buffer from the Queue Buffer Area if the buffer does not reside within the System Data Segment. The size of the buffer is calculated from the NMSGS and MSGLEN fields. The system call returns an error code if there is not enough unused buffer area left to accomodate this new buffer.

All system queues must have unique names. The system call returns an error code if a system queue already exists by the given name.

Under Concurrent CP/M, all system queues must be explicitly opened (refer to the Q_OPEN system call) before being used to read or write messages, or to delete the queue.

Q_OPEN

Open a system queue for subsequent queue operations.

```
Entry Parameters:
    Register CL: 135  (87h)
           DX: QPB Address -- Offset
           DS: QPB Address -- Segment
    QPB_Name filled in

Returned  Values:
    Register AX:  0000h on Success,
              0FFFFh on Failure
           BX: Same as AX
           CX: Error Code
    QPB_QueueID filled in
```

All system queues under Concurrent CP/M must be explicitly opened before a read, write, or delete operation can be done. The Q_OPEN system call examines each existing system queue, and attempts to match the name in the QPB with the name of a system queue. All eight bytes of the name must match for a successful open. All bits of each byte are examined. If the open operation is successful, the Q_OPEN system call modifies the Queue ID field of the QPB. Once the queue is opened, subsequent reads, writes, or a delete are allowed.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_READ

Read a message from a system queue; suspend calling process until message is available.

```
Entry Parameters:
    Register CL: 137  (89h)
             DX: QPB Address -- Offset
             DS: QPB Address -- Segment
    QPB_QueueID filled in by previous Q_OPEN
    QPB_Buffer set to message buffer offset

Returned  Values:
    Register AX:  0000h on Success,
                 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The Q_READ system call reads a message from a system queue that was previously opened by the calling process. The system call returns an error code if the queue was not previously opened, ot if the system queue has been deleted since the Q_OPEN call. If there are not enough messages to read from the queue, the calling process waits until another process writes into the queue before returning.

Refer to Table 6-5 for a list of error codes returned in CX.

Q_WRITE

Write a message to a system queue; suspend calling process until space becomes available.

```
Entry Parameters:
    Register CL: 139  (8Bh)
             DX: QPB Address -- Offset
             DS: QPB Address -- Segment
    QDPB_QueueID filled in by previous Q_OPEN
    QPB_Buffer set to message buffer offset

Returned  Values:
    Register AX:  0000h on Success,
                 0FFFFh on Failure
             BX: Same as AX
             CX: Error Code
```

The Q_WRITE system call writes a message to a system queue that was previously opened by the calling process. The system call returns an error code if the queue was not previously opened, or if the system queue has been deleted since the Q_OPEN call. If there is not enough buffer space in the queue, the calling process waits until another process reads from the queue before writing to the queue and returning.

Refer to Table 6-5 for a list of error codes returned in CX.

6.2.9 System information system calls

S_BDOSVER

Return BDOS version number, CPU and operating system type.

```
Entry Parameters:
    Register CL: 12  (0Ch)

Returned  Values:
    Register AL: 31h (BDOS Version 3.1)
             AH: 14h (Concurrent CP/M)
             BX: Same as AX
```

The S_BDOSVER system call returns the BDOS file system version number, allowing version-independent programming.

```
      AL High Nibble = BDOS Version Number
      AL Low  Nibble = BDOS Revision Level
      AH High Nibble = CPU Type : 0 = 8080, 1 = 8086
      AH Low  Nibble = OS  Type : 0 = CP/M
                                  1 = MP/M
                                  2 = CP/M with networking
                                  3 = MP/M with networking
                                  4 = Concurrent CP/M
                                  5 = Reserved
                                  6 = Concurrent CP/M with networking
                                  7 to 0Eh = Reserved
```

Figure 6-17. BDOS Version Number format

S_BIOS

Call specified CP/M-86 BIOS Character I/O routine.

```
Entry Parameters:
    Register CL: 50  (32h)
             DX: BIOS Descriptor Address -- Offset
             DS: BIOS Descriptor Address -- Segment
```

Returned Values:
    Register AX: BIOS Return
           BX: Same as AX

```
      +------+------+------+------+------+
      | FUNC |     CX      |     DX      |
      +------+------+------+------+------+
```

      Figure 6-18. BIOS Descriptor format

The S_BIOS system call is provided under Concurrent CP/M for compatibility with programs generated under CP/M-86 that use this system call (Function 50). Under Concurrent CP/M, only routines that interface with character devices are supported. The arguments to character routines such as CONIN and LIST must be converted to those appropriate for the Concurrent CP/M XIOS. Refer to the "Concurrent CP/M System Guide" for further information about the XIOS.

Note: Calls to the XIOS Console Status, Input, and Output system calls do not go to the XIOS if the referenced device is a virtual console.

S_OSVER

Return type and version number of Concurrent CP/M.

Entry Parameters:
    Register CL: 163 (0A3h)

Returned Values:
    Register AX: Version Number (1431h)
           BX: Same as AX
           CX: Error Code

The S_OSVER system call provides information that allows version_independent programming. The system call returns a two-byte value, with AH set to 14h for Concurrent CP/M, and AL set to the Concurrent CP/M version level. The AH register contains a value set to the type of operating system. A value of 1431h indicates Concurrent CP/M 3.1.

Refer to Table 6-5 for a list of error codes returned in CX.

```
      AL High Nibble = Concurrent CP/M Version Number
      AL Low  Nibble = Concurrent CP/M Revision Level
      AH High Nibble = CPU Type : 0 = 8080, 1 = 8086
      AH Low  Nibble = OS  Type : 0 = CP/M
                                  1 = MP/M
                                  2 = CP/M with networking
                                  3 = MP/M with networking
                                  4 = Concurrent CP/M
                                  5 = Reserved
                                  6 = Concurrent CP/M with networking
                                  7 to 0Eh = Reserved
```

Figure 6-19. Operating System Version Number format

S_SERIAL

Return the Concurrent CP/M system serial number.

Entry Parameters:
    Register CL: 107 (6Bh)
             DX: Serial Address -- Offset
             DS: Serial Address -- Segment

Returned Values:
    Serial Number filled in

```
      +---+---+---+---+---+---+
      | 0 | 1 | 2 | 3 | 4 | 5 |
      +---+---+---+---+---+---+
```

Figure 6-20. Serial Number format

S_SERIAL returns the Concurrent CP/M serial number to the addressed, 6-byte Serial field as a 6-byte ASCII numeral.

S_SYSDAT

Return address of the System Data Segment (SYSDAT).

Entry Parameters:
    Register CL: 154 (9Ah)

Returned Values:
    Register AX: Sysdat Address -- Offset
             BX: Same as AX
             ES: Sysdat Address -- Segment

The S_SYSDAT system call returns the address of the System Data Segment of the calling process. The System Data Segment contains all Process Descriptors, Queue Descriptors, the roots of system lists, and other internal data that Concurrent CP/M uses.

Figure 6-21 illustrates the SYSDAT DATA and its fields.

```
        +-----+-----+-----+-----+-----+-----+-----+-----+
    00h |       SUP ENTRY       |       RESERVED        |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    08h |                   RESERVED                    |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    10h |                   RESERVED                    |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    18h |                RESERVED               |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    20h |                RESERVED               |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    28h |     XIOS ENTRY      |      XIOS INIT      |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    30h |                RESERVED               |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    38h |      DISPATCHER     |        PDISP        |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    40h |  CCPMSEG  |  RSPSEG   |  ENDSEG   |RESER|NVCNS|
        |           |           |           | -VED|     |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    48h |NLCB |NCCB |  N_ | SYS_|    MMP    |RESER| DAY |
        |     |     |FLAGS| DISK|           | -VED| FILE|
        +-----+-----+-----+-----+-----+-----+-----+-----+
    50h | TEMP|TICKS|    LUL    |    CCB    |   FLAGS   |
        | DISK| /SEC|           |           |           |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    58h |   MDUL    |    MFL    |    PUL    |    QUL    |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    60h |           |        QMAU           |           |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    68h |    RLR    |    DLR    |    DRL    |    PLR    |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    70h |  RESERVED |  THRDRT   |    QLR    |    MAL    |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    78h |  VERSION  |  VERNUM   |CCPMVERNUM |  TOD_DAY  |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    80h | TOD | TOD | TOD |NCON |NLST |NCIO |    LCB    |
        | _HR | _MIN| _SEC| DEV | DEV | DEV |           |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    88h | OPEN_FILE |LOCK_|OPEN_|OWNER_8087 |  RESERVED |
        |           | MAX | MAX |           |           |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    90h |                RESERVED               |
        +-----+-----+-----+-----+-----+-----+-----+-----+
    98h |               RESERVED              |XPCNS|
        +-----+-----+-----+-----+-----+-----+-----+-----+
    A0h | OFF_8087  | SEG_8087  | SYS_87_OF | SYS_87_SG |
        +-----+-----+-----+-----+-----+-----+-----+-----+
```

Figure 6-21. SYSDAT DATA

Table 6-22. SYSDAT DATA field definitions

Format: Field
Explanation

SUP ENTRY
Double-word address of the Supervisor entry point for intermodule communication. All internal system calls go through this entry point.

XIOS ENTRY
Double-word address of the Extended I/O System entry point for intermodule communication. All XIOS function calls go through this entry point.

XIOS INIT
Double-word address of the Extended I/O System initialization entry point. System hardware initialization takes place by a call through this entry point.

DISPATCHER
Double-word address of the Dispatcher entry point that handles interrupt returns. Executing a JMPF instruction to this address is equivalent to executing an Interrupt RETurn instruction. The Dispatcher routine causes a dispatch to occur, and then executes an Interrupt Return. All registers are preserved, and one level of stack is used. This location should be used as an exit point by all XIOS interrupt handlers that use the DEV_SETFLAG system call.

PDISP
Double-word address of the Dispatcher entry point that causes a dispatch to occur with all registers preserved. Once the dispatch is done, a RETF instruction is executed. Executing a JMPF PDISP is equivalent to executing a RETF instruction. This location should be used as an exit point whenever the XIOS releases a resource that might be wanted by a waiting process.

CCPMSEG
Starting paragraph of the operating system area. This is also the Code Segment of the Supervisor Module.

RSPSEG
Paragraph Address of the first RSP in a linked list of RSP Data Segments. The first word of the data segment points to the next RSP in the list. Once the system has been initialized, this field is zero.

ENDSEG
First paragraph beyond the end of the operating system area, including any buffers consisting of un-initialized RAM allocated to the operating system by GENCCPM. These include the Directory Hashing, Disk Data, and XIOS ALLOC buffers. These buffers areas, however, are not part of the CCPM.SYS file.

NVCNS
Number of Virtual CoNSoles, copied from the XIOS Header by GENCCPM.

NLCB
Number of List Control Blocks, copied from the XIOS Header by GENCCPM.

NCCB
Number of Character Control Blocks, copied from the XIOS Header by GENCCPM.

NFLAGS
Number of system flags, as specified by GENCCPM.

SYSDISK
Default system disk. The CLI (Command Line Interpreter) looks on this disk if it cannot open the command file on the user's current default disk. Set during

GENCCPM.

MMP
Maximum Memory allocated per Process. Set during GENCCPM.

DAY FILE
Day File option. If this field is 0FFh, the operating system displays date and time information when an RSP or CMD file is invoked. Set during GENCCPM.

TEMP DISK
Default temporary disk. Programs that create temporary files should use this disk. Set during GENCCPM.

TICKS/SEC
The number of system ticks per second.

LUL
Locked Unused List. Link list root of unused Lock list items.

CCB
Address of the Character Control Block Table, copied from the XIOS Header by GENCCPM.

FLAGS
Address of the Flag Table.

MDUL
Memory Descriptor Unused List. Link list root of unused Memory Descriptors.

MFL
Memory Free List. Link list root of free memory partitions.

PUL
Process Unused List. Link list root of unused Process Descriptors.

QUL
Queue Unused List. Link list root of unused Queue Descriptors.

QMAU
Queue buffer Memory Allocation Unit.

RLR
Ready List Root. Linked list of PDs that are ready to run.

DLR
Delay List Root. Linked list of PDs that are delaying for a specified number of system ticks.

DRL
Dispatcher Ready List. Temporary holding place for PDs that have just been made ready to run.

PLR
Poll List Root. Linked list of PDs that are polling on devices.

THRDRT
THReaD list RooT. Linked list of all current PDs on the system. The list is threaded through the THREAD field of the PD, instead of the LINK field.

QLR
Queue List Root. Linked list of all System QDs.

MAL
Memory Allocation List. Link list of active memory allocation units. A MAU is created from one or more memory partitions.

VERSION
Address, relative to CCPMSEG, of ASCII version string.

VERNUM
BDOS version number (returned by the S_BDOSVER system call).

CCPMVERNUM
Concurrent CP/M version number (system call 163, S_OSVER).

TOD_DAY
Time Of Day. Number of days since 1 Jan, 1978.

TOD_HR
Time Of Day. Hour of the day.

TOD_MIN
Time Of Day. Minute of the hour.

TOD_SEC
Time Of Day. Second of the minute.

NCONDEV
Number of XIOS CONsole DEVices, copied from the XIOS Header by GENCCPM.

NLSTDEV
Number of XIOS LiST DEVices, copied from the XIOS Header by GENCCPM.

NCIODEV
Total Number of Character I/O DEVices (NCONDEV + NLSTDEV).

LCB
Offset of the List Control Block Table, copied from the XIOS Header by GENCCPM.

OPEN_FILE
Open File Drive Vector. Designates drives that have open files on them. Each bit of the word value represents a disk drive; the least significant bit represents Drive A, and so on through the most significant bit, Drive P. Bits which are set indicate drives containing open files.

LOCK_MAX
Maximum number of locked records per process. Set during GENCCPM.

OPEN_MAX
Maximum number of open disk files per process. Set during GENCCPM.

OWNER_8087
Specifies 8087 information. If set to 0FFFFh, the system assumes that there is no 8087 in the system. If set to 0, there is an 8087 but no one owns it. If set to any other value, the system assumes that this value is the PD offset of the 8087 current process.

XPCNS
Number of Physical CoNSoles.

OFF_8087
OFFset of the 8087 interrupt vector in low memory.

SEG_8087
SEGment of the 8087 interrupt vector in low memory.

SYS_87_OF
OFfset of the default 8087 exception handler.

SYS_87_SG
SeGment of the default 8087 exception handler.

6.2.10 Time system calls

T_GET

Obtain the system calendar and clock (hours and minutes only).

Entry Parameters:
    Register CL: 105 (69h)
            DX: TOD Address -- Offset
            DS: TOD Address -- Segment

Returned Values:
    Register AL: Seconds
    TOD filled in (Days, Hours, and Minutes only)

```
      +------+------+------+------+------+
      |     DAY     | HOUR | MIN  | SEC  |
      +------+------+------+------+------+
```

Figure 6-22. TOD -- Time-Of-Day structure

Table 6-23. Time-Of-Day field definitions

Format: Field
      Definition

DAY
The number of days since 31 December 1977. The day is stored as a 16-bit integer.

HOUR
The current hour of the current day. The hour is represented as a 24 hour clock in 2 binary coded decimal (BCD) digits.

MIN
The current minute of the current hour. The minute is stored as 2 BCD digits.

SEC
The current second of the current minute. The second is stored as 2 BCD digits.

The T_GET system call obtains the system internal time and date. The calling process passes the address of a 4-byte data structure that receives the time and date values. This system call is equivalent to the T_SECONDS system call, except that it does not return the SECONDS field of the internal time.

T_SECONDS

Return current system date and time; hours, minutes, seconds.

```
Entry Parameters:
    Register CL: 155  (9Bh)
             DX: TOD Address -- Offset
             DS: TOD Address -- Segment

Returned  Values:
    TOD filled in (Days, Hours, Minutes, and Seconds)
```

The T_SECONDS system call returns the current encoded time and date (including seconds) in the TOD structure passed by the calling process.

T_SET

Set internal system calendar and clock to specified value.

```
Entry Parameters:
    Register CL: 104  (68h)
             DX: TOD Address -- Offset
             DS: TOD Address -- Segment
```

The T_SET system call sets the system internal time and date. The calling process passes the address of a 4-byte structure containing the time and date specification.

The date is represented as a 16-bit integer, with day 1 corresponding to

January 1, 1978. The time is represented as 2 bytes hours and minutes stored as 2 BCD digits.

Under Concurrent CP/M, this system call also sets the second field of the system time and date to 00h.

EOF

CCPMPRGA.WS4 (Concurrent CP/M Programmer's Reference Guide, Appendix A)

(Retyped by Emmanuel ROCHE.)

Appendix A: System call summary by function number

This appendix lists the Concurrent CP/M system calls by function number including the parameters a process must pass when calling the function, and the values that the function returns to the process.

Table A-1. System call summary by function number

Dec	Mnemonic	Input parameters	Returned values
0	P_TERMCPM	none	AX = Rtn Code
1	C_READ	none	AL = char
2	C_WRITE	DL = char	none
5	L_WRITE	DL = char	none
6	C_RAWIO	see def	see def
9	C_WRITESTR	DX = .Buffer	none
10	C_READSTR	DX = .Buffer	see def
11	C_STAT	none	AL = 1 if ready, 0 if not
12	S_BDOSVER	none	AX = Version #
13	DRV_ALLRESET	none	see def
14	DRV_SET	AL = Drive #	see def
15	F_OPEN	DX = .FCB	AL = Dir Code
16	F_CLOSE	DX = .FCB	AL = Dir Code
17	F_SFIRST	DX = .FCB	AL = Dir Code
18	F_SNEXT	none	AL = Dir Code
19	F_DELETE	DX = .FCB	AL = Dir Code
20	F_READ	DX = .FCB	AL = Err Code
21	F_WRITE	DX = .FCB	AL = Err Code
22	F_MAKE	DX = .FCB	AL = Dir Code
23	F_RENAME	DX = .FCB	AL = Dir Code
24	DRV_LOGINVEC	none	AX = Login Vector
25	DRV_GET	none	AL = Cur Drive
26	F_DMAOFF	DX = .DMA	none
27	DRV_ALLOCVEC	none	ES:AX = Alloc Addr
28	DRV_SETRO	none	see def
29	DRV_ROVEC	none	AX = R/O Vector
30	F_ATTRIB	DX = .FCB	see def
31	DRV_DPB	none	ES:AX = DPB Addr
32	F_USERNUM	DL = 0FFh (Get)	AL = User #
		= User # (Set)	none
33	F_READRAND	DX = .FCB	AL = Err Code
34	F_WRITERAND	DX = .FCB	AL = Err Code
35	F_SIZE	DX = .FCB	R0,R1,R2 & AL = Dir Code
36	F_RANDREC	DX = .FCB	R0,R1,R2

37	DRV_RESET	DX = Drive Vector	AL = Err Code
38	DRV_ACCESS	DX = Drive Vector	none
39	DRV_FREE	DX = Drive Vector	none
40	F_WRITEZF	DX = .FCB	AL = Err Code
42	F_LOCK	DX = .FCB	AL = Err Code
43	F_UNLOCK	DX = .FCB	AL = Err Code
44	F_MULTISEC	DL = # of Records	AL = Rtn Code
45	F_ERRMODE	DL = Error Mode	none
46	DRV_SPACE	DL = Drive #	see def
47	P_CHAIN	see def	none
48	DRV_FLUSH	none	see def
50	S_BIOS	DX = .BD	AX = BIOS Rtn
51	F_DMASEG	DX = .DMA Seg	none
52	F_DMAGET	none	ES:AX = DMA Addr
53	MC_MAX	DX = .MCB	see def
54	MC_ABSMAX	DX = .MCB	see def
55	MC_ALLOC	DX = .MCB	see def
56	MC_ABSALLOC	DX = .MCB	see def
57	MC_FREE	DX = .MCB	see def
58	MC_ALLFREE	none	none
59	P_LOAD	DX = .FCB	AX = BP Addr
99	F_TRUNCATE	DX = .FCB	see def
100	DRV_SETLABEL	DX = .FCB	AL = Dir Code
101	DRV_GETLABEL	DX = Drive #	AL = Label Data Byte
102	F_TIMEDATE	DX = .XFCB	AL = Dir Code
103	F_WRITEXFCB	DX = .XFCB	AL = Dir Code
104	T_SET	DX = .TOD	none
105	T_GET	DX = .TOD	AL = seconds
106	F_PASSWD	DX = .Password	none
107	S_SERIAL	DX = .serial #	serial #
109	C_MODE	DX = Con Mode	none
		= 0FFFFh	AX = Con Mode
110	C_DELIMIT	DL = Out Delim	none
		= 0FFFFh	AL = Out Delim
111	C_WRITEBLK	DX = .CHCB	none
112	L_WRITEBLK	DX = .CHCB	none
128	M_ALLOC	DX = .MPB	AX = Rtn Code
129	M_ALLOC	Same as above	Same as above
130	M_FREE	DX = .MPB	none
131	DEV_POLL	DL = Device	none
132	DEV_WAITFLAG	DL = Flag	AX = Rtn Code
133	DEV_SETFLAG	DL = Flag	AX = Rtn Code
134	Q_MAKE	DX = .QD	none
135	Q_OPEN	DX = .QPB	AX = Rtn Code
136	Q_DELETE	DX = .QPB	AX = Rtn Code
137	Q_READ	DX = .QPB	none
138	Q_CREAD	DX = .QPB	AX = Rtn Code
139	Q_WRITE	DX = .QPB	
140	Q_CWRITE	DX = .QPB	AX = Rtn Code

141	P_DELAY	DX = # ticks	none
142	P_DISPATCH	none	none
143	P_TERM	DL = Term Code	AX = Rtn Code
144	P_CREATE	DX = .PD	none
145	P_PRIORITY	DL = Priority	none
146	C_ATTACH	none	none
147	C_DETACH	none	none
148	C_SET	DL = console #	none
149	C_ASSIGN	DX = .ACB	AX = Rtn Code
150	P_CLI	DX = .CLBUF	none
151	P_RPL	DX = .CPB	AX = result
152	F_PARSE	DX = .PFCB	see def
153	C_GET	none	AL = console #
154	S_SYSDAT	none	ES:AX = Sys Data Addr
155	T_SECONDS	DX = .TOD	TOD filled in
156	P_PDADR	none	ES:AX = PD Addr
157	P_ABORT	DX = .ABP	AX = Rtn Code
158	L_ATTACH	none	none
159	L_DETACH	none	none
160	L_SET	DL = List #	none
161	L_CATTACH	none	AX = Rtn Code
162	C_CATTACH	none	AX = Rtn Code
163	S_OSVER	none	AX = Version #
164	L_GET	none	AL = List #

Conventions used in Appendix A:

#	= Number
ACB	= Assigned Control Block
APB	= Abort Parameter Block
Addr	= Address
BD	= BIOS Descriptor
BP	= Base Page
Char	= ASCII Character
CHCB	= CHaracter Control Block
CLBUF	= Command Line BUFfer
CPB	= Call Parameter Block
Con	= Console
Cur	= Current
Delim	= Delimiter
Dir	= Directory
DMA	= Direct Memory Address
Err	= Error
FCB	= File Control Block
MCB	= Memory Control Block
MPB	= Memory Parameter Block
Num	= Number
Out	= Output
PD	= Process Descriptor
PFCB	= Parse Filename Control Block
QD	= Queue Descriptor
QPB	= Queue Parameter Block
Rec	= Record

Rtn	= Return
Sys	= System
Term	= Termination
TOD	= Time Of Day
Vect	= Vector

EOF

CCPMPRGB.WS4   (Concurrent CP/M Programmer's Reference Guide, Appendix B)

(Retyped by Emmanuel ROCHE.)

Appendix B: ASCII and hexadecimal conversions

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 non-printing characters used to store data on a disk. Table B-1 defines ASCII symbols; Table B-2 lists the ASCII and hexadecimal conversions. Table B-2 includes binary, decimal, hexadecimal, and ASCII conversions.

Table B-1. ASCII Symbols

Symbol	Meaning	Symbol	Meaning
ACK	acknowledge	FS	file separator
BEL	bell	GS	group separator
BS	backspace	HT	horizontal tabulation
CAN	cancel	LF	line feed
CR	carriage return	NAK	negative acknowledge
DC	device control	NUL	null
DEL	delete	RS	record separator
DLE	data link escape	SI	shift in
EM	end of medium	SO	shift out
ENQ	enquiry	SOH	start of heading
EOT	end of transmission	SP	space
ESC	escape	STX	start of text
ETB	end of transmission	SUB	substitute
ETX	end of text	SYN	synchronous idle
FF	form-feed	US	unit separator
		VT	vertical tabulation

Table B-2. ASCII Conversion Table

Binary	Decimal	Hexa	ASCII
00000000	0	0	NUL (Ctrl-@)
00000001	1	1	SOH (Ctrl-A)
00000010	2	2	STX (Ctrl-B)
00000011	3	3	ETX (Ctrl-C)
00000100	4	4	EOT (Ctrl-D)
00000101	5	5	ENQ (Ctrl-E)
00000110	6	6	ACK (Ctrl-F)
00000111	7	7	BEL (Ctrl-G)
00001000	8	8	BS (Ctrl-H)
00001001	9	9	HT (Ctrl-I)
00001010	10	A	LF (Ctrl-J)

00001011	11	B	VT (Ctrl-K)
00001100	12	C	FF (Ctrl-L)
00001101	13	D	CR (Ctrl-M)
00001110	14	E	SO (Ctrl-N)
00001111	15	F	SI (Ctrl-O)
00010000	16	10	DLE (Ctrl-P)
00010001	17	11	DC1 (Ctrl-Q)
00010010	18	12	DC2 (Ctrl-R)
00010011	19	13	DC3 (Ctrl-S)
00010100	20	14	DC4 (Ctrl-T)
00010101	21	15	NAK (Ctrl-U)
00010110	22	16	SYN (Ctrl-V)
00010111	23	17	ETB (Ctrl-W)
00011000	24	18	CAN (Ctrl-X)
00011001	25	19	EM (Ctrl-Y)
00011010	26	1A	SUB (Ctrl-Z)
00011011	27	1B	ESC (Ctrl-[)
00011100	28	1C	FS (Ctrl-\\)
00011101	29	1D	GS (Ctrl-])
00011110	30	1E	RS (Ctrl-^)
00011111	31	1F	US (Ctrl-_)
00100000	32	20	(SPACE)
00100001	33	21	!
00100010	34	22	"
00100011	35	23	#
00100100	36	24	$
00100101	37	25	%
00100110	38	26	&
00100111	39	27	'
00101000	40	28	(
00101001	41	29	)
00101010	42	2A	*
00101011	43	2B	+
00101100	44	2C	,
00101101	45	2D	-
00101110	46	2E	.
00101111	47	2F	/
00110000	48	30	0
00110001	49	31	1
00110010	50	32	2
00110011	51	33	3
00110100	52	34	4
00110101	53	35	5
00110110	54	36	6
00110111	55	37	7
00111000	56	38	8
00111001	57	39	9
00111010	58	3A	:
00111011	59	3B	;
00111100	60	3C	<
00111101	61	3D	=
00111110	62	3E	>
00111111	63	3F	?
01000000	64	40	@

01000001	65	41	A
01000010	66	42	B
01000011	67	43	C
01000100	68	44	D
01000101	69	45	E
01000110	70	46	F
01000111	71	47	G
01001000	72	48	H
01001001	73	49	I
01001010	74	4A	J
01001011	75	4B	K
01001100	76	4C	L
01001101	77	4D	M
01001110	78	4E	N
01001111	79	4F	O
01010000	80	50	P
01010001	81	51	Q
01010010	82	52	R
01010011	83	53	S
01010100	84	54	T
01010101	85	55	U
01010110	86	56	V
01010111	87	57	W
01011000	88	58	X
01011001	89	59	Y
01011010	90	5A	Z
01011011	91	5B	[
01011100	92	5C	\
01011101	93	5D	]
01011110	94	5E	^
01011111	95	5F	_
01100000	96	60	'
01100001	97	61	a
01100010	98	62	b
01100011	99	63	c
01100100	100	64	d
01100101	101	65	e
01100110	102	66	f
01100111	103	67	g
01101000	104	68	h
01101001	105	69	i
01101010	106	6A	j
01101011	107	6B	k
01101100	108	6C	l
01101101	109	6D	m
01101110	110	6E	n
01101111	111	6F	o
01110000	112	70	p
01110001	113	71	q
01110010	114	72	r
01110011	115	73	s
01110100	116	74	t
01110101	117	75	u
01110110	118	76	v

01110111	119	77	w
01111000	120	78	x
01111001	121	79	y
01111010	122	7A	z
01111011	123	7B	{
01111100	124	7C	|
01111101	125	7D	}
01111110	126	7E	~
01111111	127	7F	DEL

EOF

CCPMPRGC.WS4   (Concurrent CP/M Programmer's Reference Guide, Appendix C)

(Retyped by Emmanuel ROCHE.)

Appendix C: Error codes

Table C-1. Concurrent CP/M error codes

Code #	Definition
0	No error
1	Function not implemented
2	Illegal function number
3	Can't find memory
4	Illegal system flag number
5	Flag overrun
6	Flag underrun
7	No unused queue desciptors left in QD table
8	No unused queue buffer area left
9	Can't find queue
10	Queue in use
12	No unused process descriptors left in process descriptor table
13	Queue access denied
14	Empty queue
15	Full queue
16	CLI queue missing
17	No 8087 in system
18	No unusued memory descriptors left in memory descriptor table
19	Illegal console number
20	Can't find process descriptor by name
21	Console does not match
22	No CLI process
23	Illegal disk number
24	Illegal file name
25	Illegal file type
26	Character not ready
27	Illegal memory descriptor
28	Bad load
29	Bad read
30	Bad open
31	Null command
32	Not owner
33	No code segment in load file
34	Active process descriptor
35	Can't terminate
36	Can't attach
37	Illegal list device number
38	Illegal paswword
40	External termination occurred

41	Fixup error upon load
42	Flag set ignored

EOF

CCPMPRGD.WS4   (Concurrent CP/M Programmer's Reference Guide, Appendix D)

(Retyped by Emmanuel ROCHE.)

Appendix D: ECHO.A86 listing

Listing D-1. ECHO.A86

```
; ECHO.A86
; --------
;
; Concurrent CP/M 3.1 -- ECHO.RSP
;
; Print command tail to console.
;
; Generation:
; A>asm86 echo
; A>gencmd echo
; A>ren ECHO.RSP=ECHO.CMD
;
;---------------------------------------
;
ccpmint         equ     224             ; CCP/M entry interrupt
;
c_writestr      equ     9               ; Print string
q_make          equ     134             ; Create queue
q_open          equ     135             ; Open queue
q_read          equ     137             ; Read queue
q_write         equ     139             ; Write queue
p_priority      equ     145             ; Set priority
c_detach        equ     147             ; Detach console
c_set           equ     148             ; Set default console
;
PDlen           equ     48              ; Length of Process Descriptor
;
p_cns           equ     byte ptr 20h    ; Default CoNSole
p_disk          equ     byte ptr 12h    ; Default disk
p_user          equ     byte ptr 13h    ; Default user
p_list          equ     byte ptr 24h    ; Default list
;
ps_run          equ     0               ; PD run status
pf_keep         equ     2               ; PD nokill flag
;
rsp_top         equ     0               ; RSP offset
rsp_pd          equ     10h             ; PD offset
rsp_uda         equ     40h             ; UDA offset
rsp_bottom      equ     140h            ; End RSP Header
;
qf_rsp          equ     8               ; Queue RSP flag
;
```

```
;--------------------------------------
       CSEG                        ; Small Memory Model
       ORG     0000h
;--------------------------------------
;
ccpm:   int     ccpmint
       ret
;
;--------------------------------------
;
main:   mov cl,q_make! mov dx,offset qd ; Create ECHO queue
       call ccpm                   ;
       mov cl,q_open! mov dx,offset qpb ; Open ECHO queue
       call ccpm                   ;
       mov cl,p_priority! mov dx,200   ; Set priority to normal
       call ccpm                   ;
       mov es,sdatseg              ; ES points to SYSDAT
loop:                              ; Forever
       mov cl,q_read! mov dx,offset qpb ; Read command tail from queue
       call ccpm                   ;
       mov bx,PDadr                ; Set default values from PD
;      mov dl,es:p_disk[bx]        ; P_DISK = 0-15
;      inc dl! mov disk,dl         ; Make disk = 1-16
;      mov dl,es:p_user[bx]        ;
;      mov user,dl                 ;
;      mov dl,es:p_list[bx]        ;
;      mov list,dl                 ;
       mov dl,es:p_cns[bx]         ;
       mov console,dl              ;
;      mov dl,console              ; Set default console
       mov cl,c_set                ;
       call ccpm                   ;
;
; Scan command tail and look for '$' or 0.
; When found, replace with cr,lf,'$'.
;
       lea bx,cmdtail! mov al,'$'! mov ah,0
       mov dx,bx! add dx,131        ;
nextchar:
       cmp bx,dx! ja endcmd         ;
       cmp [bx],al! je endcmd       ; '$' ?
       cmp [bx],ah! je endcmd       ; 0 ?
         inc bx! jmps nextchar      ;
endcmd:
       mov byte ptr [bx],13         ; Carriage Return
       mov byte ptr 1[bx],10        ; Line-Feed
       mov byte ptr 2[bx],'$'       ; String terminator
       lea dx,cmdtail! mov cl,c_writestr ; Write command tail to console
       call ccpm                   ;
       mov dl,console! mov cl,c_detach ; detach console
       call ccpm                   ;
       jmps loop                   ; Done, get next command
;
;--------------------------------------
```

```
        DSEG
;----------------------------------------
;
        ORG     rsp_top
;
sdatseg         dw      0,0,0
                dw      0,0,0
                dw      0,0
;
;----------------------------------------
;
        ORG     rsp_pd
;
pd              dw      0,0             ; Link, thread
                db      ps_run          ; Status
                db      190             ; Priority
                dw      pf_keep         ; Flags
                db      'ECHO    '      ; Name
                dw      offset uda/10h  ; UDA Seg
                db      0,0             ; Disk, user
                db      0,0             ; Load disk, user
                dw      0               ; Mem
                dw      0,0             ; Dvract, wait
                db      0,0             ;
                dw      0               ;
                db      0               ; Console
                db      0,0,0           ;
                db      0               ; List
                db      0,0,0           ;
                dw      0,0,0,0         ;
;
;----------------------------------------
;
        ORG     rsp_uda
;
uda             dw      0,offset DMA,0,0        ; 0
                dw      0,0,0,0                 ;
                dw      0,0,0,0                 ; 10h
                dw      0,0,0,0                 ;
                dw      0,0,0,0                 ; 20h
                dw      0,0,0,0                 ;
                dw      0,0,offset stack_tos,0  ; 30h
                dw      0,0,0,0                 ;
                dw      0,0,0,0                 ; 40h
                dw      0,0,0,0                 ;
                dw      0,0,0,0                 ; 50h
                dw      0,0,0,0                 ;
                dw      0,0,0,0                 ; 60h
;
;----------------------------------------
;
        ORG     rsp_bottom
;
qbuf            rb      131             ; Queue buffer
```

```
;
qd          dw   0          ; Link
            db   0,0        ; Net, org
            dw   qf_rsp     ; Flags
            db   'ECHO    '  ; Name
            dw   131        ; Message length
            dw   1          ; nmsgs
            dw   0,0        ; DQ, NQ
            dw   0,0        ; msgcnt, msgout
            dw   offset qbuf   ; Buffer address
;
dma         rb   128
;
stack       dw   0CCCCh,0CCCCh,0CCCCh
            dw   0CCCCh,0CCCCh,0CCCCh
            dw   0CCCCh,0CCCCh,0CCCCh
            dw   0CCCCh,0CCCCh,0CCCCh
            dw   0CCCCh,0CCCCh,0CCCCh
stack_tos   dw   offset main   ; Start offset
            dw   0          ; Start segment
            dw   0          ; Init flags
;
PDadr       rw   1          ; QPB buffer
;
cmdtail     rb   129        ; Starts here
            db   13,10,'$'  ; Terminators
;
qpb         db   0,0        ; Must be zero
            dw   0          ; Queue ID
            dw   1          ; nmsgs
            dw   offset PDadr  ; Buffer address
            db   'ECHO    '  ; Name to open
;
console     db   0          ;
; disk      db   0          ;
; user      db   0          ;
; list      db   0          ;
;
;---------------------------------------
;
      END


EOF
```

CCPMPRGE.WS4   (Concurrent CP/M Programmer's Reference Guide, Appendix E)

(Retyped by Emmanuel ROCHE.)

Appendix E: 8087 exception handling

This appendix includes an example of an 8087 interrupt handling routine, to demonstrate the requirements for using the 8087 processor. Refer to Intel's "IAPX 86,88 User's Manual" for a description of 8087 exception handling, in the section on "8087 Numeric Data Processor".

In order to guarantee the data integrity for each 8087 process in the multitasking environment, any user-defined exception handler must adhere to a minimum sequence of steps within the exception handler:

1. Save the 8086 environment of the 8086-running process.

2. Save the environment of the 8087-running process. The OWNER_8087 field in SYSDAT will contain the offset of the 8087_running process (see description of SYSDAT in Section 6, "System calls", with the S_SYSDAT system call).

3. Clear the 8087 interrupt request bit in the status word.

4. Disable the 8087 interrupts.

5. Clear the PIC interrupt (this instruction is hardware-dependent).

6. At this point, you might want to modify the 8087 environment image saved in step 2 above.

7. Before enabling the 8086 interrupts, restore the 8087 environment with its status word's interrupt request bit cleared. If the environment is not restored before 8086 interrupts are enabled, and an interrupt occurs (like a tick), a different 8087 process can gain control of the 8087 and swap in its 8087 context. On a second interrupt, or on an IRET instruction, the 8086-running process that happened to be executing the exception handler code is brought back into 8086 context and writes over the new 8087 context.

The user program, which uses its own exception handler, must replace the system's interrupt vector with its own. Once this is done, the system swaps this vector into memory every time the program comes back into 8087 context. The address of the interrupt vector is in the SYSDAT table at offset 00A0h (the description of the SYSDAT Table in included in the description of the S_SYSDAT system call in Section 6, "System calls").

The default exception handler aborts those 8087 programs that have enabled 8087 interrupts and that generate a severe error (such as stack underrun, divide by zero, and so forth). Any other errors are ignored by the default exception handler.

```
ndpint:         ; 8087 interrupt routine
;======

; This exception handler is non-specific, and is meant as an example
; default. It is assumed that, if the 8087 programmer has enabled
; 8087 interrupts and has specified exception flags in the control
; word, then the programmer has also included an exception handler
; to take specific actions within the program before continuing in
; the 8087. This handler will ignore non-severe errors (overflow, etc)
; and will terminate processes with severe errors (divide by zero,
; stack violation).

        push ds                 ; Save current Data Segment
        mov ds,sysdat           ; Get XIOS Data Segment
        mov ndp_SSreg,ss        ; Do stack switch for 8086 environment
        mov ndp_SPreg,sp        ; Save
        mov ss,sysdat           ;
        mov sp,offset ndp_tos   ; Save the 8086 registers
        push ax! push bx        ;
        push cx! push dx        ;
        push di! push si        ;
        push bp! push es        ;
        mov es,sysdat           ; Now, save the 8087 environment
        FNSTENV env_8087        ; Save 8087 process info
        FWAIT                   ;
        FNCLEX                  ; Clear its INT request bit
        xor ax,ax               ;
        FNDISI                  ; Disable its interrupts
        mov al,20h              ; Send 2 interrupts acknowledges:
        out 60h,al              ;   one for master PIC, one for slave.
        mov al,20h              ;
        out 58h,al              ; IN_8087 will check the 8087 error
        call in_8087            ;   condition. If error is severe,
                                ;   it will abort; else, it will
                                ;   return with no changes.
        mov bx,offset env_8087  ; Clear its status word for env restore
        mov byte ptr 2[bx],0    ;
        pop es! pop bp          ; Restore the 8086 environment
        pop si! pop di          ;
        pop dx! pop cx          ;
        pop bx! pop ax          ;
        mov ss,ndp_SSreg        ; Switch back to previous stack
        mov sp,ndp_SPreg        ;
        FLDENV env_8087         ; Restore 8087 env with good status
        FWAIT                   ;
        pop ds                  ; Restore previous Data Segment
        iret                    ;
;
in_8087:
;-------

; Entry: DS = SYSDAT
; Only user-specified error conditions generate interrupts from the 8087.
```

```
        mov bx,owner_8087       ; Get the Process Descriptor
        test bx,bx              ; Check if owner has already
        jz end_87               ;   terminated.
           mov si,offset env_8087 ; If it is a severe error, terminate
           mov ax,statusw[si]   ;
                                ; If not severe, return and continue
           test ax,3Ah          ; 3A = under/overflow, precision,
           jnz end_87           ;   and denormalized operand.
               or p_flag[bx],80h ; Not 3A = zero divide or invalid
                                ;   operation (stack error).
end_87:
        ret                     ;
```

Listing E-1. 8087 exception handling

EOF

CCPMPRGG.WS4   (Concurrent CP/M Programmer's Reference Guide, Appendeis G)

(Retyped by Emmanuel ROCHE.)

Glossary

Base Page
Memory region between 0000h and 0100h, relative to the beginning of the Data Segment, used to hold system parameters. Base Page serves primarily as an interface region between user programs. Note that, in the 8080 Memory Model, the Code and Data are intermixed in the Code Segment.

BCD
Accronym for "Binary Coded Decimal". Representation of decimal numbers using binary digits. See Table B-2, in Appendix B, "ASCII and hexadecimal conversions", for representations of ASCII codes.

BDOS
Basic Disk Operating System. The BDOS manages the Concurrent CP/M file structure, and executes most of the Concurrent CP/M system calls.

block
Basic unit of disk space allocation under Concurrent CP/M. Each disk drive has a fixed block size (BLS) defined in its disk Parameter Block in the XIOS. The block size can be 1, 2, 4, 8, or 16 KB of consecutive bytes. Blocks are numbered relative to zero on a disk. Blocks are not shared between files.

Boolean
Variable that can have only two values; usually interpreted as true/false or on/off.

CheckSum Vector (CSV)
Contiguous data area in the XIOS with one byte for each directory sector to be checked, that is, CKS bytes. A Checksum Vector is initialized and maintained for each logged-in drive. Each directory access by the system results in a checksum calculation that is compared with that in the Checksum Vector. If there is a discrepancy, the drive is set to Read-Only status. This prevents the user from inadvertently switching disks without logging in the new disk with a Ctrl-C. If not logged in, the new disk is treated the same as the old one, and you can destroy data on it if you write to it.

CIO
Character Input/Ouput module. The CIO module handles all character I/O to and from consoles and list devices.

CLI
Command Line Interpreter. The P_CLI system call interprets the command requested in a command line, and performs the system calls needed to open a process, load the command file, and execute the code.

CMD
Filetype for Concurrent CP/M command files. These are machine language object modules ready to be loaded and executed. Any file with this filetype can be executed by simply typing its filename after the drive prompt ("A>"). For example, the program PIP.CMD can be executed by simply typing "pip".

command
Set of instructions that are executed when the command name is typed after the system prompt ("A>"). These instructions can be built in the Concurrent CP/M system, or can reside on disk as a file of type CMD. Concurrent CP/M commands consist of three parts: the command name, the command tail, and a carriage return.

console
Primary I/O device used by Concurrent CP/M. The console usually consists of a CRT screen for displaying output, and a keyboard for input.

control character
Non-printing ASCII character produced on the console by holding down the "Ctrl" (control) key while striking the character key. Ctrl-H means "hold down Ctrl and press H". Control characters are sometimes indicated using the up-arrow symbol ("^"), so Ctrl-H can be represented as ^H. Certain control characters are treated as special commands by Concurrent CP/M.

default buffer
128-byte buffer maintained at 0080h in the Base Page. When the CLI loads a CMD file, it initializes this buffer to the command tail, that is, any characters typed after the CMD file name. The first byte at 0080h contains the length of the command tail, while the command tail itself begins at 0081h. A binary zero terminates the command tail value. The "I" command under DDT-86 initializes this buffer in the same way as the CLI.

default FCB
One of two FCBs maintained at 005Ch and 006Ch in the Base Page. The P_CLI system call initializes the first default FCB from the first delimited field in the command tail, and initializes the second default FCB from the next field in the command tail.

delimiters
ASCII characters used to separate constituent parts of a file specification. The P_CLI system call recognizes certain delimiter characters, as : . = ; < > _ ' blank and carriage return ("ENTER"). Several Concurrent CP/M commands also treat ; [ ] ( ) , and $ as delimiter characters. It is advisable to avoid the use of delimiter characters and lowercase characters in filenames.

directory
Portion of a disk containing entries for each file on the disk, and locations of the blocks allocated to the files. Each file directory entry is in the form of a 32-byte FCB, although one file can have several entries, depending on its size. The maximum number of directory entries supported is specified in the drive's Disk Parameter Block.

directory entry
32-byte entry associated with each disk file. A file can have more than one

directory entry associated with it. There are four directory entries per
directory sector. Directory entries can also be referred to as directory FCBs.

disk, diskette
Magnetic media used for mass storage of data in the computer system. The term
"disk" can refer to a diskette, a removable cartridge disk, or a fixed hard
disk.

Disk Parameter Block (DPB)
Table residing in the XIOS that defines the characteristics of a drive in the
disk subsystem used with Concurrent CP/M. The address of the DPB is in the
Disk Parameter Header, at DPbase + 0Ah. Drives with the same characteristics
can use the same DPB. However, each logical drive must have its own Disk
Parameter Header and DPB. The address of the drive's Disk Parameter Header
must be returned in registers HL when the BDOS calls the SELDSK entry point in
the XIOS. DRV_DPB returns the DPB address.

Disk Parameter Header (DPH)
16-byte area in the XIOS containing information about the disk drive and a
scratchpad area for certain BDOS operations. See the "Concurrent CP/M System
Guide" for further details.

extent (EX)
16 KB consecutive bytes in a file. Extents are numbered from 0 to 31. One
extent can contain 1, 2, 4, w8, or 16 blocks. EX is the extent number field of
an FCB, and is a one-byte field at FCB+12, where FCB labels the first byte in
the FCB. Depending on the Block Size (BLS) and the maximum Data Block Number
(DSM), a directory entry contains 1, 2, 4, 8, or 16 extents. The EX field is
usually set to 0 by the user, but contains the current extent number during
file I/O. The term "Extent Folding" describes directory entries containing
more than one extent. In CP/M version 1.4, each FCB contained only one extent.

FCB
See "File Control Block".

file
Collection of data containing from zero to 242,144 records. Each record
contains 128 bytes, and can contain either binary or ASCII data. Files
consist of one or more 16 KB extent, with 128 records per extent.

File Control Block (FCB)
Thirty-six consecutive bytes maintained and updated by system calls for file
I/O. The FCB fields are described in Section 2.4, "File Control Block
definition".

hex file format
Absolute output of ASM-86 for the Intel 8086. A H86 file contains a sequence
of absolute records, which give a load address and byte values to be stored,
starting at the load address (refer to Section 4.3, "Intel hexadecimal file
format").

I/O
Acronym for "Input/Output" operations, or routines handling the input and
output of data in the computer system.

logical drive
Logically distinct region of a physical drive. A physical drive can be divided into one or more logical drives, and designated with specific drive references (such as "A:" or "C:"). Thus, at the user interface, it appears that there are several disks in the system.

MEM
Memory Module. The Memory Module handles all memory management calls by methods transparent to your applications program.

parse
Separate a command tail into its syntactic parts.

queue
Data structure used by the file system to keep track of system information, such as processes ready to run, locked files, and resources in use by processes. Processes also use queues to communicate with one another. The BDOS system calls create and maintain queues.

Read-Only
Condition in which a logical disk drive can be read, but not written to. A drive can be set to Read-Only status by using the SET utility. This protects the user from switching disks without executing a disk reset. Files can also be set to Read-Only status with the SET utility or the F_ATTRIB system call. Read-Only is often abbreviated as "R/O".

record
Smallest unit of data in a disk file that can be read or written. A record consists of 128 consecutive bytes whose byte displacement in a file is the product of the Record Number times 128. A 128-byte record in a file occupies one 128-byte sector on the diskette. If the blocking and deblocking algorithm is used, several records can occupy each disk sector.

re-entrant code
Code that can be used by one process while another is already executing it. Re-entrant code must not be self-modifying; it must be pure code that does not contain data. The data for re-entrant code can be kept in a separate data area, or placed on the stack.

RSP
Reserved System Process. An RSP is a Concurrent CP/M utility included within Concurrent CP/M during the execution of GENCCPM.

RTM
Real-Time Monitor. The RTM is the nucleus of Concurrent CP/M, managing queues and flags, polling devices, and dispatching and suspending processes. Application programs gain access to RTM functions through system calls.

sector
Unit of data read from and written to the disk by the XIOS. The sector size is dependent on the disk drive hardware, and is usually a power of two, such as 256, 512, 1024, or 2048 bytes. These disk sectors are referred to as "Host Sectors".

source file
ASCII text file usually created with a text editor that is an input file to a program, such as a compiler, assembler, or a text formatter.

stack
Reserved area of memory where the processor saves the return address when it receives a Call instruction. When the processor encounters a Return instruction, it restores the current address on the stack to the Instruction Pointer. Data such as the contents of the registers can also be saved on the stack, on a first-in, last-out basis. The Push instruction places data on the stack, and the Pop instruction removes it. 8086 stacks are 16-bits wide; instructions operating on the stack add and remove stack items one word at a time. An item is pushed onto the stack by decrementing the stack pointer (SP) by 2, and writing the item at the SP address. In other words, the stack grows downward in memory.

SUP
The Supervisor (SUP) manages communications between processes and the operating system kernel, and between other operating system modules. All system calls are intercepted by the SUP.

track
Concentric ring on the disk; the standard IBM single density disks have 77 tracks. Each track consists of a fixed number of numbered sectors. Tracks are numbered from 0 to one less than the number of tracks on the disk. Data on the disk media is accessed by combinations of track and sector numbers.

TMP
Terminal Message Processes. The TMPs are Resident System Processes that intercept command lines from the virtual consoles, check for errors, and pass on executable requests to the CLI. The TMP prints the prompt and some system error messages on your console. Each virtual console has an independent TMP heading defining the console's environment, including the default disk, user number, and console.

transient command file
File of type CMD stored on disk. Such files must be loaded into the system each time they are executed, and therefore execute more slowly than Resident System Processes (RSPs), which are an integral part of the operating system and execute rapidly. Transient commands are created with the GENCMD utility; RSPs are included in the operating system during execution of GENCCPM.

user
logically distinct subdivision of the directory. Each directory can be divided into 16 user numbers.

wildcard
A "?" or "*" character. The BDOS directory search calls matches "?" with any single character, and "*" with multiple characters. Refer to the F_SFIRST and F_SNEXT system calls in Section 6, "System calls", for further details.

XIOS
Extended Input/Output System. In Concurrent CP/M, the BDOS is the invariant

file-handling system, which operates independent of the hardware implementation. The XIOS is the customizable I/O interface configured for your hardware system by the system manufacturer. The XIOS is similar to the BIOS in CP/M and CP/M-86, but it has been extended to implement virtual consoles and associated features.

EOF