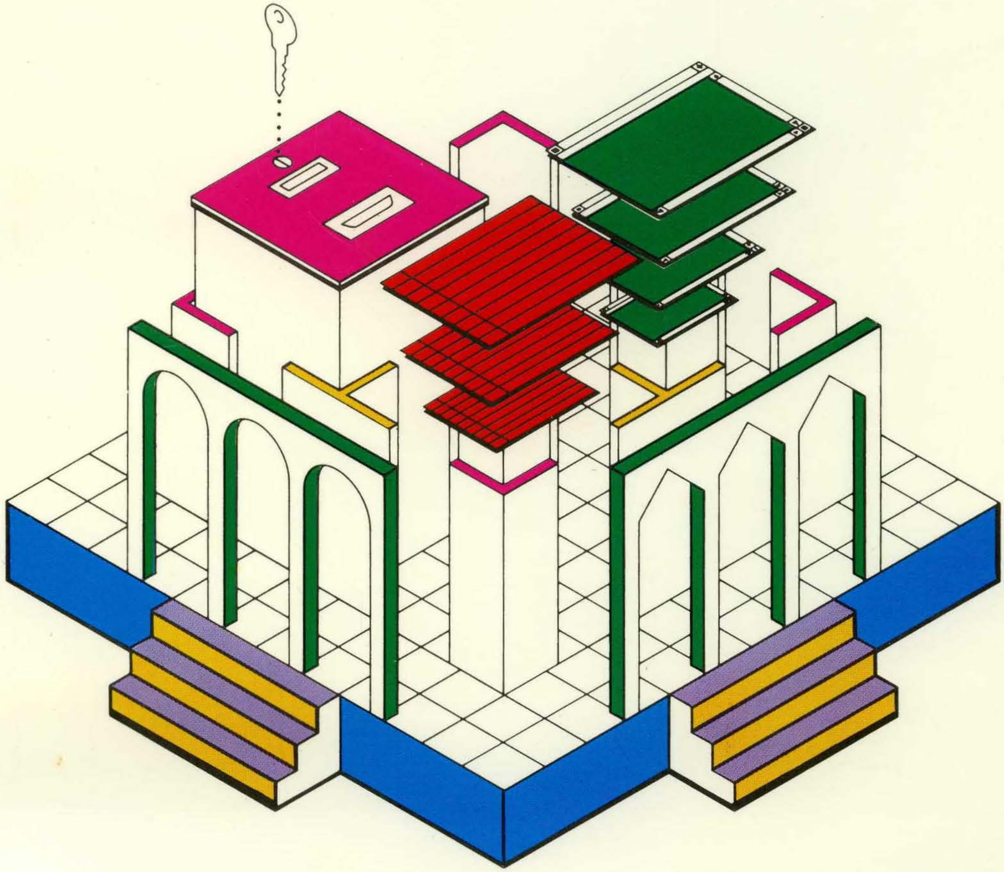
Concurrent™ DOS

Multiuser/Multitasking Operating System

SYSTEMS GUIDE



DIGITAL RESEARCH®

Concurrent™ DOS

SYSTEMS GUIDE

DIGITAL RESEARCH®

COPYRIGHT

DISCLAIMER

DIGITAL RESEARCH INC. MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Digital Research Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research Inc. to notify any person of such revision or changes.

NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or materials and facilities exist on the distribution disk as part of the materials and programs distributed. Many software products include a release note that explains variations from the manual that do constitute modification of the manual and the items included therewith. If such a release note exists, be sure to read it before using the software.

TRADEMARKS

The following are registered trademarks of the listed companies.

Registered Trademark	Company
IBM	International Business Machines
Intel	Intel Corporation
Lotus	Lotus Development Corp.
Microsoft	Microsoft Corporation
Quadram	Quadram Corporation

The following are trademarks of the listed companies.

Trademark	Company
1-2-3	Lotus Development Corp.
Ashton-Tate	Ashton-Tate Corporation
MS-DOS	Microsoft Corporation
Symphony	Lotus Development Corp.

The Concurrent DOS System Guide was printed in the United Kingdom.

November 1987

FOREWORD

The Concurrent™ DOS System Guide is intended for the original equipment manufacturer (OEM) and system programmer responsible for implementing Concurrent DOS XM or Concurrent DOS 386 (hereinafter cited as Concurrent) on a disk-based microcomputer that uses an Intel(R) 8086, 8088, 80186, 80286 or 80386 microprocessor with a real-time clock. Concurrent is a multitasking, real-time operating system that can be implemented in single or multiple user configurations.

Manual Organization

This manual documents the internal, hardware-dependent structures of Concurrent and is arranged as follows:

* Section 1 introduces the major features of the system and describes its levels of interface and the responsibilities of the primary modules.

* Section 2 describes how to generate a Concurrent system with the GENSYS utility.

* Section 3 provides an overview of the physical interface to Concurrent, the Extended Input/Output System (XIOS).

* Section 4 describes XIOS support for character devices.

* Section 5 describes XIOS support for disk devices.

* Section 6 describes Concurrent's support for expanded memory.

* Section 7 describes the special functions required for DOS program support.

* Section 8 details the responsibilities of an interrupt-driven XIOS tick routine.

* Section 9 contains suggestions for debugging the XIOS.

* Section 10 describes an example bootstrap procedure.

* Section 11 offers guidelines for writing hardware-specific utilities for OEM distribution.

* Section 12 outlines the changes required in Concurrent's end-user documentation when an OEM makes certain modifications to the operating system.

* Appendix A describes removable media support.

* Appendix B offers considerations for implementing graphics capabilities.

* Appendix C describes specific details for Concurrent 386.

Example XIOS

Many sections of this manual refer to the example XIOS. Digital Research[R] provides an example XIOS for the IBM PC, XT, AT and PS2 family of personal computers. The example XIOS contains all the major features of a multi-user, hard-disk, paged-memory Concurrent implementation including support of DOS applications via serial terminals on Concurrent DOS 386 and character windowing support for a video-mapped display. Source code is included on the Concurrent distribution disks. We strongly suggest that you assemble the source files according to the instructions in Section 2 and that you refer often to the assembly listing while reading this manual.

Concurrent Documentation

The presentation of information in this manual assumes that you are familiar with the following documents:

- The Concurrent DOS User's Guide (cited as the User's Guide) describes the various features of Concurrent's user interface.

- The Concurrent DOS Reference Guide (cited as the User's Reference Guide) is a user reference manual that provides a detailed description of every Concurrent command.

- The Concurrent DOS Programmer's Guide (cited as the Programmer's Guide) describes the system's application programmer interface.

- The Assembler plus Tools Guide manuals document the RASM-86™ relocating assembler, the LINK-86™ linkage editor, the LIB-86™ software librarian, XREF-86™ the cross referencer and the SID-86 symbolic instruction debugger. This manual is cited as the Programmer's Reference Guide.

Digital Research supports the user and software interfaces described in the User's Guide, User's Reference Guide, and Programmer's Guide; Digital Research does not support any additions or modifications made to Concurrent by an OEM or distributor. The OEM or distributor must also support the hardware interface (XIOS) for a particular hardware environment.

Terminology

Concurrent system functions available through the logically invariant software interface are called system calls. The names of all data structures internal to the operating system or XIOS are capitalized. For example, the Concurrent system data segment is referred to as the SYSDAT Area or simply SYSDAT. Terms that are particular to a specific section of this manual are defined in that section.

Contents

5 Disk Devices

6 Expanded Memory Support

7 PC/AT ROS Support

10 Bootstrap Adaption

11 OEM Disk Utilities

Tables

Figures

Listings

SYSTEM OVERVIEW

Concurrent is a multitasking, real-time operating system. It can be configured to support one or more user terminals. Each user terminal can run multiple tasks simultaneously on virtual consoles. Concurrent's extended features include intercommunication and synchronization of independently running processes and expanded memory support that allows the system to address up to eight megabytes of memory. Concurrent is designed to be implemented in a large variety of hardware environments and as such, you can customize it to fit that hardware environment and/or user's needs.

Concurrent also supports many IBM Personal Computer Disk Operating System (PC DOS) and MS-DOS™ programs. The standard disk media type (DOS) is fully compatible with PC DOS and MS-DOS disks. In this manual, the term DOS refers to both PC-DOS and MS-DOS. In addition, for compatibility with previous versions of Concurrent, the XIOS may support CP/M media and a special utility supplied with Concurrent allows files to be moved easily to and from the CP/M media.

Concurrent consists of three levels of interface: the user interface, the logically invariant software interface, and the hardware interface. The user interface distributed by Digital Research is a Resident System Process (RSP) called the Terminal Message Process (TMP). The TMP accepts commands from the user and either performs the function itself or passes the command to the operating system via the Command Line Interpreter (CLI). The Command Line Interpreter in the operating system kernel either invokes an RSP or loads a disk file to perform the command.

The logically invariant interface to the operating system consists of the system calls described in the Programmer's Guide. The logically invariant interface also connects transient and resident processes with the hardware interface.

The physical interface, or XIOS (extended I/O system), communicates directly with the particular hardware environment. The XIOS is composed of a set of functions which are called by processes that need physical I/O. Sections 3 through 7 describe these functions. Figure 1-1 shows the relationships between the three interfaces.

This section describes the modules that comprise a typical Concurrent system. It is important that you understand this material before you try to customize the operating system for a particular application.

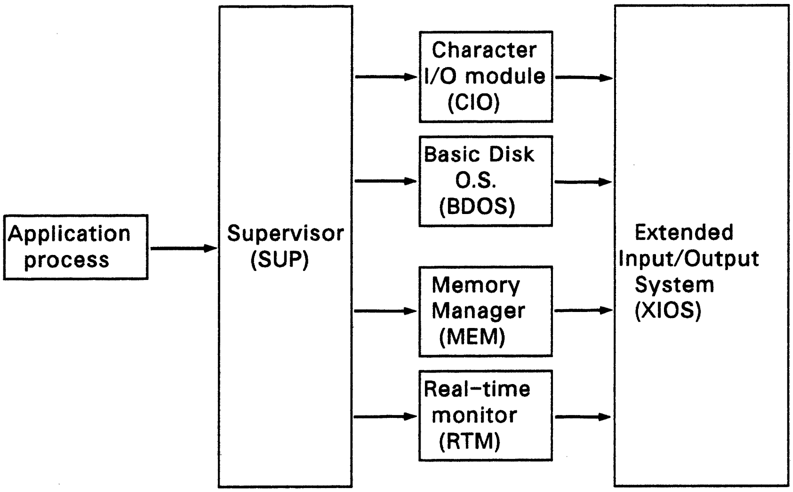


Figure 1-1. Concurrent Interfaces

1.1 CONCURRENT ORGANIZATION

Concurrent is composed of seven basic code modules. The Real-time Monitor (RTM) handles process-related functions, including dispatching, creation, and termination, as well as the Input/Output system state logic. The Memory module (MEM) manages memory and handles the Memory Allocate and Memory Free system calls. The Character I/O module (CIO) handles all console, list, and auxiliary device functions. The Basic Disk Operating System (BDOS) manages the file system. These four modules communicate with the Supervisor (SUP), the Extended Input/Output System (XIOS) and the Dos function support module (PCMODE).

The SUP and PCMODE module manage the interaction between transient processes, such as user programs, and the system modules. All function calls go through a software interrupt interface to SUP or PCMODE.

The XIOS handles the physical interface to a particular hardware environment. Any of the Concurrent modules can call the XIOS to perform specific hardware-dependent functions.

All operating system code modules, including the SUP and XIOS, share a data segment called the System Data Area (SYSDAT). The beginning of SYSDAT is the SYSDAT DATA, a well-defined structure containing public data used by all system code modules. Following this fixed portion are local data areas that belong to specific code modules. The XIOS area is the last of these code module areas. Following the XIOS Area are Table Areas, used for the Process Descriptors, Queue Descriptors, System Flag Tables, and other operating system tables. These tables vary in size depending on the options you choose when you use GENSYS during system generation. See Section 2, "Building the System," for a description of how to use GENSYS.

The Resident System Processes (RSPs) occupy the area in memory immediately before the SYSDAT module. The RSPs you select at system generation time become an integral part of the operating system. For more information on RSPs, see Section 1.11, "Resident System Processes," and the Programmer's Guide.

Concurrent loads all transient programs into the Transient Program Area (TPA). The TPA for a given implementation of Concurrent is determined at system generation time.

1.2 MEMORY LAYOUT

The Concurrent DOS operating system area can exist anywhere in memory except over the interrupt vector area. To provide complete DOS support, neither the system nor the TPA should be located below paragraph 60H. You define the exact location of Concurrent during system generation. The GENSYS program determines the memory locations of the system modules that make up Concurrent based upon system generation parameters and the size of the modules. In the case of a system that runs from Read only memory (ROM) then the image is built in the usual way except that the location of Concurrent is given as the start of ROM. The image is then blown into one or more ROMs as required. At "power up" the data part of the ROM image is first copied by a small routine to its location in Random access memory (RAM).

The XIOS is written as a small model (8080 model in previous versions).

The CCPM.SYS file is built by GENSYS from some or all of the following files depending on the requirement.

CDOS.CON	-	Concurrent Kernel (SUP,PCMODE,MEM,RTM,CIO,BDOS)
XIOS.CON	-	Input/output system
CLOCK.RSP	-	Clock process
PIN.RSP	-	Physical input process
PCTMP.RSP	-	Terminal process
INIT.CON	-	Initialisation code
NET.CON	-	Network code

Figure 1-2 shows the memory map of Concurrent and the structure of the CCPM.SYS file

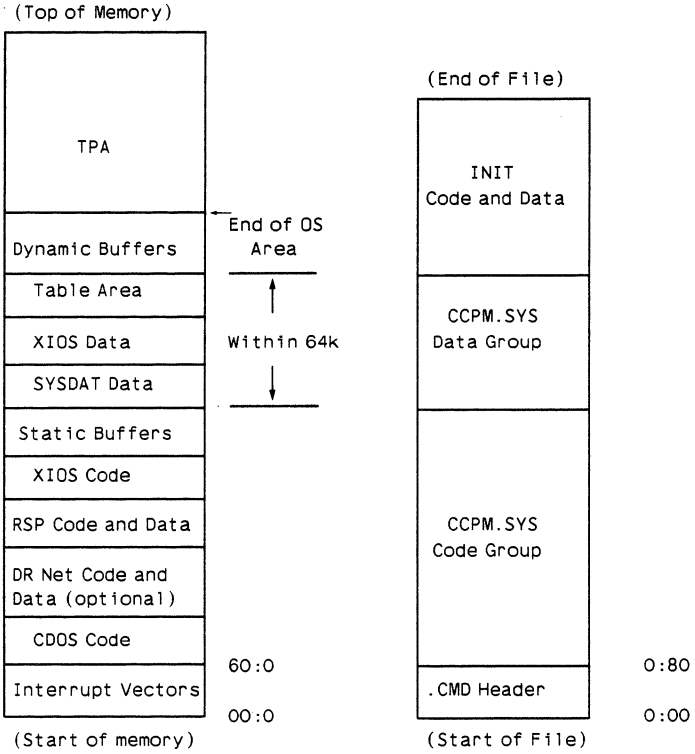


Figure 1-2. Memory Layout and File Structure

1.3 SUPERVISOR and PCMODE

The Concurrent Supervisor (SUP) and (PCMODE) manage the interface between system and transient processes and the invariant operating system.

The SUP module also contains system calls that invoke other system calls, like P_LOAD (Program Load) and P_CLI (Command Line Interpreter). Table 1-1 lists the system calls contained in the SUP module.

Table 1-1. Supervisor System Calls

System Call	Number	Hex
F_PARSE	152	98
P_CHAIN	47	2F
P_CLI	150	96
P_LOAD	59	3B
P_RPL	151	97
S_BDOSVER	12	0C
S_BIOS	50	32
S_OSVER	163	A3
S_SYSDAT	154	9A
S_SERIAL	107	6B
T_SECONDS	155	9B
T_GETTIME	83	53
T_SETTIME	84	54
T_GETDATE	85	55
T_SETDATE	86	56
S_MEMORY	89	59
P_PATH	92	5C
P_EXEC	93	5D
P_EXITCODE	94	5E
S_SETCOUNTRY	95	5F
S_GETCOUNTRY	96	60

1.4 REAL-TIME MONITOR

The Real-time Monitor (RTM) is Concurrent's multitasking kernel. It handles process dispatching, queue and flag management, device polling, system timing tasks, and the logical interrupt system. The primary function of the RTM is to transfer the CPU resource from one process to another, a task accomplished by the RTM dispatcher. At every dispatch operation, the dispatcher stops the currently running process and stores its state in the Process Descriptor (PD) and User Data Area (UDA) associated with that process. The dispatcher then selects the highest-priority process in the ready state and restores it to execution, using the data in its PD and UDA. A process is in the ready state if it is waiting for the CPU resource only. The new process continues to execute until it needs an unavailable resource, a resource needed by another process becomes available, or an external event, such as an interrupt, occurs. At this time the RTM performs another dispatch operation, allowing another process to run.

The Concurrent RTM dispatcher also performs device polling. The XIOS waits for a polled device through the RTM DEV_POLL system call.

When the XIOS needs to wait for an interrupt, it issues a DEV_WAITFLAG system call on a logical interrupt device. When the appropriate interrupt actually occurs, the XIOS calls the DEV_SETFLAG system call to put process in the ready state. The interrupt routine then performs a Far Jump to the RTM dispatcher, which reschedules the interrupted process, and all other ready processes that are not yet on the Ready List. At this point, the dispatcher brings the process with the highest priority into execution.

The system clock typically generates interrupts, or clock ticks, 50 or 60 times per second. This allows Concurrent to effect process time slicing. Since the operating system waits for the tick flag, the XIOS tick interrupt routine must execute a Concurrent DEV_SETFLAG system call at each tick (see Section 8, "XIOS Tick Interrupt Routine") and then perform a Far Jump to the RTM dispatcher. Processes with equal priority are scheduled for the CPU resource in round-robin fashion. If no process is ready to use the CPU, Concurrent remains in the dispatcher until an interrupt occurs or a polling process is ready to run.

The RTM also handles queue management. System queues are composed of a Queue Descriptor, which contains the queue name and other parameters, and a Queue Buffer, which can contain a specified number of fixed-length messages. Processes read these messages from the queue on a FIFO basis. A process can write to or read from a queue either conditionally or unconditionally.

When a process attempts a conditional read from an empty queue, or a conditional write to a full one, the RTM returns an error code to the calling process. A process that attempts an unconditional read or write in these situations is suspended until the requested operation can be accomplished.

Table 1-2 lists the RTM system calls. See the Programmer's Guide for more complete information about RTM functions.

Table 1-2. Real-time Monitor System Calls

System Call	Number	Hex	System Call	Number	Hex
DEV_SETFLAG	133	85	P_TERM	143	8F
DEV_WAITFLAG	132	84	P_TERMCPM	0	00
DEV_POLL	131	83	Q_CREAT	138	8A
P_ABORT	157	9D	Q_CWRITE	140	8C
P_CREATE	144	90	Q_DELETE	136	88
P_DELAY	141	8D	Q_MAKE	134	86
P_DISPATCH	142	8E	Q_OPEN	135	87
P_PDADR	156	9C	Q_READ	137	89
P_PRIORITY	145	91	Q_WRITE	139	8B
DEV_FLAGFREE	88	58			

1.5 MEMORY MANAGEMENT MODULE

The Memory Management module (MEM) handles all memory functions for both conventional (non-banked) and expanded memory (banked). Expanded memory is defined as memory (up to eight megabytes) that can be enabled or disabled at a specific address on command by the XIOS. Memory that cannot be changed in this way is conventional memory. The MEM module uses linked lists to allocate and deallocate the memory and to allow the dispatcher to ensure that a process always has its memory in context when it runs. The XIOS creates or trims these lists at initialisation time depending on the hardware requirement.

Table 1-3. Memory Management System Calls

System Call	Number	Hex	System Call	Number	HEX
M_ALLOC	128/129	80/81	MC_ALLOC		
M_FREE	130	82	MC_ABSALLOC	56	38
MC_ABSMAX	54	36	MC_FREE	57	39
MC_ALLFREE	58	3A	MC_MAX	53	35

Note: The MC_ prefix denotes system calls supported for compatibility with CP/M-86 and MP/M-86™. These calls internally execute the M_ALLOC and M_FREE system calls. See the Programmer's Guide for descriptions of the memory management system calls.

1.6 CHARACTER I/O MODULE

The Character Input/Output (CIO) module handles all console, list, and auxiliary device I/O and provides an interface to the XIOS and the PIN (Physical Input) process. The PIN process handles keyboard input for each user terminal.

The XIOS associates device control blocks with each of the devices managed by the CIO. The Console Control Block (CCB), List Control Block (LCB), and Auxiliary Control Block (ACB) data structures are described in Sections 4.1, 4.4, and 4.5, respectively.

Table 1-4 lists the CIO system calls.

Table 1-4. Character I/O System Calls

System Call	Number	Hex	System Call	Number	Hex
C_ASSIGN	149	95	C_STAT	11	0B
C_ATTACH	146	92	C_WRITE	02	02
C_CATTACH	162	A2	C_WRITEBLK	111	6F
C_DELIMIT	110	6E	C_WRITESTR	09	09
C_DETACH	147	93	L_ATTACH	158	9E
C_GET	153	99	L_CATTACH	161	A1
C_MODE	109	6D	L_DETACH	159	9F
C_RAWIO	06	06	L_GET	164	A4
C_READ	01	01	L_SET	160	A0
C_READSTR	10	0A	L_WRITE	05	05
C_SET	148	94	L_WRITEBLK	112	70

The Programmer's Guide presents an overview of the CIO and describes the CIO system calls in detail. The XIOS character device functions are described in Section 4.

1.7 BASIC DISK OPERATING SYSTEM

Table 1-5 lists the Concurrent Basic Operating System (BDOS) system calls. These calls handle all file system functions. The Programmer's Guide describes the BDOS in detail.

Table 1-5. BDOS System Calls

System Call	Number	Hex	System Call	Number	Hex
DRV_ACCESS	38	26	F_MAKE	22	16
DRV_ALLOCVEC	27	1B	F_MULTISEC	44	2C
DRV_DPB	31	1F	F_OPEN	15	0F
DRV_FLUSH	48	30	F_PASSWD	106	6A
DRV_GET	25	19	F_READ	20	14
DRV_GETLABEL	101	65	F_READRAND	33	21
DRV_LOGINVEC	24	18	F_RANDREC	36	24
DRV_RESET	37	25	F_RENAME	23	17
DRV_ROVEC	29	1D	F_SFIRST	17	11
DRV_SET	14	0E	F_SIZE	35	23
DRV_SETLABEL	100	64	F_SNEXT	18	12

Table 1-5. (Cont'd)

System Call	Number	Hex	System Call	Number	Hex
DRV_SETRO	28	1E	F_TIMEDATE	102	66
DRV_SPACE	46	2E	F_TRUNCATE	99	63
F_ATTRIB	30	1E	F_UNLOCK	43	2B
F_CLOSE	16	10	F_USERNUM	32	20
F_DELETE	19	13	F_WRITE	21	15
F_DMASEG	51	33	F_WRITERAND	34	22
F_DMAGET	52	34	F_WRITEXFCB	103	67
F_DMAOFF	26	1A	F_WRITEZF	40	28
F_ERRMODE	45	2D	T_GET	105	69
F_LOCK	42	2A	T_SET	104	68
DEV_LOCK	90	5A	DEV_UNLOCK	91	5B
F_DOS	113	71			

1.8 EXTENDED I/O SYSTEM

The Extended Input/Output System (XIOS) manages Concurrent's interface to the hardware. By modifying the XIOS, you can run Concurrent in a large variety of hardware environments.

The XIOS recognizes two basic types of I/O devices: character devices and disk devices. Character devices are devices that handle one character at a time, while disk devices handle random blocked I/O using data blocks sized from one physical disk sector to the number of physical sectors in 16K bytes. Devices that vary from these two models must be implemented within the XIOS so that they appear to the other operating system modules as standard Concurrent I/O devices.

Sections 4, "Character Devices" through 7, "PC Mode Character I/O" contain detailed descriptions of the XIOS functions.

1.9 XIOS REENTRANCY

Concurrent allows multiple processes to use certain XIOS functions simultaneously and guarantees that only one process can use a particular physical device at any given time. However, some XIOS functions handle more than one physical device, and thus their interfaces must be reentrant. An example of this is the IO_CONOUT (2) function to which the calling process passes the virtual console number. There can be several processes using the function, each writing a character to a different virtual console or character device; however, only one process is actually sending a character to a given device at any time.

The IO_STATLINE (8) function is reentrant: the CLOCK process calls it once per second, and the PIN process calls it on screen switches and on CTRL-S, CTRL-P, and CTRL-O input.

The IO_SELDSK (9), IO_READ (10), IO_WRITE(11), and IO_FLUSH (12) file functions are assumed to be protected by the BDOS or the DEV_LOCK function so that only one process may access them at a time. Because of this protection, none of these XIOS functions need to be reentrant.

1.10 THE SYSTEM DATA AREA

The System Data Area (SYSDAT) is the data segment for all modules of Concurrent. As shown in Figure 1-3, the SYSDAT segment is composed of four main areas. The first part is the fixed-format portion, containing global data used by all modules. This is the SYSDAT DATA. It contains system variables, including values set by GENSYS and pointers to the various system tables. The Internal Data portion contains fields of data that belong to individual operating system modules. The XIOS data begins at the end of this second area of SYSDAT. The fourth portion of SYSDAT is the System Table Area. The System Table Area is generated and initialized by the GENSYS system generation utility.

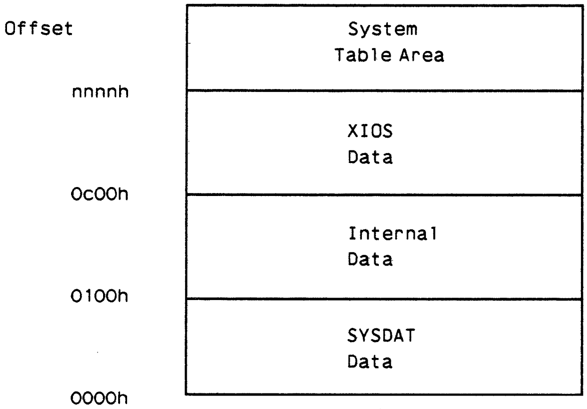


Figure 1-3. SYSDAT

Figure 1-4 shows the format of the SYSDAT DATA. Table 1-6 describes the SYSDAT DATA fields.

<table>
<tr><td>00H</td><td colspan="4">SUP ENTRY</td><td colspan="4">RESERVED</td></tr>
<tr><td>08H</td><td colspan="4">RESERVED</td><td colspan="4">RESERVED</td></tr>
<tr><td>10H</td><td colspan="4">RESERVED</td><td colspan="4">RESERVED</td></tr>
<tr><td>18H</td><td colspan="4">RESERVED</td><td colspan="4">RESERVED</td></tr>
<tr><td>20H</td><td colspan="4">RESERVED</td><td colspan="4">RESERVED</td></tr>
<tr><td>28H</td><td colspan="4">XIOS ENTRY</td><td colspan="4">XIOS INIT</td></tr>
<tr><td>30H</td><td colspan="8">RESERVED</td></tr>
<tr><td>38H</td><td colspan="4">DISPATCHER</td><td colspan="4">PDISP</td></tr>
<tr><td>40H</td><td colspan="2">CCPMSEG</td><td colspan="2">RSPSEG</td><td colspan="2">ENDSEG</td><td>RESER</td><td>NVCNS</td></tr>
<tr><td>48H</td><td>NLCB</td><td>NCCB</td><td>N_
FLAGS</td><td>SRCH
DISK</td><td colspan="2">MMP</td><td>RESER</td><td>DAY
FILE</td></tr>
<tr><td>50H</td><td>TEMP
DISK</td><td>TICKS
/SEC</td><td colspan="2">LUL</td><td colspan="2">CCB</td><td colspan="2">FLAGS</td></tr>
<tr><td>58H</td><td colspan="2">MDUL</td><td colspan="2">MFL</td><td colspan="2">PUL</td><td colspan="2">QUL</td></tr>
<tr><td>60H</td><td colspan="8">QMAU</td></tr>
<tr><td>68H</td><td colspan="2">RLR</td><td colspan="2">DLR</td><td colspan="2">DRL</td><td colspan="2">PLR</td></tr>
<tr><td>70H</td><td colspan="2">RESERVED</td><td colspan="2">THRDRT</td><td colspan="2">QLR</td><td colspan="2">MAL</td></tr>
<tr><td>78H</td><td colspan="2">VERSION</td><td colspan="2">VERNUM</td><td colspan="2">CCPMVERNUM</td><td colspan="2">TOD_DAY</td></tr>
</table>

Figure 1-4a. SYSDAT Format

80H	TOD _HR	TOD _MIN	TOD _SEC	NCON DEV	NLST DEV	NCIO DEV	LCB	
88H	OPEN_FILE		LOCK_ MAX	OPEN_ MAX	OWNER_ 8087		ACB	
90H	RESERVED							
98H	MWDR		BCB_ROOT		NACB	PSD	RESER	XPCNS
A0H	OFF_8087				SYS_87_OF			
A8H	RESERVED		DEVHEAD				PHASE1R	
B0H	HASHROOT		INDMA		NDRV	TTIKS	SYSDATMEM	
B8H	ENVSIZE				RESERVED			
C0H	RESERVED							
C8H	V386_PTR		COUNTRY		INTL_XLAT			
D0H	TICK_CNT				XIOS_DISP			
D8H	BUFFERS		RESERVED					
E0H	RESERVED							
E8H	RESERVED							
F0H	RESERVED							
F8H	RESERVED							

Figure 1-4b. SYSDAT Format

Table 1-6. SYSDAT Data Fields

Field	Description
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 XIOS entry point for intermodule communication. All XIOS function calls go through this entry point.
XIOS INIT	Double-word address of the XIOS Initialization entry point. System hardware initialization takes place by a call through this entry point.

Table 1-6. (Cont'd)

Field	Description
DISPATCHER	Double-word address of the Dispatcher entry point that handles interrupt returns. Execute a JMPF instruction to this address instead of using an IRET (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. Interrupt handlers that make DEV_SETFLAG calls end with a JMPF to the address stored in the DISPATCHER field. Refer to Sections 3.4, 3.5, 8, and the example XIOS interrupt routines for more detailed information.
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. Use this location as the exit point when the XIOS releases a resource that another process might want.
CCPMSEG	Starting paragraph of the operating system area. This is also the Code Segment of the Supervisor.
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 uninitialized RAM allocated to the operating system by GENSYS. These include the Directory Hashing, Disk Data, and XIOS ALLOC buffers. These buffer areas, however, are not part of the CCPM.SYS file.
NVCNS	Number of virtual consoles, copied from the XIOS Header by GENSYS.
NLCB	Number of List Control Blocks, copied from the XIOS Header by GENSYS.
NCCB	Number of Character Control Blocks, copied from the XIOS Header by GENSYS.
NFLAGS	Number of system flags as specified by GENSYS.
SRCHDISK	Default search disk. The Command Line Interpreter (CLI) looks on this disk if it cannot open the command file on the user's current (or default) disk. This disk letter is set by Concurrent to L:. This L: virtual drive points to the sub-directory from which the application was loaded and this enables overlays to be located.
MMP	Maximum memory allowed per process. Set during GENSYS.
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 by GENSYS.
TEMP DISK	Default disk for temporary files. Programs that create temporary files should use this disk. Set by GENSYS.
TICKS/SEC	The number of system ticks per second.

Table 1-6. (Cont'd)

Field	Description
LUL	Locked Unused List. This is the link list root of unused Lock List Items. Concurrent uses Lock List Items to manage open files and records locked for executing processes. Lock List Items are also used to store information on subdirectory paths and files opened in unlocked mode. The number of Lock List Items is determined by the value you set for NOPENFILES during GENSYS. See Section 2.4 for a description of related system parameters and the Programmer's Guide for information about Concurrent's security mechanisms for file access.
CCB	Address of the Character Control Block Table, copied from the XIOS Header by GENSYS.
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 (PDs).
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 are in the ready state.
PLR	Poll List Root. Linked list of PDs that are polling on devices.
THRDRT	Thread List Root. Linked list of all current PDs in the system. The list is threaded though the THREAD field of the PD instead of the LINK field. See the Programmer's Guide for a description of the Process Descriptor's fields.
QLR	Queue List Root. Linked list of all System Queue Descriptors.
MAL	Memory Allocation List. Linked list of active Memory Allocation Units (MAUs). A MAU is created from one or more memory partitions.
VERSION	Address, relative to CCPMSEG, of ASCII version string.
VERNUM	Concurrent version number (returned by the S_BDOSVER system call).
CCPMVERNUM	Concurrent 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.

Table 1-6. (Cont'd)

Field	Description
TOD_SEC	Time of Day. Second of the minute.
NCONDEV	Number of XIOS consoles, copied from the XIOS Header by GENSYS.
NLSTDEV	Number of XIOS list devices, copied from the XIOS Header by GENSYS.
NCIODEV	Total number of character devices (NCONDEV + NLSTDEV).
LCB	Offset of the List Control Block Table, copied from the XIOS Header by GENSYS.
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 are set to indicate drives that contain open files.
LOCK_MAX	Maximum number of locked records per process. Set during GENSYS.
OPEN_MAX	Maximum number of open disk files per process. Set during GENSYS.
OWNER_8087	Process currently owning the 8087. Set to 0 if 8087 is not owned. Set to 0FFFFH if no 8087 present.
ACB	Word-length address of the Auxiliary Control Block Table copied from the XIOS Header by GENSYS.
MWDR	Memory Window Descriptor Root. Linked list of Memory Window Descriptors (MWDs) that describes the location of available logical address windows for mapping to physical memory pages. See Section 6 for a description of how Concurrent uses these structures for expanded memory support.
BCB_ROOT	This is the segment of the first Disk Buffer Control Block.
NACB	Number of Auxiliary Control Blocks, copied from the XIOS Header by GENSYS.
PSD	Physical Search Disk. This is the default system disk as copied from the SYSDISK field by the SUP INIT routine. The BDOS uses PSD as the initial drive for drives N and O and the system drive, P. Concurrent searches the system drive when it cannot find a command file on the default disk.
XPCNS	Number of physical consoles.
OFF_8087	Double word pointer to the 8087 interrupt vector in low memory.
SYS_87_OF	Double word pointer to the default 8087 exception handler.
DEVHEAD	Double word pointer to the first device driver
PHASE1R	Segment of first RSP in Phase 1
HASHROOT	DOS Media Hash Root
INDMA	PD of current DMA process

Table 1-6. (Cont'd)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>NDRV</td><td>Number of physical drives</td></tr>
<tr><td>TTIKS</td><td>Tiks expired in this second</td></tr>
<tr><td>SYSDATMEM</td><td>This is the root of a linked list of SYSDAT memory descriptors (SMDs) which describe the free memory in SYSDAT. The OEM can insert extra SMDs into the linked list to add free SYSDAT memory to the list. All memory must be allocated using the S_MEMORY function. The format of an SMD is:</td></tr>
<tr><td>OOH</td><td><table><tr><td>LINK</td><td>OFFSET</td><td>SIZE</td></tr></table></td></tr>
<tr><td>ENVSIZE</td><td>Specifies the number of bytes to be allocated to the environment for each TMP.</td></tr>
<tr><td>V386PTR</td><td>Offset to XIOS 386 data area</td></tr>
<tr><td>COUNTRY</td><td>Initial country code, set by GENSYS or INIT.</td></tr>
<tr><td>INTL_XLAT</td><td>Double word point to a routine that will convert an international character (ie > 80h) in AL to the corresponding uppercase character for the current country. This value should only be modified by the S_SETCOUNTRY function.</td></tr>
<tr><td>TICK_CNT</td><td>The TICK_CNT is incremented every tick by the TICK interrupt handler and therefore contains the total number of ticks for the time the system has been active. This value is used to generate the hundredths of a second field by Concurrent. This field can also be used by the XIOS to calculate how long it has been since a drive was last used.</td></tr>
<tr><td>XIOS_DISP</td><td>XIOS Dispatch intercept vector
A far call uses this vector every time a process is brought into context. This allows the XIOS to remap the Video RAM etc. All segment registers and the SP must be preserved by the XIOS dispatch code. This vector is called with the following parameters:
DS = SYSDAT
BX = PD of new process
ES = UDA of new process</td></tr>
<tr><td></td><td>BUFFERS Number of disk buffers allocated.</td></tr>
</table>

1.11 RESIDENT SYSTEM PROCESSES

Resident System Processes (RSPs) are an integral part of the operating system. At system generation, the GENSYS RSP List menu lets you select which RSPs to include in the operating system. GENSYS then places all selected RSPs in a contiguous area of RAM starting at the end of the BDOS code. RSPs are permanently resident within the Operating System Area and do not have to be loaded from disk when they are needed. Concurrent itself uses three RSPs, PIN, TMP and CLOCK.

Concurrent automatically allocates a Process Descriptor (PD) and User Data Area (UDA) for a transient program, but each RSP is responsible for the allocation and initialization of its own PD, UDA, and Queue Descriptor (QD). Concurrent uses the PD and QD structures declared within an RSP directly if they fall within 64K of the SYSDAT segment address. If the RPS's PD and QD are outside of 64K, they are copied to a PD or QD allocated from the Process Unused List or the Queue Unused List. In either case, the PD and QD of the RSP lie within 64K of the beginning of the SYSDAT Segment. This allows RSPs to occupy more area than remains in the 64K SYSDAT segment.

See the descriptions of the process system calls and the queue management calls in the Programmer's Guide for definitions of the UDA and QD structures, respectively. The Programmer's Guide also contains details on creating and using RSPs.

1.12 DOS DEVICE DRIVER SUPPORT

Concurrent supports DOS installable drivers for device, fixed disks, and expanded memory.

The Init module checks for the presence of the ASCII file CCONFIG.SYS on the root of the current disk. If it finds CCONFIG.SYS, it reads the file and executes the following commands (if present).

DEVICE = filespec	- Loads a device driver
FIXED-DEVICE = filespec	- Loads a fixed disk device driver
EMM = filespec	- Loads an expanded memory driver
EEMM = filespec	- Loads an expanded memory driver and the SCEPTER.CMD program.
COUNTRY = nnn	- nnn is the required country code
LASTDRIVE = d	- d is the last drive (A - Z)
ENV-SIZE = nnnn	- environment size in bytes (128-32,750)
BREAK = ON	- or OFF (set keyboard break)
BUFFERS = nnn	- defines the number of additional disk buffers

Note: Filespec is a full path and filename.

See the Programmer's Guide for detailed explanations on how to construct CCONFIG.SYS and specifications for writing a DOS device driver.

BUILDING THE SYSTEM

The GENSYS utility allows you to build the Concurrent system image file, CCPM.SYS. This program combines an assembled XIOS and the system modules provided on the distribution disks. GENSYS also displays menus that allow you to define

* certain hardware-dependent variables
* the amount of memory to reserve for system data structures
* the Resident System Processes to be included in CCPM.SYS
* other system parameters.

This section describes how to use the GENSYS utility.

Note: You can generate a Concurrent system by running GENSYS under an existing DOS Plus or Concurrent system.

See Section 9, "Debugging the XIOS," for debugging suggestions.

2.1 REQUIRED FILES

GENSYS requires that the following files be present on the current disk:

* INIT.CON -- Concurrent Initialisation module
* CDOS.CON -- Concurrent Kernel module
* XIOS.CON -- Input/Output System module
* PIN.RSP -- Physical keyboard input process
* TMP.RSP -- Terminal Message Process
* CLOCK.RSP -- CLOCK process
* FLUSH.RSP -- DOS terminal flush process (CDOS 386 only)

If GENSYS cannot find these files, it displays the following error message:

```
Can't find these modules: <FILESPEC>...{<FILESPEC>}
```

where FILESPEC is the name of the missing file.

An extension of RSP denotes a Resident System Process file. Concurrent requires the PIN, TMP, and CLOCK RSPs to run.

All of the modules listed except XIOS.CON are provided on the distribution disk. An unassembled XIOS is provided as a model. You should write the XIOS as small model program (separate code and data) originating the data at location 0C00H.

Note: For compatibility the 8080 model is still supported by GENSYS. If the 8080 model is used combine the XIOS headers at 0C00H as before.

Use the RASM-86 relocatable assembler and the LINK-86 linker according to the following guidelines:

* Include CSEG and DSEG directives in every module to instruct the assembler to place the code and data in the appropriate group. If the segments are named and grouped then this will allow alignment of data items on word boundaries.

* The first module named in the LINK-86 command line that contains code or data should be the module that defines the XIOS Header. This module should define the data starting at "ORG 0C00H" and immediately define the header data fields. It should also define the code starting at "ORG 0" and immediately define the required code segment fields.

  See section 3.1 for a description of the XIOS Header.

* Use the LINK-86 "DATA[ORIGIN[0]]" option to prevent the linker from automatically adding a base page.

2.2 INVOKING GENSYS

GENSYS has the following command syntax:

```
GENSYS [-xm] [-386] [<input file] [>output file]
```

Both the input and output files are optional. GENSYS reads the input file to set parameters. See section 2.11 below for instructions on creating input files for GENSYS. Specify an output file to record the screen displays produced by GENSYS.

Before displaying the Main Menu, GENSYS checks the current drive for the files necessary to construct the operating system image. If a module is not found, GENSYS displays the message shown above and terminates operation. Otherwise, the Main Menu is displayed.

The switch -xm instructs GENSYS to look for the files CDOSXM.CON and XIOSXM.CON rather than CDOS.CON and XIOS.CON. The switch -386 operates in the same way with the modules XIOS386.CON and CDOS386.CON. This allows the building of either Concurrent XM or Concurrent 386 in the same sub-directory and avoids confusion.

2.3 THE GENSYS MAIN MENU

Figure 2-1 depicts the GENSYS Main Menu. Default values for an option are contained in brackets.

```
*** Concurrent DOS 6.0 GENSYS Main Menu ***
              help         GENSYS Help
           verbose  [Y]    More Verbose GENSYS Messages
         destdrive  [C:]   CCPM.SYS Output To (Destination) Drive
         deletesys  [N]    Delete (instead of rename) old CCPM.SYS file

         sysparams         Display/Change System Parameters
            memory         Display/Change Memory Allocation Partitions
       diskbuffers         Display/Change Disk Buffer Allocation
           oslabel         Display/Change Operating System Label
             drnet  [N]    Attaching Networking
              rsps         Display/Change RSP list
             phase         Display/Change Phase 1 Process list

            gensys         I'm finished changing things, go GEN a SYStem

Changes?
```

Figure 2-1. GENSYS Main Menu

The Main Menu items set general parameters (verbose, destdrive, drnet and deletesys), provide access to the submenus (sysparams, memory, diskbuffers, oslabel, phase, and rsps) and initiate system building (gensys).

The general parameters are defined as follows:

verbose — GENSYS displays submenu options either with or without an explanation. Set verbose to Y to get the explanations; set it to N for the minimum explanations.

destdrive — GENSYS writes the new CCPM.SYS file on the drive designated by destdrive. If you do not specify a destination drive, GENSYS uses the current drive.

deletesys — GENSYS either deletes or renames the existing CCPM.SYS file when it creates a new one. Set deletsys to Y to delete the old file; set it to N to rename it. GENSYS renames the previous copy of CCPM.SYS as CCPM.OLD.

drnet — GENSYS either attaches the NETSYS.CON module to the built CCPM.SYS or not.

The six submenus allow you to set system operating parameters as follows (number in parenthesis indicates section in which parameters are described):

sysparams (2.4)
Command response
FCB compatibility mode
Maximum memory per process
Maximum number of open files.per process
Maximum number of locked records per process
System's starting paragraph
Total number of open file and locked record entries
Number of Process Descriptors and Queue Control Blocks
Total size of queue buffer
Number of system flags

memory (2.5) Memory partitions and their addresses

diskbuffers (2.6)
Maximum number of buffers per process
Use of directory buffer hashing

oslabel (2.7) Virtual console sign-on message.

rsps (2.8) Resident System Processes (RSPs) to be included or excluded

phase (2.9) Specify phase order of the initial process starting.

The gensys option builds the system image file. If an input file is used to specify system parameters and the GENSYS command is not the last line in the input file, GENSYS uses its interactive mode to prompt you for any additional changes. See Section 2.11, "GENSYS Input Files," for more information.

Changes? is the Main Menu prompt. Terminate your entries by typing a carriage return at the Changes? prompt.

To change the three Main Menu options, enter the option name followed by the new value. You can use the following abbreviations:

v for verbose
des for destdrive
del for deletesys
dr for drnet

To access one or more of the submenus, enter the submenu name or names at the Changes? prompt. You can change options and list more than one submenu on a single line. For example, the following command:

```
Changes? v=y,des=c:,sysparams,oslabel,rsps,gensys
```

* sets the verbose mode
* sets the destination drive for the new system image file
* displays the sysparams, oslabel, and rsps submenus
* builds the new system image

The submenus are displayed in the order entered. Enter a response at the Changes? prompt to proceed to the next menu.

Type HELP at the prompt to display an explanation of Main Menu features and how to proceed.

The following sections describe the submenu parameters and how to construct an input file.

2.4 SYSTEM PARAMETERS MENU

The System Parameters Menu is shown in Figure 2-2. Type SYSPARAMS in response to the Main Menu Change? prompt to select the System Parameters menu. Note that all numeric values are in hexadecimal notation. The values in brackets are the default values. Table 2-1 describes the System Parameters menu items.

```
Display/Change System Parameters Menu
       sysdrive  [L:]    System Drive
       tmpdrive  [A:]    Temporary File Drive
     cmdlogging  [N]     Command Day/File Logging at Console
     compatmode  [Y]     CP/M FCB Compatibility Mode
        envsize  [#512]  Command Environment Size
        country  [#44]   Initial Country Code
         memmax  [4000]  Maximum Memory per Process (paragraphs)
         sysmem  [ 100]  Extra System Data Page Memory (Bytes)
        openmax  [20]    Open Files per Process Maximum
        lockmax  [20]    Locked Records per Process Maximum

        osstart  [1008]  Starting Paragraph of Operating System
       nhandles  [  80]  Number of File Handle Entries
     nopenfiles  [  40]  Number of Open Files
         nlocks  [  40]  Number of Locked Record Entries
        npdescs  [14]    Number of Process Descriptors
          nqcbs  [20]    Number of Queue Control Blocks
       qbufsize  [ 400]  Queue Buffer Total Size in bytes
         nflags  [20]    Number of System Flags

Changes?
```

Figure 2-2. GENSYS System Parameters Menu

Table 2-1. System Parameters Menu Options

Parameter	Description
SYSDRIVE	This is the default drive searched by Concurrent when it cannot locate a file in the current directory on a P_CLI call. This value is recorded in the SRCHDISK field of the SYSDAT DATA structure. The default value is L: which is the virtual drive used to load the current application.
TMPDRIVE	Select the temporary drive. The temporary drive is used for temporary disk files and should be the fastest drive in the system. This value is recorded in the TEMP DISK field in the SYSDAT DATA structure.

Table 2-1. (Cont'd)

Parameter	Description
CMDLOGGING	Set to Y to display the time and the name of the file executed after each user command entry. The information is displayed on a single line in the following form: `hh:mm:ss filename` A Y value also enters 0FFH in the DAY FILE field of the SYSDAT DATA structure.
COMPATMODE	Set to Y to have Concurrent use the CP/M compatibility attributes recorded with each file; specify N to have Concurrent ignore the file's compatibility attributes. See the Programmer's Guide for a description of the compatibility attributes.
ENVSIZE	Set the default size of the environment to be given to all loaded applications. This value becomes the value in the ENV_SIZE field of the SYSDAT DATA.
COUNTRY	Set the initial country code. 44 is the code for the UK, 01 is the code for USA. This value is placed in the COUNTRY field of the SYSDAT DATA.
MEMMAX	Set to the maximum paragraphs of memory you want allocated to each process. Although processes make their own memory allocations, this value represents an upper limit on how much memory each process can obtain. This value becomes the MMP value in the SYSDAT DATA structure.
OPENMAX	Set to the maximum number of open files you allow per process. The range for this value is 0 to 255 (0FFH). The value must be less than or equal to the total open files and locked records for the system (see the NOPENFILES parameter description below). This value becomes the OPEN_MAX value in SYSDAT DATA.
LOCKMAX	Set to the maximum number of locked records you allow per process. The range is 0 to 255 (0FFH), and the value must be less than or equal to the total open files and locked records for the system (see NOPENFILES parameter description below). This value is used for LOCK_MAX in SYSDAT DATA.
OSSTART	Set to the operating system's starting paragraph address. Code execution starts here with register CS set to this value, IP set to 0, and the Data Segment Register set to the SYSDAT segment address. This value is used for the CCPMSEG field in SYSDAT DATA.
NHANDLES	This value sets the total number of file handle entries in the system. 1 handle entry is required for each character device opened by a process, 2 handles are required for the first open of a file and 1 handle entry for each subsequent open of the same file (file sharing). Each handle entry occupies 2 bytes of SYSDAT DATA.

Table 2-1. (Cont'd)

Parameter	Description
NOPENFILES	This value is the total number of files that may be open in the system. Each entry in this table occupies 2 bytes of SYSDAT DATA.
NLOCKS	This entry sets the total number of allowed lock list and HDS items in the system. An HDS item is required for each sub- directory currently open on the system. A lock list item is required for each record currently locked on the system. The size of an item in this list is 10 bytes.
NPDESCS	Set to the number of Process Descriptors. A Process Descriptor is required for each process in the system, every RSP that extends 64K past the beginning of SYSDAT, and the first time a process changes directories. A Process Descriptor is 30H bytes long. Note that processes created by other processes (referred to as child processes) require their own Process Descriptor. See the description of the PUL (Process Unused List) field in Section 1.10, "SYSDAT DATA," for related information.
NQCBS	Set to the maximum number of Queue Control Blocks. Queues created by transient programs and those RSPs outside the 64K of the SYSDAT segment address each require a Queue Control Block. The Physical Keyboard Input (PIN) RSP also uses one Queue Control Block for each Virtual Input Queue (VINQ) it creates. The PIN RSP requires a VINQ for each of the system's virtual consoles. Determine a value for NQCBS by adding the number of virtual consoles in your system to the number of Queue Control Blocks you allocate for transient processes.
QBUFSIZE	Set to the size, in bytes, of the Queue Buffer Area. Allocate enough space for queues created by transient programs, RSPs outside the SYSDAT segment addess, and VINQs (see NQCBS, above). The amount of buffer area required for each queue is the message length times the number of messages. The PIN RSP creates a VINQ of 80H bytes for each virtual console in a system. Determine the size of the Queue Buffer Area by adding 80H bytes for each virtual console to the amount of space required for transient programs' queue messages. The maximum size of the Queue Buffer Area is 0FFFFH.
NFLAGS	Set to the maximum number of flags required by the system. This value is used for NFLAGS in SYSDAT DATA. The number of flags is dependent upon the design of the XIOS. Section 3.5 contains information about using flags for interrupt devices. See the descriptions of the DEV_SETFLAG and DEV_WAITFLAG system calls in the Programmer's Guide.

2.5 MEMORY ALLOCATION MENU

Figure 2-3 depicts the use of the Memory Allocation Menu. Initially, the menu indicates the current memory partitions. Add and delete partitions by entering ADD and DELETE commands at the Changes? prompt. The session shown in Figure 2-3 deletes all current partitions and creates 16K (4000H) partitions from address 2400:0 to 4000:0 and 32K (8000H) partitions from 4000:0 to 6000:0.

```
   Addresses        Partitions
#  Start  Last      Size   Qty
1. 400h   6000h     400h   17h

Display/Change Memory Allocation Partitions
         add   ADD memory partition(s)
      delete   DELETE memory partition(s)

Changes? delete=* add=2400,4000,400 add=4000,6000,800

   Addresses        Partitions
#  Start  Last      Size   Qty
1. 2400h  4000h     400h   7h
2. 4000h  6000h     800h   4h

Display/Change Memory Allocation Partitions
         add   ADD memory partition(s)
      delete   DELETE memory partition(s)

Changes? <cr>
```

Figure 2-3. GENSYS Memory Allocation Sample Session

The add and delete partition specifications consist of:

* the beginning address of the memory region to be partitioned
* the ending address of the memory region to be partitioned
* the size, in paragraphs, of the partitions

All values must be in hexadecimal notation and separated by commas. The Start and Last values on the menu are paragraph addresses; divide the absolute address by 10H to specify the paragraph address. Partition sizes are also in paragraphs; divide the actual partition size by 10H to enter the size parameter.

Use an asterisk to delete all memory partitions.

The memory partitions cannot overlap each other or the operating system area. GENSYS checks and trims memory partitions that overlap the operating system, but does not trim partitions that overlap each other. GENSYS displays an error message when the partitions overlap or are incorrectly sized and does not allow you to exit the Memory Allocation Menu if the memory partition list is not valid.

GENSYS does not check for partitions that refer to nonexistent system memory.

Concurrent allocates partitions in whole; a single partition is never divided among unrelated programs. If a memory request requires a memory segment that is larger than the available partitions, Concurrent concatenates adjoining partitions to form a single contiguous area of memory. The MEM module algorithm that determines the best fit for a given memory allocation request takes into account the number of partitions requested and the amount of partition space left over.

To determine the appropriate partitions, consider the applications to be run. Where small programs are run frequently and large programs only occasionally, divide memory into small partitions. This model simulates dynamic memory management and is very memory-efficient as long as memory does not become too fragmented. (When memory is too fragmented, the MEM module cannot find enough contiguous memory space for programs with large memory requirements even when enough memory is available.) Use the large partition model when large programs are run most frequently or programs are run serially. A good general-purpose compromise is to divide memory into 4K to 16K partitions.

2.6 DISK BUFFERING MENU

Figure 2-4 shows a sample Disk Buffering Menu session. This allows the default number of disk buffers required on the system to be set. INIT will extend the number of buffers if the CCONFIG.SYS file requests more buffers than have already been set up. The XIOS can optionally decide to set the default buffers itself. If this is the case a flag in the XIOS header is set to indicate to GENSYS that no default buffers are required. The minimum number of buffers required for the system to "come up" is 3. The number of directory entries to be hashed is also set by this menu. Again this may be allocated by the XIOS and it flags in the XIOS header if no hash entries are to be allocated by GENSYS.

Make changes by entering responses to the prompts displayed by GENSYS.

```
      *** Disk Buffering Information ***
  nobuffers   [ 3]    The number of default disk buffers to be allocated
hashentries   [ 200]  The number of hashed directory entries
 buffersize   [ 200]  Maximum physical sector size
clustersize   [0800]  Maximum cluster size
```

Figure 2-4. Disk Buffering Menu Session

2.7 OSLABEL MENU

Figure 2-5 shows the OSLABEL Menu. Select this menu to write a message for display on all virtual consoles when the system is loaded.

```
Display/Change Operating System Label
Current message is:
<null>

Add lines to message. Terminate by entering only RETURN:
```

Figure 2-5. GENSYS Operating System Label Menu

Enter the message using spaces to format the display. Terminate each line with a carriage return. To end the message, enter a carriage return at the beginning of the line. Use the $ character to terminate the display message but continue with comments. Note that where the XIOS prints its own sign-on message, the XIOS message appears before the message specified in the GENSYS OSLABEL Menu.

2.8 RSP LIST MENU

Figure 2-6 shows a session with the GENSYS RSP (Resident System Process) List Menu. GENSYS gets the initial list of RSPs by scanning the current drive's directory. All files with an extension of RSP are listed. The following session excludes MY.RSP and COM.RSP:

```
RSPs to be included are:

   PIN.RSP     TMP.RSP      CLOCK.RSP   MY.RSP
   COM.RSP

Display/Change RSP List

   include     Include RSPs
   exclude     Exclude RSPs

Changes? ex=com.rsp,my.rsp

RSPs to be included are:

   PIN.RSP     TMP.RSP      CLOCK.RSP

Changes? <cr>
```

Figure 2-6. GENSYS RSP List Menu Sample Session

GENSYS includes all RSPs in the current directory unless instructed to do otherwise. Use the wildcard *.* with include or exclude to designate all RSPs. Note that you can abbreviate include to "in" and exclude to "ex".
Terminate the commands with a carriage return. End the RSP List Menu session by entering a carriage return in response to the Changes? prompt.

Important: You must include PIN.RSP, CLOCK.RSP, and TMP.RSP in the system image. (In Concurrent 386 with DOS terminal support you must include FLUSH.RSP).

2.9 PHASE OPTION

Select the RSPs to be started in the first phase. This gives priority to processes that need to run before the rest of the RSPs.

2.10 GENSYS OPTION

Select the gensys option to build the system image. GENSYS displays the messages shown in Figure 2-7 during system generation.

```
Generating new SYS file
Appending RSPs to SYS file
Generating tables
Doing Fixups
SYS image load map:
          Code starts at GGGG
      Last RSP starts at JJJJ
          Data starts at HHHH
         Tables start at IIII
   XIOS Buffers start at KKKK
           End of OS at LLLL
Trimming memory partitions. New List:

   Addresses           Partitions (in paragraphs)
#  Start     Last      Size          Qty
1. AAAAh     BBBBh     XXXXh         Yh
2. MMMMh     NNNNh     QQQQh         Vh
Wrapping up
```

Figure 2-7. GENSYS System Generation Messages

2.11 GENSYS INPUT FILES

GENSYS accepts system generation commands from an input file and allows you to redirect console output to a disk file. To use these features, invoke GENSYS as follows:

```
A>GENSYS < filein > fileout
```

filein represents the name of the input file; fileout the name of the file in which console output is recorded.

In the input file, enter each command on a separate line, followed by a carriage return, in the exact sequence required during a manually operated GENSYS session. Be sure to include the carriage return that ends the session with each menu to proceed with the next. Use a semicolon to make comments.

Enter GENSYS as the last command in your input file to end the command sequence and generate the system. If the GENSYS command is not present, GENSYS queries the console for your changes.

Listing 2-1 demonstrates a GENSYS input file. Assuming that the name of the input file is GENSYS.IN, use the following command to invoke GENSYS:

```
A>GENSYS < GENSYS.IN [parameter list]
```

Note: The parameters in the list are substituted for the %1, %2 etc in the input file.

Listing 2-1. Example GENSYS Input File

```
OSLABEL
----------------------------
   Concurrent DOS %1a
   %2a
   %2d  %8t
----------------------------


SYSPARAMS
sysdrive=L: cmdlogging=off osstart=60 nflags=7f qbuf=c00
nopenfiles=80 mem=8000 npd=30 sysmem=#512

MEMORY
delete=* add=2000,A000,400

RSPS
ex=*.*
in=tmp.rsp,pin.rsp,clock.rsp

PHASE
del=*.* add=TMP,PIN0,PIN1,PIN2,CLOCK

GENSYS
```

XIOS OVERVIEW

All Concurrent DOS 86 hardware dependencies are concentrated in subroutines collectively referred to as the Extended Input/Output System, or XIOS. You can modify these subroutines to tailor the system to almost any disk-based operating environment. This section provides an overview of the XIOS and describes some important variables and tables referenced within it. While reading this section, you should refer frequently to the XIOS examples. The XIOS examples are stored in source code form on the Concurrent distribution disk.

The XIOS is a small model (separate code and data). Concurrent accesses the XIOS through two entry points in the code segment INIT (offset 0H) and ENTRY (offset 3H). The INIT entry point is for system hardware initialization only; ENTRY is for all other XIOS functions. Because all operating system routines use a CALL FAR instruction to access the XIOS through these two entry points, the XIOS function routines must end with a RETURN FAR instruction. Subsequent sections describe the XIOS entry points and other fixed data fields.

Note: Programs that depend upon the interface to the XIOS must check the version number of the operating system before trying direct access to the XIOS. Future versions of Concurrent can have different XIOS interfaces, including changes to XIOS function numbers and/or parameters passed to XIOS routines. In the sample XIOS, for compatibility with previous OEM utilities that made a direct call to 0c03h in the data segment, we have added some code to allow access to the XIOS entry point in the code segment.

3.1 XIOS HEADER

The XIOS Header contains variables that GENSYS uses when constructing the CCPM.SYS file and that the operating system uses when executing. Figure 3-1 shows the XIOS Header format. Table 3-1, immediately following the figure, defines the XIOS Header fields.

CODE SEGMENT header fields

000H	JMP INIT			JMP ENTRY			SYSDAT	
008H	SUPERVISOR							

DATA SEGMENT header fields

C00H	RESERVED			RESERVED			RESERVED	
C08H	RESERVED				TICK	T/SEC	DOOR	NACB
C10H	NPCNS	NVCNS	NCCB	NLCB	CCB		LCB	
C18H	DPH(A)		DPH(B)		DPH(C)		DPH(D)	
C20H	DPH(E)		DPH(F)		DPH(G)		DPH(H)	
C28H	DPH(I)		DPH(J)		DPH(K)		DPH(L)	
C30H	DPH(M)		DPH(N)		DPH(O)		DPH(P)	
C38H	ALLOC		ACB		OEMPTR		CCBLIST	
C40H	LCBLIST		ACBLIST		FFLAG			

Figure 3-1. XIOS Header

Table 3-1. XIOS Header Fields

Field	Explanation
INIT XIOS	Initialization Point. At system boot, the Supervisor module executes a CALL FAR instruction to this location in the XIOS (XIOS Code Segment: 000H). This call transfers control to the XIOS INIT routine, which initializes the XIOS and hardware, then executes a RETURN FAR instruction.
JMP ENTRY	XIOS Entry Point. All access to the XIOS functions goes through the XIOS Entry Point. The operating system executes a CALL FAR (CALLF) to this location in the XIOS (XIOS Code Segment: 003H) whenever I/O is needed. This instruction transfers control to the XIOS ENTRY routine, which calls the appropriate function within the XIOS. Once the function is complete, the ENTRY routine executes a RETURN FAR (RETF) instruction to the operating system.
SYSDAT	This field is initialized by GENSYS to contain the segment address of SYSDAT. This address in the XIOS Code Segment gives processes in interrupt routines and other areas of code where the Data Segment is unknown access to SYSDAT data. Listing 3-1, immediately following this table, contains an example routine that accesses the current process's Process Descriptor using the SYSDAT segment address.

Table 3-1. (Cont'd)

Field	Explanation
SUPERVISOR FAR	Address (double-word pointer) of the Supervisor Module entry point. Whenever the XIOS makes a system call, it must access the operating system through this entry point in the code segment. GENSYS initializes this field. Section 3.8 describes XIOS register usage and restrictions.
TICK	Set Tick Flag (Boolean value). The Timer Interrupt routine uses this variable to determine whether the DEV_SETFLAG system call should be used to set the TICK_FLAG. Initialize this variable to zero (00H) in the XIOS.CON file. Concurrent sets this field to 0FFH whenever a process is delaying. The field is reset to zero (00H) when all processes finish delaying. See the Programmer's Guide for details on the DEV_SETFLAG and P_DELAY system calls.
TICKS_SEC	Number of Ticks per Second. In the XIOS.CON file, this field must be initialized to the number of ticks that make up one second as implemented by the particular XIOS. GENSYS copies this field into the SYSDAT DATA. Application programmers can use TICKS_SEC to determine how many ticks to allow for a delay of one second. See Section 8, "XIOS Tick Interrupt Routine," for more information.
DOOR	Global Door Open Interrupt Flag. This field must be set to 0FFH by the drive door open interrupt handler routine if the XIOS detects that any drive door has been opened. The BDOS checks this field before every disk operation to verify that the media is unchanged. If a door has been opened, the XIOS must also set the Media Flag in the DPH associated with the drive.
NACB	Number of Auxiliary Control Blocks. Initialize this field to the number of auxiliary devices supported by the XIOS. Auxiliary devices can be used for serial communication. GENSYS copies NACB into the NACB field of the SYSDAT DATA.
NPCNS	Number of Physical Consoles. Initialize this field to the number of physical consoles, or user terminals, connected to the system. This number does not include extra I/O devices. GENSYS uses this value to create a PIN process for each physical console. It also copies NPCNS into the XPCNS field of the SYSDAT DATA.
NVCNS	Number of Virtual Consoles. Initialize this field to the number of Virtual Consoles supported by the XIOS in the XIOS.CON file. GENSYS creates a TMP process for each Virtual Console. GENSYS copies NVCNS into the NVCNS field of the SYSDAT DATA. See Section 4 for more information.
NCCB	Number of Logical Consoles. Initialize this field to the number of Virtual Consoles plus the number of character I/O devices supported by the XIOS. Character I/O devices are devices accessed through the console system calls of Concurrent (functions whose mnemonic begins with C_) but whose console numbers are beyond the range of the virtual consoles. Application programs access the character I/O devices by setting their default console number to the character I/O device's console number and using Concurrent's regular console system calls. See Section 4 in this guide and the C_SET system call description in the Programmer's Guide. GENSYS copies this field into the NCCB field of the SYSDAT DATA.

Table 3-1. (Cont'd)

Field	Explanation
NLCB	Number of List Control Blocks. Initialize this field in the XIOS.CON file to equal the number of list devices supported by the XIOS. A list device is an output-only device, typically a printer. GENSYS copies this field into the NLCB field of the SYSDAT DATA.
CCB	Offset of the Console Control Block Table. Initialize this field in the XIOS.CON file to the address of the CCB Table in the XIOS. A CCB Entry in the Table must exist for each of the consoles indicated in NCCB. Each entry in the CCB Table must be initialized as described in Section 4.1, "Console Control Block". GENSYS copies this field into the CCB field of the SYSDAT DATA.
LCB	Offset of the List Control Block. This field is initialized in the XIOS.CON file to the address of the LCB Table in the XIOS. There must be an LCB Entry for each of the list devices indicated in NLST. Each entry must be initialized as described in Section 4.4, "List Device Functions." GENSYS copies this field into the LCB field of the SYSDAT DATA.
DPH(A)-DPH(P)	Offset of initial Disk Parameter Header (DPH) for drives A through K, respectively. L:, N:, O: and P: are used by the system and are not available for use as a physical drive. It is mandatory that unused DPH entries are set to 0 and that non-zero entries reference the DPHs of the selected drive.
ALLOC	This value is initialized in the XIOS to the size, in paragraphs, of an uninitialized RAM buffer area to be reserved for the XIOS by GENSYS. GENSYS sets this field in the CCPM.SYS file to the starting paragraph (segment value) of the XIOS uninitialized buffer area. This value may then be used by the XIOS for based or indexed addressing into the buffer area. Typically, the XIOS uses this buffer area for the Virtual Console screen maps, programmable function key buffers, and nondisk-related I/O buffering. If the ALLOC field is initialized to zero in the XIOS.CON file, GENSYS allocates no buffer RAM and leaves ALLOC set to zero in the system image.
ACB	Offset of the first Auxiliary Control Block. Initialize this field to the address of the first ACB. The number of ACBs must be as specified in the NACB field. GENSYS copies this field into the ACB field of SYSDAT. See Section 4.5, "Auxiliary Device Functions."
OEMPTR	This word is NOT used by Concurrent and has been reserved for the OEM to use as a pointer to OEM specific data. By using this pointer further extensions to the XIOS header will not affect OEM utilities that require access to data held in SYSDAT.
CCBLIST	This WORD gives the offset of an NVCNS entry array of pointers to CCBs. Therefore in a 6 virtual console system an array of 6 pointers is required.
LCBLIST	This WORD gives the offset of an NLCB entry array of pointers to LCBs.
ACBLIST	This WORD gives the offset of an NACB entry array of pointers to ACBs.

Table 3-1. (Cont'd)

Field	Explanation

FFLAG	This byte specifies the first flag that can be allocated by the DEV_FLAGALLOC function. All flags from 0 to FFLAG-1 are used by the system or the XIOS. FFLAG can be used by the XIOS for either dynamic or static flag allocation.
	Static allocation.
	Initialise the FFLAG variable to the number of flags required by the XIOS + 2 (Flags 0 and 1 are reserved). This reserves all the XIOS flags.
	Dynamic allocation.
	If the XIOS is self configuring the number of flags required may vary considerably. The following use of the FFLAG variable will optimise flag usage.
	`FFLAG          db    2         ; reserve the system flags`
	`mov dl, FFLAG                   ; get the next available flag`
	`inc FFLAG                       ; and increment count`

Listing 3-1. Process Descriptor Access Routine

```
DSEG
RLR         EQU WORD PTR .68H
                                ; RLR field of SYSDAT
            CSEG                ; of XIOS
            PUSHDS              ; Save XIOS Data Segment
            MOV DS,CS:SYSDAT    ; Move SYSDAT segment address
                                ; into DS
            MOV BX,RLR          ; Move the current process's PD
                 .              ; address into BX and perform
                 .              ; and perform operation. (See
                 .              ; Section 1.10 for an
                                ; explanation of RLR)
            POP DS              ; Restore the XIOS
                                ; Data Segment
```

Listing 3-2 is an assembly language definition of the XIOS Header.

Listing 3-2. XIOS HEADER Definition

```
;*************************************************
;*
;*      XIOS Header Definition
;*
;*************************************************

        CSEG
        org     0h

        jmp init            ;system initialization
        jmp entry           ;xios entry point

 sysdat         dw      0       ;Sysdat Segment

 supervisor     rw      2

        DSEG
        org     0c0ch

 tick           dw      false           ;tick enable flag
 ticks_sec      db      60              ;ticks per second
 door           db      0               ;global drive door
                                        ;open interrupt flag
 nacb           db      1               ;# of aux devices
 npcns          db      4               ;# of phys consoles
 nvcns          db      8               ;# of VCs
 nccb           db      8               ;total ccbs
 nlst           db      1               ;# of list devices
 ccb            dw      offset ccb0     ;offset of 1st ccb
 lcb            dw      offset lcb0     ;offset of 1st lcb

                        ;disk parameter header offset table
 dph_tbl        dw      offset dph0     ;drive A:
                dw      offset dph1     ;B:
                dw      0,0,0           ;C:,D:,E:
                dw      0,0,0           ;F:,G:,H:
                dw      0,0,0           ;I:,J:,K:
                dw      0               ;L:
                dw      offset dph2     ;M:
                dw      0,0,0           ;N:,O:,P: reserved
                                        ;for system use
 alloc          dw      0
 acb            dw      offset acb0     ;offset of first acb
 oemptr         dw      0               ; not used
 ccblist        dw      offset ccbptrs  ; pointer array to ccb's
 lcblist        dw      offset lcbptrs  ; pointer array to lcb's
 acblist        dw      offset acbptrs  ; pointer array to acb's
 fflag          db      2               ; 2 reserved

;-----------------------------------------------------
```

3.2 INIT ENTRY POINT

As shown in Listing 3-2, the entry point to the XIOS Initialization routine, INIT, is at offset 0H from the beginning of the XIOS code module. The INIT process calls the XIOS Initialization routine during system initialization.

This section describes the events that begin from the time CCPM.SYS is loaded into memory until the RSPs are created.

1. The loader loads CCPM.SYS into memory at the absolute Code Segment location contained in the CCPM.SYS file Header and initializes the CS and DS registers to the Supervisor code segment and the SYSDAT, respectively. At this point, the loader executes a JMPF to offset 0 of the CCPM.SYS code and begins the SUP module's initialization code as described below.

2. The first step of SUP initialization is to set up the INIT process. The INIT process performs the rest of system initialization at a priority equal to 1.

3. The INIT process calls the initialization routines of each of the other modules with a CALL FAR instruction. The first instruction of each code module is assumed to be a JMP instruction to its initialization routine. The XIOS initialization routine is the last of these modules called. Once this call is made, the XIOS initialization code is never used again. Thus, it can be located in a directory buffer or other uninitialized data area.

4. As shown in the example XIOS listing, the initialization routine must initialize all hardware and interrupt vectors.

5. The XIOS initialization routine can optionally print a message to the console before it executes a RETURN FAR (RETF) instruction upon completion. Note that each TMP prints out the string addressed by the VERSION variable in the SYSDAT DATA. You can change this string through the OSLABEL Menu in GENSYS.

6. Upon return from the XIOS, the SUP Initialization routine, running under the INIT process, creates some queues and starts up the RSPs in the order dictated by the PHASE command used in GENSYS.

7. Once this is done, the INIT process terminates.

The XIOS INIT routine should initialize all unused interrupts to vector to an interrupt trap routine that prevents spurious interrupts from vectoring to an unknown location. The example XIOS handles uninitialized interrupts by printing the name of the process that caused the interrupt followed by an uninitialized interrupt error message. Then the interrupting process is unconditionally terminated.

Concurrent saves Interrupt Vector 224 prior to system initialization and restores it following execution of the XIOS INIT routine. However, it does not store or alter the Non-Maskable Interrupt (NMI) vector, INT 2. Setting NMI is also the responsibility of the XIOS. The example XIOS first initializes all the Interrupt Vectors to the uninitialized interrupt trap, then initializes specifically used interrupts.

3.3 XIOS ENTRY

All access to the XIOS after initialization passes through the ENTRY routine. The entry point for this routine is at offset 03H from the beginning of the XIOS code module. The operating system accesses the ENTRY routine with a CALL FAR to the location offset 03H bytes from the beginning of the SYSDAT Segment. When the XIOS function is complete, the ENTRY routine returns by executing a RETURN FAR instruction, as in the example XIOS's. On entry, the AL register contains the function number of the routine being accessed, and registers CX and DX contain arguments passed to that routine. The XIOS must maintain all segment registers through the call. This means that the CS, DS, ES, SS, and SP registers must be preserved.

All the XIOS functions except the disk functions use the register conventions shown in Table 3-2.

Table 3-2. XIOS Register Usage

	Register Content	
On Entry:	AL	Function Number
	AH	Undefined
	BX	PC Mode Parameter
	CX	First Parameter
	DX	Second Parameter
	DS	SYSDAT Segment
	ES	User Data Area
	SI, DI, and BP are undefined	
On Return:	AX	Return or XIOS Error Code
	BX	Equal to AX
	DS	SYSDAT Segment
	ES	User Data Area
	SI, DI, BP, DX, and CX are undefined	

The segment registers (DS and ES) must be preserved through the ENTRY routine. When calling the SUP from within the XIOS, however, the ES Register must equal the UDA of the running process and DS must equal the System Data Segment. Thus, if the XIOS is going to perform a string move or other code using the ES register, it must preserve ES. Alternativly the ES and DS registers must be specifically set to the UDA and SYSDAT before the SUP is called or the function returns.

In the example XIOS's, the XIOS function routines are accessed through a function table where the function number is the actual table entry. Table 3-3 lists the XIOS function numbers and the corresponding XIOS routines; detailed explanations of the functions appear in the referenced sections of this document. Listing 3-3 is an example XIOS ENTRY Jump Table.

Table 3-3. XIOS Functions

Number	XIOS Routine	
	Console Functions -- Section 4.2	
0	IO_CONST	CONSOLE INPUT STATUS
1	IO_CONIN	CONSOLE INPUT
2	IO_CONOUT	CONSOLE OUTPUT
7	IO_SWITCH	SWITCH SCREEN
8	IO_STATLINE	DISPLAY STATUS LINE
48	IO_OPEN	CHARACTER DEVICE OPEN
49	IO_CLOSE	CHARACTER DEVICE CLOSE
	Window Functions -- Section 4.3.1	
16	WW_POINTER	GET WINDOW DATA
17	WW_KEY	GET KEYBOARD INPUT/STATUS
18	WW_STATLINE	DISPLAY STATUS LINE
19	WW_IM_HERE	SWITCH WINDOW
20	WW_NEW_WINDOW	DEFINE WINDOW
21	WW_CURSOR_VIEW	SET CURSOR
22	WW_WRAP_COLUMN	SET SCREEN WIDTH
23	WW_FULL_WINDOW	SWITCH WINDOW SIZE
24	WW_SWITCH_DISPLAY	SWITCH MONITOR
	List Device Functions -- Section 4.4	
3	IO_LSTST	LIST STATUS
4	IO_LSTOUT	LIST OUTPUT
	Auxiliary Device Support -- Section 4.5	
5	IO_AUXIN	AUXILIARY INPUT
6	IO_AUXOUT	AUXILIARY OUTPUT
14	IO_POINTER	GET OFFSET OF MACHINE-SPECIFIC DATA
37	IO_AUXSTIN	AUXILIARY STATUS INPUT
38	IO_AUXSTOUT	AUXILIARY STATUS OUTPUT
	Device Block Read/Write Function -- Section 4.6	
39	IO_DEVIO	BLOCK READ/WRITE
	Poll Device Function -- Section 4.7	
13	IO_POLL	POLL DEVICE

Table 3-3. (Cont'd)

Number	XIOS Routine	
System Space Allocation Function -- Section 5.9		
28	SYSDAT_ALLOC	ALLOCATE SYSDAT SPACE
44	IO_GET_HISTORY_BUFF	FOR COMMAND LINE RECALL
45	IO_GET_VECTORS	GET VECTOR TABLE
46	MEM_ALLOC	ALLOCATE SEGMENT_SPACE (CDOS 386)
50	IO_PROTECT	PROTECT MODE INITIALISATION (CDOS 386)
51	IO_FLUSH_IO	FLUSH (386 ONLY)
Disk Functions -- Section 5		
9	IO_SELDSK	SELECT DISK
10	IO_READ	READ DISK
11	IO_WRITE	WRITE DISK
12	IO_FLUSH	FLUSH BUFFERS
15	IO_FORMAT	FORMAT CP/M DISK
29	IO_NEW_FORMAT	VARIABLE DISK FORMAT
47	IO_VERIFY	WRITE WITH VERIFY
ROS support Functions -- Section 7		
30	GET_SET_SCREEN	(optional) ALPHA/GRAPHICS SCREEN
31	PC_VIDEO	(optional) VIDEO IO
32	PC_KBD	KEYBOARD MODE SWITCH
33	PC_SHIFTS	RETURN SHIFT STATUS
34	PC_EQUIP	EQUIPMENT CHECK
35	IO_INT13	DOS INT 13 DISK I/O
Expanded Memory Support Functions -- Section 6		
40	IO_MPALLOC	ALLOCATE PAGED MEMORY
41	IO_MPFREE	RELEASE PAGED MEMORY
42	IO_MPSAVE	SAVE HARDWARE STATE
43	IO_MPRESTORE	RESTORE HARDWARE STATE

Listing 3-3. XIOS Function Table

```
;---------------------------------------------------------
; XIOS FUNCTION TABLE
;---------------------------------------------------------

 function_table:
 ; Functions required by the kernel:
     dw      io_const                  ; 0 console status
     dw      io_conin                  ; 1 console input
     dw      io_conout                 ; 2 console output
     dw      io_listst                 ; 3 list status
     dw      io_list                   ; 4 list output
     dw      io_auxin                  ; 5 auxiliary input
     dw      io_auxout                 ; 6 auxiliary out
     dw      io_switch                 ; 7 switch screen
     dw      io_statline               ; 8 update/print new status
     dw      io_seldsk                 ; 9 select disk
     dw      io_read                   ; 10 read logical sector
     dw      io_write                  ; 11 write logical sector
     dw      io_flushbuf               ; 12 flush buffers
     dw      io_poll                   ; 13 poll device

 ; Misc functions used by system-specific utilities:
     dw      io_pointer                ; 14 general pointer return
                                       : for machine specific data
     dw      io_format                 ; 15 CP/M only format entry
                                       ; This is not a kernel function

 ; Back door window control entry points and misc:
 ; (Optional functions unused by the kernel)
     dw      ww_pointer                ; 16 return data pointers
     dw      ww_key                    ; 17 wait for a key
     dw      ww_statline               ; 18 char/attib status line
     dw      ww_im_here                ; 19 window process state
     dw      ww_new_window             ; 20 redefine a window
     dw      ww_cursor_view            ; 21 track mode, viewpoint
     dw      ww_wrap_column            ; 22 set wrap around column
     dw      ww_full_window            ; 23 same as full key
     dw      ww_switch_display         ; 24 mono/color monitors
     dw      io_ret                    ; 25 dummy return
     dw      io_ret                    ; 26 dummy return
     dw      io_ret                    ; 27 dummy return

 ; Misc functions used by system-specific utilities:
     dw      sysdat_alloc              ; 28 allocate sysdat space
     dw      io_new_format             ; 29 variable format entry
```

Listing 3-3. (Cont'd)

```
 ; PC Mode ROM BIOS entry point emulators:
 ; (Functions required only for DOS support)
     dw      get_set_screen              ; 30 alpha/graphics screens
     dw      pc_video                    ; 31 ROM BIOS video IO
     dw      pc_kbd                      ; 32 ROM BIOS keyboard switch
     dw      pc_shifts                   ; 33 ROM BIOS style shift status
     dw      pc_equip                    ; 34 ROM BIOS style equip word
     dw      io_int13                    ; 35 ROM BIOS disk I/O/Format
     dw      io_ret                      ; 36 dummy return

 ; Auxiliary device support:
 ; (Functions required by the kernel)
     dw      io_auxin_stat               ; 37 aux input status
     dw      io_auxout_stat              ; 38 aux output status
     dw      io_devio                    ; 39 device block read/write

 ; Expanded memory support functions:
 (Functions required by the kernel in XM)
     dw      io_mpalloc                  ; 40 allocate paged memory
     dw      io_mpfree                   ; 41 free paged memory
     dw      io_mpsave                   ; 42 save paged mem hardware
     dw      io_mprestore                ; 43 restore saved hardware

 ;
 ; (Functions required by the kernel)
     dw      io_get_history_buffer       ; 44 for command line recall
     dw      io_get_vector               ; 45 get interrupt vector table
     dw      mem_alloc                   ; 46 allocate system space (386 only)
     dw      io_write_vfy                ; 47 write with verify
     dw      io_open                     ; 48 open a character device
     dw      io_close                    ; 49 close a character device
     dw      io_protect                  ;  50  protected  mode  initialise  (386
                                         only)
     dw      io_flush_io                 ; 51 dos terminal flush (386 only)

;-------------------------------------------------------------
```

3.4 POLLED DEVICES

Polled I/O device drivers in single tasking systems typically execute a small compute-bound instruction loop waiting for a ready status from the I/O device. This causes the driver routine to spend a significant portion of CPU execution time looping. To allow other processes use of the CPU resource during hardware wait periods, the Concurrent XIOS must use the DEV_POLL system call to place the polling process on the Poll List. After the DEV_POLL call, the dispatcher stops the process and calls the XIOS IO_POLL function every dispatch until IO_POLL indicates the hardware is ready. The dispatcher then restores the polling process to execution and the process returns from the DEV_POLL call. Since the process calling the DEV_POLL function does not remain in ready state, the CPU resource becomes available to other processes until the I/O hardware is ready.

When polling, a process executing an XIOS function calls the Concurrent DEV_POLL system call with a poll device number. The dispatcher then calls the XIOS IO_POLL function with the same poll device number. The example XIOS uses the poll device number as an index into a table of poll routine entry points. It then calls the appropriate poll function and returns the I/O device status to the dispatcher.

3.5 INTERRUPT DEVICES

As in the case of polled I/O devices, an XIOS driver handling an interrupt-driven I/O device should not execute a wait loop or halt instruction while waiting for an interrupt to occur.

The Concurrent XIOS handles interrupt-driven devices with DEV_WAITFLAG and DEV_SETFLAG system calls. A process that needs to wait for an interrupt to occur makes a DEV_WAITFLAG system call with a flag number. The system stops this process until the desired XIOS interrupt handler routine makes a DEV_SETFLAG system call with the same flag number. The waiting process then continues execution. The interrupt handler follows the steps outlined below, executing a JUMP FAR (JMPF) to the Dispatcher entry point. The interrupt handler can also perform an IRET instruction when it needs to return, but jumping directly to the Dispatcher provides a faster response to the process waiting on the flag and is logically equivalent to the IRET instruction.

If interrupts are enabled within an interrupt routine, a Tick interrupt can cause the interrupt handler to be dispatched. This dispatch could make interrupt response time unacceptable. To avoid this situation, do not re-enable interrupts within the interrupt handlers or only jump to the Dispatcher when not in another interrupt handler routine.

Because of machine architecture differences, Concurrent interrupt handlers differ from those in an 8080, 8085, or Z-80 environment. Study the example XIOS Tick interrupt handler carefully. During initial debugging, do not implement interrupts until after the system works in a polled environment.

An XIOS interrupt handler routine must perform the following basic steps:

1. Do a stack switch to a local stack. The interrupted process might not have enough stack space for a context save.

2. Save the register environment of the interrupted process, or at least the registers that will be used by the interrupt routine. Usually the registers are saved on the local stack established by the first step.

3. Satisfy the interrupting condition. This can include resetting the hardware and performing a DEV_SETFLAG system call to notify a process that the interrupt for which it was waiting has occurred.

4. Restore the interrupted process's register environment.

5. Switch back to the original stack.

6. Return from the interrupt routine with either a JUMP FAR (JMPF) to the dispatcher or an Interrupt Return (IRET). When interrupts are not re-enabled within the interrupt handler, a JMPF to the dispatcher is executed on each system tick and after a DEV_SETFLAG call is made. If interrupts are re-enabled, an IRET instruction is executed.

Note: DEV_SETFLAG is the only Concurrent system call an interrupt routine can use. This is because DEV_SETFLAG is the only system call the operating system assumes has no process context associated with it. DEV_SETFLAG must enter the operating system through the SUP entry point at SYSDAT:0000H and cannot use INT 224.

3.6 Numeric Data Processor EXCEPTION HANDLER

This section explains how to provide support for the NDP chip. This is required to allow more than one process to use the NDP without conflict and to handle the NDP exception interrupt. The following discussion applies to the 8087, 80287 and 80387 Numeric Data Processor (NDP) from Intel. The XIOS initialization code must initialize five fields in the SYSDAT area to support the NDP. The XIOS must also contain a default exception handler to handle any interrupts from the NDP. The system is structured so that a programmer can write an individual exception handler for the NDP.

The XIOS initialization code must first check for the presence of the NDP chip by using the FNINIT instruction. If the NDP is present, the XIOS initialization code must set up the following SYSDAT fields:

SEG_8087 Must be set to the segment and
OFF_8087 offset of the NDP interrupt vector.

SYS_87_SG Must be set to the segment and
SYS_87_OF offset of the XIOS default exception handler.

OWNER_8087 Must be set to 0 to indicate that there is an NDP present in the system. The default value is FFFFH which indicates no NDP. FFFFH is put in this field by the SUP initialization code.

The NDP interrupt vector itself must also be set to the segment and offset of the XIOS default exception handler.

To guarantee program integrity in a multitasking environment, the NDP exception handler must perform its functions in a specific order. The following steps outline the primary functions of the example default NDP exception handler (see Listing 3-4):

1. Save the Processor environment.

2. Save the NDP environment.

3. Clear the NDP IR (status word).

4. Disable NDP interrupts.

5. Acknowledge the interrupt (hardware dependent).

6. Read the OWNER_8087 field in SYSDAT and perform the desired action. Note that processor interrupts are currently disabled. Do not perform any action that would turn them back on yet. The default exception handler uses the OWNER_8087 field to terminate the process on a severe error.

7. Restore the processor environment.

8. Restore the NDP environment with clear status. This enables the NDP interrupts.

9. Execute an IRET instruction to return and enable the processor interrupts.

If the NDP environment is not restored before processor interrupts are enabled and an interrupt occurs (for example, Tick), a different NDP process can gain control of the NDP and swap in its NDP context. On a second interrupt, or on an IRET instruction, the process that happened to be executing the exception handler code will be brought back into context and will write over the new NDP context.

All NDP processes are initialized by the system with the address of the default exception handler. If a process wants to use its own exception handler, it must initially overwrite the NDP interrupt vector with the address of its own exception handler. On each context switch, the NDP interrupt vector is saved and restored as part of the NDP process's environment.

The hardware-dependent address of the NDP interrupt vector is provided in the SEG_8087 and OFF_8087 fields of the system data area.

An individual exception handler that does not follow the sequence of events described for the default handler will have unpredictable results on the system. If possible, make this default interrupt handler re-entrant.

Listing 3-4. NDP Exception Handler

```
;=========================================================
; NDP Default Exception Handler
;=========================================================

; This is the example default exception handler. It is assumed that if the
; NDP programmer has enabled NDP interrupts and has specified exception flags in
; the control word, that the programmer has also included an exception handler to
; take specific actions in response to these conditions. This handler ignores
; non-severe errors (overflow, etc.) and terminates processes with severe errors
; (divide by zero, stack violation).

push        ds                         ;Save current data seg
mov         ds,sysdat                  ; Get XIOS data seg
mov         ndp_ssreg,ss               ; Stack switch for processor
mov         ndp_spreg,sp               ; environment
mov         ss,sysdat
mov         sp,offset ndp_tos          ; Save processor registers
push        ax
push        bx
push        cx
push        dx
push        di
push        si
push        bp
push        es
mov         es,sysdat                  ; Now save NDP env
FNSTENV     env_8087                   ; Save NDP Process Info
FWAIT
FNCLEX                                 ; Clear NDP interrupt
xor         ax,ax
FNDISI                                 ; Disable NDP interrupts

mov         al,020h                    ; Send int ack's - 1 for slave
out         060h,al
mov         al,020h                    ; - 1 for master PIC
out         058h,al
call        in_8087                    ; Check NDP error condition
                                       ; if error is severe,
                                       ; process will abort

mov         bx,offset env_8087         ; clear NDP status word
mov         byte ptr 2[bx],0           ; for env restore
pop         es                         ; Restore processor env.
pop         bp
pop         si
pop         di
pop         dx
pop         cx
pop         bx
pop         ax
mov         ss,ndp_ssreg               ; Switch to previous stack
mov         sp,ndp_spreg
```

Listing 3-4. (Cont'd)

```
FLDENV      env_8087                  ; Restore NDP env
FWAIT                                 ; with good status
pop         ds                        ; Restore previous data seg
iret

in_8087:

mov         bx,owner_8087             ; Get the PD
test        bx,bx                     ; Check if owner has
jz          end_87                    ; already terminated
mov         si,offset env_8087        ; If severe error, terminate
mov         ax,statusw[si]            ; If not, return and continue
test        ax,03ah                   ; 3A = under/overflow, precision,
jnz         end_87                    ; and denormalized operand
or          p_flag[bx],080h           ; Must be zero divide or invalid
                                      ; operation (stack error)
                                      ; Turn on terminate flag

end_87:
ret

;====================================================
```

3.7 PC/AT ROS Interrupt Support

Concurrent will intercept some of the ROS interrupts and emulate a limited subset of functions, the remaining functions will be passed on to the original interrupt service routines (ISRs). This means that the XIOS INIT routine should set all the interrupt vectors between 10h and 1Fh to point to XIOS code. These XIOS service routines should either handle the function requested or print an error message and terminate the current process. Obviously on a PC or compatible many of the functions can be passed on to the ROS.

The following ROS interrupts are intercepted by Concurrent.

INT 11h System Configuration

This function calls the PC_EQUIP XIOS function.

INT 12h Memory Size
The PCMODE returns the correct memory size based on the memory allocated to the current process.

INT 13h Disk I/O
Concurrent DOS uses the IO_INT13 XIOS function to support the Disk I/O ROS function.

INT 14h Serial I/O
Sub-functions 1 (Send Character), 2 (Get Character) and 3 (Status) are trapped by Concurrent and passed to the appropriate AUX devices. All other sub-functions are passed to the original ISR. Check new PS/2 sub-functions.

INT 16h Keyboard Services
Sub-functions 0 to 2 and 10h to 12h are emulated by Concurrent. All other sub-functions are passed to the original ISR. INT 16h is commonly used by clone manufacturers to access extra facilities provided by their hardware, ie Processor Speed Switching.

INT 17h Parallel Printer I/O
Sub-functions 0 (Print Character) and 2 (Status) are emulated by Concurrent. All other sub-functions are passed to the original ISR.

3.8 XIOS SYSTEM CALLS

Routines in the XIOS cannot make system calls in the conventional manner of executing an INT 224 instruction. The conventional entry point to the SUP does a stack switch to the User Data Area (UDA) of the current process. The XIOS is considered within the operating system and a process entering the XIOS is already using the UDA stack. A separate entry point is used for internal system calls.

Location 0003H of the SUP code segment is the entry point for internal system calls. Register usage for system calls through this entry point is similar to the conventional entry point. They are as follows:

Entry:	CX	= System call number
	DX	= Parameter
	DS	= Segment address if DX is an offset to a structure
	ES	= User Data Area
Return:	AX	= BX = Return
	CX	= Error Code
	ES	= Segment value if system call returns an offset and segment. Otherwise ES is unaltered and equals the UDA. DX, SI, DI, BP are not preserved.

The only differences between the internal and user entry points are the contents of registers CX and ES on entry. CH must always be 0 for the internal call. ES must always point to the UDA of the current process. You can obtain the UDA segment address with the following code:

```
rlr equ word ptr .68h ; ready list root

mov si,rlr
mov es,10h[si]
```

Note: On entry to the XIOS, the ES register is equal to the UDA segment address. ES must equal the UDA on return from any XIOS function called by the XIOS ENTRY routine. Interrupt routines must restore ES and any other altered registers to their value upon entry to the routine and before performing an IRET instruction or a JMPF to the dispatcher.

CHARACTER DEVICES

This section describes the XIOS character I/O functions. To run DOS programs, your XIOS must support the functions described in Section 7.

Concurrent calls all serial I/O devices consoles at the programmer level. Each process contains, in its Process Descriptor, the number of its default console. The default console can be either a Virtual Console (a console from which the user may run an application and for which a terminal message process (TMP) exists) or an extra serial I/O device. Each Virtual Console is assigned to a specific physical console (user terminal). The system associates a Console Control Block (CCB) with each serial I/O device (Virtual Console or extra I/O device). The serial I/O devices and CCBs are numbered relative to zero.

Concurrent can be configured in a number of different ways by changing the CCB Table in the XIOS. You may configure the system to support one or more physical consoles and extra I/O devices. The number of Virtual Consoles assigned to each physical console is set in the CCB Table. Up to 256 serial I/O devices can be implemented, depending on the specific application.

The XIOS Header defines the size and location of the CCB Table. The Headers, CCB field points to the beginning of the CCB Table; the NCCB field contains the number of CCB Table entries. The CCBLIST contains a pointer to an ordered array of pointers to the CCBs. The value of the NVCNS field declares how many of the CCBs are Virtual Consoles (see Section 3.1, "XIOS Header").

The XIOS might or might not maintain a buffer containing the screen contents and cursor position for each Virtual Console, depending on how the system is to appear to the user. Keep in mind that this buffer can be over 4K bytes per Virtual Console. Practical considerations of memory space might require keeping the number of Virtual Consoles reasonably small if buffers are maintained. The buffers are a requirement for DOS terminal support (Concurrent 386).

By convention, the first NVCNS serial I/O devices are the Virtual Consoles. The NVCNS parameter is located in the XIOS Header. The NPCNS field indicates the number of user terminals. NPCNS must be less than or equal to NVCNS. NPCNS does not include extra I/O Devices. Consoles beyond the last Virtual Console represent other serial I/O devices. When a process makes a console I/O call with a console number higher than the last Virtual Console, it references the Console Control Block for the called device number. A CCB for each serial I/O device is absolutely necessary.

List devices are output-only under Concurrent. The XIOS must reserve and initialize a List Control Block for each list output device. When a process makes a list device XIOS call, it references the appropriate LCB.

Similarly, the XIOS must reserve and initialize an Auxiliary Control Block (ACB) for each auxiliary device. The number of ACBs must equal the number specified in the NACB field of the XIOS Header. See Section 4.5.

There are two basic methods that Concurrent XIOS drivers use to wait for a hardware event to occur: noninterrupt-driven devices use the DEV_POLL method; interrupt-driven devices use the DEV_SETFLAG/DEV_FLAGWAIT method. Both methods allow a process waiting for an external event to give up the CPU resource so that other processes can run concurrently. See Section 6 of the Programmer's Guide for detailed explanations of the DEV_POLL, DEV_FLAGWAIT and DEV_SETFLAG system calls.

4.1 CONSOLE CONTROL BLOCK

A Console Control Block Table must be defined in the XIOS. There must be one CCB for each Virtual Console and Character I/O device supported by the XIOS, as indicated by the NCCB variable in the XIOS Header. The table must begin at the address indicated by the CCB variable in the XIOS Header. Figure 4-1 shows the format of the CCB Table.

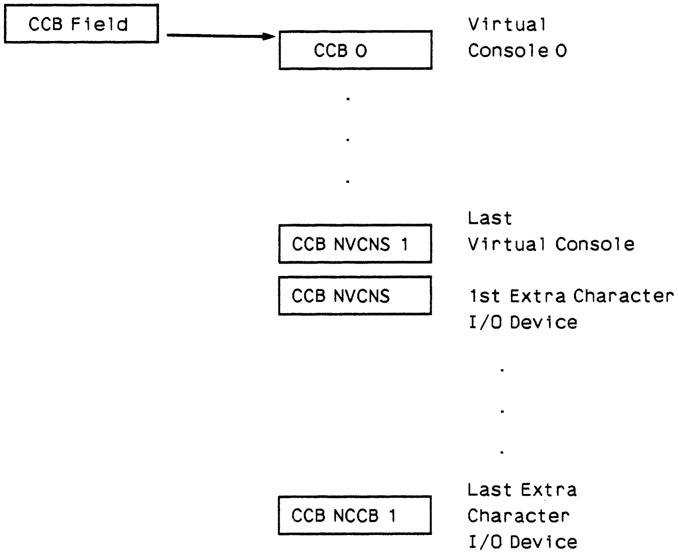


Figure 4-1. The CCB Table

The CCBLIST pointer in the XIOS header points to an array of pointers to the CCBs. This array should be used to access a particular CCB so that future modifications to the length of a CCB structure can be made.

Example of the CCB array for 6 CCB's

CCBLIST

00H	CCB(0)	CCB(1)	CCB(2)	CCB(3)
08H	CCB(4)	CCB(5)		

The number of CCBs used for Virtual Consoles equals the NVCNS field in the XIOS Header. Any additional CCB entries are used for other character devices to be supported by the XIOS. CCB entries are numbered starting with zero to match their logical console device numbers. The last CCB in the CCB Table is the (NCCB−1)th CCB.

Each CCB corresponding to a Virtual Console has several fields which must be initialized, either when the XIOS is assembled or by the XIOS INIT routine. These fields allow you to choose the configuration of the Virtual Consoles. The PC field indicates the physical console to which this Virtual Console is assigned. The VC field is the Virtual Console number. This number must be unique within the system. The LINK field points to the CCB of the next Virtual Console assigned to this physical console. The last Virtual Console assigned to each physical console should have the LINK field set to zero (0000H). For CCBs outside the Virtual Console range corresponding to extra I/O devices, these fields must all be initialized to zero (00H), except for the PC field. All fields marked RESERVED in Figure 4-2 must also be initialized to zero (OOH).

Figure 4-2 shows the format of a Console Control Block. Table 4-1, immediately following Figure 4-2, defines the Console Control Block fields.

OOH	OWNER		RESERVED					
O8H	MIMIC	RES.	PC	VC	RES.	ATTR	STATE (O)	
1OH	RESERVED							
28H	LINK		RESERVED					

Figure 4-2. Console Control Block Format

Table 4-1. Console Control Block Fields

Field	Explanation
OWNER	Address of the Process Descriptor for the process that currently owns this Virtual Console or character I/O device. This field is used by the XIOS Status Line Function (IO_STATLINE) to find the name of the current owner. Initialize this field to zero (0000H). If the value in this field is zero when Concurrent is running, no process owns the device.
MIMIC	This field indicates the list device that receives the characters typed on the Virtual Console when CTRL-P is in effect. MIMIC must be initialized to 0FFH. Note that this list device is not necessarily the same as the default list device indicated in the Process Descriptor pointed to by the OWNER field.
PC	Physical console number.

Table 4-1. (Cont'd)

Field	Explanation
VC	Virtual Console number. Virtual Console numbers must be unique within the system.

ATTR	The attribute field specifies which IBM PC characteristics are supported by this virtual console. The default value 0 corresponds to plain TTY support.
	The bits are defined as follows.
	`CA_251INES    equ 02h    ; Supports 25 line operation`
	`CA_ANSI       equ 04h    ; Supports ANSI control codes`
	`CA_ROS        equ 08h    ; Provides ROS emulation`
	`CA_HARDWARE   equ 10h    ; PC hardware compatible`
	Other bits are RESERVED.

STATE	Bit 3 in the lower byte of this field is used as a NOSWITCH flag. When set, this flag disables an attempt to switch Virtual Consoles on this console. Initialize this field to zero.
LINK	Address of the next CCB assigned to the same physical console. This field contains zero (0000H) if this is the last or only Virtual Console for this physical console.

Figure 4-3 depicts the CCBs for a system with two physical consoles, the first with three Virtual Consoles and the second with two.

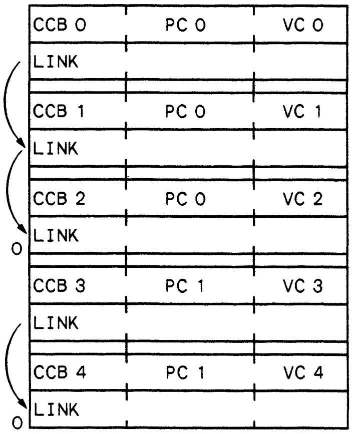


Figure 4-3. CCBs for Two Physical Consoles

4.2 CONSOLE I/O FUNCTIONS

IO_CONST

Return input status of a specified serial I/O device

Entry Parameters:

Register	AL:	00H
	DL:	Device number

Returned Values:

Register	AL:	0FFH if character ready 0 if no character ready
	BL:	Same as AL

ES, DS, SS, SP: preserved

The IO_CONST routine returns the input status of the specified character I/O device. This function is called by the operating system only for console numbers greater than NVCNS-1, in other words, only for devices which are not Virtual Consoles. If the status returned is 0FFH, then one or more characters are available for input from the specified device.

IO_CONIN

Return character from console keyboard or serial I/O device

Entry Parameters:

Register	AL:	01H
	DL:	Device number

Returned Values:

Register	AH:	00H if returning character data
	AL:	Character
	AH:	0FFH if returning a switch screen request
	AL:	Virtual Console requested
	BX:	same as AX in all cases

ES, DS, SS, SP: preserved

Because Concurrent supports the full 8-bit ASCII character set, the parity bit must be masked off from input devices which use it. It should not, however, be masked off if valid 8-bit characters are being input.

You choose the key or combination of keys that represent the Virtual Consoles by the implementation of IO_CONIN. One of the example XIOSs uses the function keys F1 through F3 to represent the Virtual Consoles assigned to each user terminal.

IO_CONIN must check for PC Mode. PC Mode is active whenever DOS programs are running. It is enabled or disabled by the PC_KBD function (32). If PC Mode is enabled, all function keys are passed through to the calling process. If it is disabled, function keys that do not have an associated XIOS function are usually ignored on input. See Section 7.3 "Keyboard Functions" for information on PC_KBD.

IO_CONOUT

Display and/or output a character to the specified device

Entry Parameters:

Register	AL:	02H
	CL:	Character to send
	DL:	VC number

The XIOS might or might not buffer background Virtual Consoles, depending on the user interface desired, memory constraints, and methods of updating the terminals. This section describes how the example XIOSs handle Virtual Consoles.

The example XIOSs buffer all Virtual Consoles. All Virtual Consoles have a screen image area in RAM. This image reflects the current contents of the screen, both characters and attributes. Each screen image is contained in a separate segment. Note that PC Mode requires that the screen buffers consist of 25 lines; only 24 lines need to be buffered when not in PC Mode.

Each Virtual Console also has a VC Structure associated with it. This structure contains the segment address of the screen image, the cursor location (offset in the segment), and any other information needed for the screen. The VC structure can be expanded to support additional hardware requirements, such as color CRTs. See Section 4.3.2 for a description of the VC Structure.

When a process calls this function with a device number higher than the last Virtual Console number, the character should be sent directly to the serial device that the CCB represents.

IO_DEVIO (27H) is used to output a block of characters to a console device; See Section 4.6.

IO_SWITCH

Switch the current Virtual Console into the background
and the specified Virtual Console into the foreground

Entry Parameters:

Register	AL:	07H
	DL:	VC number to switch to foreground

IO_SWITCH XIOS copies the screen image in memory to the physical screen, moving the cursor on the physical screen to the proper position for the new foreground console.

IO_SWITCH is responsible for performing a flagset to restart a background process that is waiting to go into graphics mode. If the process's screen is to be switched into the foreground, your flagset should use the flag that was used by GET_SET_SCREEN (30) to flagwait the process. See Section 7.1 for a description of GET_SET_SCREEN.

IO_SWITCH will be implemented differently for machines with video RAM (such as the IBM Personal Computer) and serial terminals. For IBM machines, the screen switch can be accomplished by performing a block move from the screen image to the video RAM, and a physical cursor positioning. A serial terminal must be updated by sending a character at a time, inserting escape sequences for the attribute changes.

Concurrent calls IO_SWITCH only when there is no process currently in the XIOS performing console output to either the foreground Virtual Console being switched out, or the background Virtual Console being switched into the foreground. Therefore, the XIOS never has to update a screen while simultaneously switching it from foreground to background, or vice versa.

One of the example IO_SWITCH routines performs the following operations:

1. Get the screen structure and image segment for the new Virtual Console.

2. Find the physical console number for this Virtual Console.

3. If this is a video-mapped console, save the current display by doing a block move. If it is a serial terminal, clear the physical screen and home the cursor.

4. If this is a video-mapped display, do a block move of the new screen image to the video RAM, and re-position the cursor. If it is a serial terminal, send each character to the physical screen. Check each character's attribute byte, and send any escape sequences necessary to display the characters with the correct attributes.

IO_STATLINE

Display specified text on the status line

Entry Parameters:

Register	AL:	08H
	CX:	0000H to continue updating the normal status line Offset to print the string at DX:CX 0FFFFH to resume normal status line display
	DL:	physical console to display status line on (CX = 0)
	DX:	segment address of optional string (CX <> 0)

When IO_STATLINE is called with CX = 0, the normal status information is displayed by IO_STATLINE on the physical console specified in DL. The normal status line typically consists of the foreground Virtual Console number, the state of the foreground Virtual Console, the process that owns the foreground Virtual Console, the removable-media drives with open files, whether control P, S, or O are active, and the default printer number. The IO_STATLINE function in the example XIOSs display some of the above information. Usually when IO_STATLINE is called, DL is set to the physical console on which to display the status line. You must translate this to the current (foreground) Virtual Console before getting the information for the status line (such as the process owning the console). The status line can be modified, expanded to any size, or displayed in a different area than the status line implemented in the example XIOSs. A common addition to the status line is a time-of-day clock.

Digital Research recommends that you implement a status line. However, if there are only 24 lines on the display device, you might choose not to implement a status line. In this case, the IO_STATLINE function should just return when called.

The normal status line is updated once per second by the CLOCK RSP. If there is more than one user terminal connected to the system, this update occurs once per second on a round-robin basis among the physical terminals. Thus, if four terminals are connected, each one is updated every four seconds.

The operating system also requests normal status line updates when screen switches are made and when control P, S, or O change state. The XIOS might call IO_STATLINE from other routines when some value displayed by the status line changes.

Note: IO_STATLINE's re-entrancy depends in part on having separate buffers for each physical console.

The IO_STATLINE routine should not display the status line on a user terminal that is in graphics mode. It should check the same variable as GET_SET_SCREEN (Function 30). GET_SET_SCREEN is described in Section 7.1, "Screen I/O Functions."

IO_STATLINE should not display on a console that is in PC Mode. Check the variable set with PC_KBD (32) to see if a console is in PC Mode. See Section 7.3 for information on PC_KBD.

Most calls to IO_STATLINE to update the status line have DL set to the physical terminal that is to be updated. When IO_STATLINE is called with CX not equal to 0000H or 0FFFFH, then CX is assumed to be the byte offset and DX the paragraph address of an ASCII string to print on the status line. This special status line remains on the screen until another special status line is requested, or IO_STATLINE is called with CX=0FFFFH. While a special status line is being displayed, calls to IO_STATLINE with CX=0000H are ignored.

When IO_STATLINE function is called with CX=0FFFFH, the normal status line is displayed and subsequent calls with CX=0000H cause the status line to be updated with current information.

When IO_STATLINE is called to display a special status line, DL does not contain the physical console number. The physical console number can be obtained by the following method:

1. Get the address of SYSDAT.

2. Look at the RLR (Ready List Root). The first process on the list is the current process.

3. Look at the Process Descriptor (pointed to by RLR). The P_CNS field contains the Virtual Console number of the current process. See the Programmer's Guide for a description of the Process Descriptor.

4. Look up the CCB for this Virtual Console and find the physical console number in it.

A process calling IO_STATLINE with a special status line (DX:CX = address of the string) must call IO_STATLINE before termination with CX=0FFFFH. Otherwise the normal status line is never shown again. There is no provision for a process to find out which status line is being displayed.

4.3 XIOS WINDOW SUPPORT

This section describes the XIOS functions and data structures that support Virtual Console windowing as implemented in the sample XIOS for the IBM PC/XT/AT/PS2. This section also outlines the algorithms used to perform console output and screen switching in a system with Virtual Consoles and windows.

Concurrent's window support is implemented exclusively in the XIOS. The windowing scheme presented in the sample XIOS for the IBM PC/XT/AT/PS2 involves two utilities, WINDOW and WMENU, which run as transient programs, and nine optional entry points used by these utilities. The window utilities perform XIOS calls under the protocol described in the Programmer's Guide. WINDOW and WMENU are described in the User's Reference Guide.

4.3.1 XIOS Window Functions

Table 4-2 summarizes the XIOS functions that support window manipulation. These functions are accessed with a CALL FAR instruction to the XIOS entry point using the standard XIOS segment register conventions, specifically: DS = SYSDAT and ES = UDA. At entry, each routine is called with a function code in AL and various parameters in BX, CX, and DX. Note that where a Virtual Console (VC) number is specified as a parameter, it is a zero-based value.

Table 4-2. XIOS Window Functions

Function	Purpose
WW_POINTER	This call (10H) returns a pointer to either the VC structure or the Window Data Block. These data structures, described in Section 4.3.2, indicate the size and position of a window, a window's position in relation to other windows, and other useful information.
WW_KEY	WW_KEY (11H) returns keyboard input and status to the WMENU utility.
WW_STATLINE	This call (12H) is used by the window manager to control its status line display. An application may use this call to write to the status line area of the physical console.
WW_IM_HERE	This call (13H) sets the window manager process state and allows an application to switch a new window to the foreground.
WW_NEW_WINDOW	This call (14H) defines new window boundaries for a Virtual Console.
WW_CURSOR_VIEW	This call (15H) sets the cursor track mode and viewpoint.
WW_WRAP_COLUMN	This call (16H) sets the column for automatic wrap-around to prevent characters from being lost outside the window during simple console output calls.
WW_FULL_WINDOW	This call (17H) toggles the current Virtual Console between full screen and its previous definition.
WW_SWITCH_DISPLAY	This call (18H) switches a Virtual Console from one display monitor to another, clears both screens, and updates all Virtual Consoles.

Each XIOS window function call is described below.

WW_POINTER

Get window information

Entry Parameters:
Register AL: 10H (16)
DL: VC number or FFH

Returned Values:
Register AX: Address of VC Structure (DL = VC number)
Address of Window Data Block (DL = FFH)

WW_POINTER provides current window information to the windowing utilities or other calling programs. If this function is called with the Virtual Console number in register DL, it returns the address of the Virtual Console Structure. If WW_POINTER is called with FFH in DL, it returns the address of the Window Data Block. Section 4.3.2 defines the VC Structure and the Window Data Block.

WW_KEY

Get keyboard Input/Status

Entry Parameters:
Register AL: 11H (17)
CL: (1) FFH for keyboard input/status
(2) FEH for keyboard status only
(3) Less than FEH to wait for input

Returned Values:
Register AL: (1) Character or 00H if no character ready
(2) FFH if character ready, 00H if no character ready
(3) character

AH: Key type value, 00H for regular key
FFH for special key

The PC XIOS assigns a unique system flag to the window manager. If a character has not been typed when input is requested by WMENU, the WMENU process "sleeps" on that flag by performing a DEV_WAITFLAG system call (84H) to take itself out of context until a key is entered. The keyboard ISR sets that flag with DEV_SETFLAG (85H) when the next key is typed, awakening WMENU. The DEV_WAITFLAG and DEV_SETFLAG calls are described in Section 6 of the Programmer's Guide.

Note that AH returns a key type value only if AL returns a character.

WW_STATLINE

Display status line characters/attributes

Entry Parameters:

Register	AL:	12H (18)
	CX:	0000 to continue updating normal status line Offset of string to print FFFFH to resume normal status line
	DL:	Physical console on which to display status line (when CX contains 0)
	DX:	Segment address of optional string (when CX does not contain 0)

Returned Values:

Register	AL:	0 on success 0FFH on failure

WW_STATLINE is a duplicate of IO_STATLINE (08H) with the exception that registers DX:CX address an 80-word string of characters/attributes instead of an 80-byte string of characters. This provides more complete control over the status line display.

WW_IM_HERE

Switch window

Entry Parameters:

Register	AL:	13H (19)
	CL:	Window Manager state: 0 Manager not resident 1 Resident but not active 2 Resident and active 3 Switch window, do not change state
	DL:	Number of Virtual Console to switch If FFH, do not switch console

WW_IM_HERE switches a window by emulating a keyboard interrupt. It does this by first poking the "key" variables normally set by the keyboard ISR with the codes that correspond to the proper screen-switch key. It then performs a DEV_SETFLAG system call (85H) to set the keyboard flag. This function places the Window Manager state value contained in CL into the im_here field of the Window Control Block. The Window Control Block is described in Section 4.3.2.

WW_NEW_WINDOW

Define window boundaries

Entry Parameters:

Register	AL:	14H (20)
	DL:	VC number
	CH:	Top left row
	CL:	Top left column
	BH:	Bottom right row
	BL:	Bottom right column

The XIOS defines a semaphore for each Virtual Console in the system. Before any XIOS routine attempts to update a window in any way, the semaphore for the window's Virtual Console is checked. If another process is in the window code, the XIOS waits until that process is finished before continuing.

Because defining new window boundaries can affect all Virtual Consoles, WW_NEW_WINDOW gains ownership of all semaphores before proceeding. At the point when it owns all semaphores, all other processes are locked out of the console output code. After updating the window variables, erasing the contents of the video display, and updating all windows, WW_NEW_WINDOW frees all the semaphores so that other processes may resume console output.

WW_CURSOR_VIEW

Set cursor tracking and update window

Entry Parameters:

Register	AL:	15H (21)
	DL:	VC number
	DH:	Cursor tracking mode: 00 Fixed window (full 80x25 image) 01 Track scrolling cursor
	CH:	Top left row
	CL:	Top left column

After obtaining the semaphore for a given console, WW_CURSOR_VIEW updates the variables in the Virtual Console Structure. Once it has updated the window on the screen and displayed the cursor if it falls within the window, this function releases the semaphore and returns.

When DH contains 01 on entry, WW_CURSOR_VIEW tracks the cursor by maintaining cursor row inside the window.

WW_WRAP_COLUMN

Set Screen Width

Entry Parameters:

Register	AL:	16H (22)
	DL:	VC number
	CL:	Wrap column number

WW_WRAP_COLUMN updates the width field of Virtual Console Structure for the selected Virtual Console with the specified column number. This function sets the Virtual Console's screen width.

WW_FULL_WINDOW

Switch window size

Entry Parameters:

Register	AL:	17H (23)
	DL:	VC number

WW_FULL_WINDOW switches the Virtual Console currently in the foreground between full screen size and its previously defined size.

The XIOS must check whether the current window is full screen for this function. If it is, retrieve the previous window boundaries from the Virtual Console Structure and internally call WW_NEW_WINDOW. If the current window is not full screen, call WW_NEW_WINDOW to make it full screen.

WW_SWITCH_DISPLAY

Switch Virtual Console monitors

Entry Parameters:

Register	AL:	18H(24)
	DL:	VC number
	CL:	Monitor code:
		00 Monochrome
		01 Color

WW_SWITCH_DISPLAY updates the Virtual Console Structure to indicate the monitor (color or monochrome) on which a window is to be displayed. For a system that has both a color and a monochrome monitor, this function moves a window from one monitor to the other. Both displays are then cleared and all windows are updated.

4.3.2 Virtual Console Data Structures

The Concurrent XIOS maintains a Virtual Console Structure for each Virtual Console in the system. WW_POINTER (10H) returns a pointer to the VC Structure that corresponds to the VC number passed as an entry parameter. Structure members include cursor position, window position, current attribute, and cursor tracking mode. Do not confuse the VC Structure with the Console Control Block (CCB).

Listing 4-1 shows a VC Structure as implemented in the XIOS for the IBM machines. NOTE this structure is not the same on the example XIOS for serial DOS consoles. These do not support character windows.

Listing 4-1. Virtual Console Structure

```
cur_col          rb 1        ;cursor column
cur_row          rb 1        ;cursor row
left             rb 1        ;window left column
top              rb 1        ;window top row
right            rb 1        ;window right column
bottom           rb 1        ;window bottom row
old_tl           rw 1        ;old top-left
old_br           rw 1        ;old bottom-right
crt_size         rw 1        ;rows/columns of full screen
win_size         rw 1        ;rows/columns of window
view_point       rw 1        ;row/column of current view point
rows             rw 1        ;window row count
cols             rw 1        ;window column count
correct          rw 1        ;character position factor
vc_seg           rw 1        ;segment base of console image
crt_seg          rw 1        ;segment base of CRT screen area
list_ptr         rw 1        ;used for updating the screen by rows
attrib           rb 1        ;current character attribute
mode             rb 1        ;current screen mode
cur_track        rb 1        ;current tracking mode (ROW/NO)
width            rb 1        ;column where character wrap occurs
number           rb 1        ;vc number
bit              rb 1        ;vc number as a bit position
save_curs        rw 1        ;cursor save location
vector           rw 1        ;ptr to current output routine
xlat             rw 1        ;address of priority-translate table
crt_rows         rb 1        ;=25 for PCDOS screen, 24 for CP/M
pc_shift         rb 1        ;current PCMODE shift state
true_view        rw 1        ;true view point for row tracking
cur_type         rw 1        ;mono or color
(function-key info.)
mxsemaphore      rb 1        ;mutual-exclusion semaphore
(reg. save area for back-door functions)
screen_mode      rb 1        ;alpha/graphics mode
screen_save      rb 1        ;save the old mode
```

Listing 4-2 defines a Window Data Block as implemented in the example XIOS for the IBM machines. WW_POINTER returns a pointer to this data structure when DL = FFH on entry.

Listing 4-2. Window Data Block

```
im_here         rb 1        ;If non-zero, the Window Manager is
                            ;present--see WW_IM_HERE (13H).
num_window      rb 1        ;Number of VCs (windows)
                            ;supported in the system.
priority        rb #w-1     ;A list of window numbers from lowest
top_screen      rb 1        ;priority (back) to highest priority
                            ;("top" or front).
```

4.3.3 Basic Console Output Algorithms

To support the four Virtual Consoles on the IBM machines, four full-screen-image RAM buffers are maintained, each 2000 (25 x 80) words (character plus attribute) in length. In a non-windowed Virtual Console system with a memory-mapped video display, character output in the XIOS consists of the following steps:

1. Check whether the current Virtual Console is switched-in.

2. If it is, put the character and attribute directly in the video display RAM.

3. If the current Virtual Console is not switched-in, put the character and attribute into the appropriate screen-image buffer.

4. Handle escape sequences (to position cursor, clear screen, set attribute, and so forth) properly.

Here are the steps required to switch Virtual Consoles in the system described above:

1. Copy the contents of the video display RAM to the screen-image buffer for that Virtual Console.

2. Copy the screen-image buffer for the newly switched-in Virtual Console into the video display RAM.

In a windowed system, the algorithms become more complex. To support windows in addition to the full-screen buffers for each Virtual Console, there must also exist a 2000-byte (80 x 25) "ownership" map in which each byte corresponds to a unique character position on the physical screen. The value of the byte indicates which window owns that character position. The priority of ownership changes according to the screen-switch history. Take the following steps to output characters in such a system:

1. If the current Virtual Console is full screen size and on top, place the character and attribute directly into the video display RAM, handle all escape sequences, and return.

2. If the current Virtual Console is not full screen and on top, put the character and attribute into the appropriate screen-image buffer.

3. If the current character position is not within the defined window for the Virtual Console, simply update the virtual cursor information and return.

4. If the current character position is within the window, check the ownership map. If the Virtual Console owns the current character position, place the character and attribute directly in the video display RAM.

5. Update the virtual cursor information and return.

The console–switch steps for the windowed Virtual Console system are as follows:

1. Draw a single–line frame around the old window in the video display RAM.

2. Update the ownership map described above.

3. Draw a double–line frame around the new window in the video display RAM.

4. Update the current window from the screen–image buffer.

5. Set the physical cursor from the virtual cursor information and return.

4.4 LIST DEVICE FUNCTIONS

This section describes the data structures and functions that the XIOS uses to support list devices.

Concurrent uses two data structures to manage list devices: List Control Blocks (LCBs) and the LCB Table. One LCB, similar to the CCB, must be defined in the XIOS for each list output device supported. The number of LCBs must equal the NLCB variable in the XIOS Header. The LCB Table, shown in Figure 4–4, begins with LCB zero, and ends with LCB NLCB–1, according to their logical list device names.

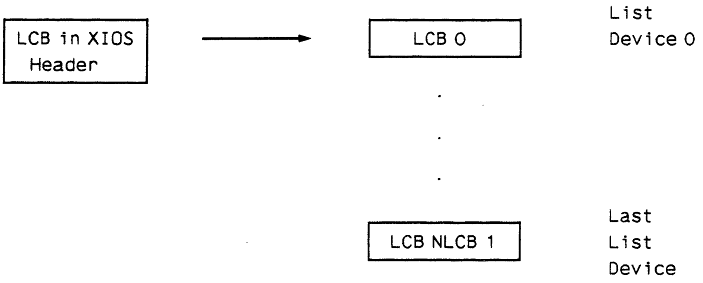


Figure 4–4. The LCB Table

In addition there is an LCBLIST pointer in the XIOS header to an array of pointers to the LCBs. This is to be used for all access to any LCB.

Because the operating system uses the LCBs to manage processes that make list device calls, each LCB Table entry must be properly initialized, either by the XIOS INIT routine or at XIOS assembly time. Table 4–3 lists the initialization values for the LCB fields.

00H	OWNER		RESERVED					
08H	RE-SERVED	M-SOURCE						

Figure 4-5. List Control Block

Table 4-3. List Control Block Fields

Field	Explanation
OWNER	Address of the PD for the process that currently owns the List Device. If no process currently owns the list device, then OWNER=0. If OWNER=0FFFFH, this list device is mimicking a console device that is in CTRL-P mode.
MSOURCE	If OWNER=0FFFFH, MSOURCE contains the number of the console device that this list device is mimicking; otherwise MSOURCE = 0FFH. **Note:** MSOURCE must be initialized to 0FFH. All other LCB fields must be initialized to 0.

IO_LSTST

Return list status

Entry Parameters:

Register	AL:	03H
	DL:	List device number

Returned Values:

Register	AL:	0FFH if device ready 0 if device not ready
	AH:	90H if device ready 10H if device not ready
	BL:	Same as AL
	BH:	Same as AH

ES, DS, SS, SP:preserved

IO_LSTST returns the output status of the specified list device.

IO_LSTSTOUT

Output character to specified list device

Entry Parameters:

Register	AL:	04H
	CL:	Character
	DL:	List device number

IO_LSTOUT sends a character to the specified list device. List device numbers are zero-based. It is the responsibility of the XIOS device driver to zero the parity bit for list devices that require it.

IO_DEVIO (27H) is used to output a block of characters to a list device; see Section 4.6.

4.5 AUXILIARY DEVICE FUNCTIONS

This section describes the data structures and functions that the XIOS uses to support auxiliary devices.

There are two data structures that the XIOS uses to manage auxiliary device access, Auxiliary Control Blocks (ACBs) and the ACB Table. The XIOS must define an ACB for each auxiliary device it supports. The number of ACBs must equal the number specified in the NACB field of the XIOS Header (offset C0FH). The ACB field of the XIOS Header (offset C3AH) contains the offset (relative to SYSDAT) of the first ACB. The ACB Table begins with ACB (0) and ends with ACB (NACB-1). As shown in Figure 4-6, the ACB Table must immediately follow the LCB Table in your XIOS.

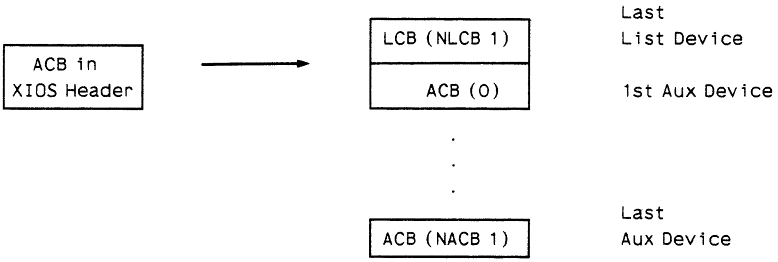


Figure 4-6. Auxiliary Control Block Table

In addition there is a ACBLIST pointer in the XIOS header to an array of pointers to the ACBs. This is to be used for all access to any LCB.

Figure 4-7 shows the ACB format.

00H	OWNER	RESERVED
08H	RESERVED	

Figure 4-7. Auxiliary Control Block

The OWNER field of the ACB contains the address of the PD of the process that currently owns the auxiliary device. If no process currently owns the device, OWNER equals zero.

The XIOS auxiliary device functions described below are accessible through device-related BDOS calls. Processes use these functions by placing a zero-based number in register DL to refer both to the physical device that is to be read or written to and to other parameters as appropriate.

IO_AUXIN

Return a character from an auxiliary device

Entry Parameters:

Register	AL:	05H
	DL:	Aux device number

Returned Values:

Register	AL:	Character

ES, DS, SS, SP:preserved

IO_AUXIN returns a character from the auxiliary device whose number is specified in register DL. Auxiliary device numbers are zero-based.

IO_DEVIO (27H) is used to return a block of characters from an auxiliary device; see Section 4.6.

IO_AUXOUT

Output a character to an auxiliary device

Entry Parameters:

Register	AL:	06H
	CL:	Character
	DL:	Aux device number

IO_AUXOUT sends a character to the auxiliary device whose number is specified in DL. Auxiliary device numbers are zero-based. The XIOS device driver is responsible for clearing the parity bit for the auxiliary devices that so require.

IO_DEVIO (27H) is used to output a block of characters from an auxiliary device; see Section 4.6.

IO_AUXSTIN

Return auxiliary device input status

Entry Parameters:

Register	AL:	25H (37)
	DL:	Aux device number

Returned Values:

Register	AL:	FF if character ready 00 if no input character available

IO_AUXSTOUT

Return auxiliary device output status

Entry Parameters:

Register	AL:	26H (38)
	DL:	Aux device number

Returned Values:

Register	AL:	FF if device ready
		00 if device not ready

IO_POINTER

Return address of machine-specific data

Entry Parameters:

Register	AL:	0EH (14)
	CL:	Data value:
		0 Equipment configuration
		1 Setup data
		2 VC switch bits
		3 Auxiliary protocol
		4 Color monitor scroll mode
		5 Offset of auxiliary port 0 interrupt routine
		6 Offset of auxiliary port 1 interrupt routine
		7 Offset of internal vector table [code segment]

Returned Values:

Register	AX:	Data/interrupt routine offset
		00 Data value out of range

IO_POINTER is called by the auxiliary I/O functions, IO_CONIN (01), IO_SWITCH (07), and the XIOS-specific disk utilities distributed with Concurrent. This function returns the offset, relative to the SYSDAT segment, of data that describe a machine's equipment configuration, setup information, Virtual Console switch bits, auxiliary port protocol, monitor scroll mode, and the auxiliary port interrupt routines.

The format of the data blocks whose offsets are returned by IO_POINTER are shown below.

Equipment configuration data block:

Offset	Data
00H	Number of floppy drives (1-4)
02H	Number of hard drives (0-2)
04H	Number of parallel printers (0-3)
06H	Number of serial ports (0-2)
08H	Number of monochrome monitors (0-1)
0AH	Number of color monitors (0-1)
0CH	Number of 8087 processors (0-1)
0EH	Main memory in kilobytes (256-640)
10H	Memory disk in kilobytes (0-384)
12H	Number of floppy designators (2-4)

Setup data block:

Offset	Data
00H	Offset of disk interrupt routine
02H	Segment of disk interrupt routine
04H	Table of function key assignments for Virtual Console 0
06H	Table of function key assignments for Virtual Console 1
08H	Table of function key assignments for Virtual Console 2
0AH	Table of function key assignments for Virtual Console 3
0CH	Last segment of main memory (4000H - A000H)
0EH	Beginning segment of high memory (C000 - E000H)
10H	Last segment of high memory (D000H - F000H)

Auxiliary I/O protocol byte:

Bit	Assignment
00	Ready, no protocol
01	DTR/DSR
02	RTS/CTS
04	XON/XOFF
80	Receive inhibited

Scroll mode byte:

Value	Meaning
00	Disable video around buffer update
01	Update only around vertical and horizontal retrace
02	No check for retrace occuring

4.6 DEVICE BLOCK READ/WRITE FUNCTION

IO_DEVIO

Device Block Read/Write

Entry Parameters:

Register	AL:	27H (39)
	DH:	Device type: 0 Console output 1 Printer output 2 Aux input 3 Aux output
	DL:	Device number
	CX:	Number of characters

Return Values:

Register	AX:	Number of characters transferred (if Aux device) or -1 on device type range error

ES, DS, SS, SP:preserved

IO_DEVIO reads or writes a block of characters to the device whose type and number is specified in registers DH and DL, respectively.

Before calling IO_DEVIO, a process must PUSH registers DX, CX, and the segment and offset of the character buffer on the stack. See Figure 4-8. The calling process must also clear the stack on return from IO_DEVIO.

SP + 6	Buffer Offset
SP + 8	Buffer Segment
SP + 10	CX
SP + 12	DX

Figure 4-8. Stack Contents for IO_DEVIO Call

If an auxiliary device is specified, IO_DEVIO returns a count of the characters transferred. IO_DEVIO returns -1 in AX if the device type value specified in DH is greater than 4.

IO_DEVIO updates two fields of the parameter block on return: Buffer Offset (SP + 6) is updated to address the last character read or written, plus one; the CX field (SP + 10) is updated to contain the number of characters remaining in the block.

4.7 IO_POLL FUNCTION

IO_POLL

Poll device and return status

Entry Parameters:

Register	AL:	0DH (13)
	DL:	Poll device number

Returned Values:

Register	AL:	0FFH if ready 00 if not ready
	BL:	Same as AL

ES, DS, SS, SP:preserved

IO_POLL interrogates the status of the device specified by the poll device number and returns its current status. This function is called by the Dispatcher.

A process polls a device only in response to Concurrent's DEV_POLL system call. The poll device number used as an argument for DEV_POLL is the same number that IO_POLL receives as a parameter. The mapping of poll device numbers to actual physical devices is maintained by the XIOS. Each polling routine must have a unique poll device number. For instance, if the console is polled, it must have different poll device numbers for console input and console output.

The sample XIOSs show IO_POLL taking the poll device number as an index to a table of poll functions. Once the address of the poll routine is determined, it is called and the return values are used directly for the return of the IO_POLL function.

DISK DEVICES

Under Concurrent, a disk drive is any I/O device that has a directory and is capable of reading and writing data in logical sectors. The XIOS can therefore treat a wide variety of peripherals as disk drives. The logical structure of a Concurrent disk drive is described in Section 11, "OEM Utilities." Concurrent's native media is fully compatible with PC-DOS, and MS-DOS (DOS media). The extensions used by Concurrent for password support is transparent to PC-DOS and MS-DOS. Concurrent supports CP/M media at the XIOS level only. This allows a utility supplied with Concurrent to access a CP/M floppy disk and perform all the basic functions on that disk (COPY, REN, ERA etc.).

This section describes the Concurrent XIOS disk functions, their input and output parameters, associated data structures, and how to calculate values for the XIOS disk tables. Also described in this section is Concurrent's support for DOS device drivers.

5.1 DISK I/O FUNCTIONS

Concurrent performs disk I/O with a single XIOS call to the IO_READ, IO_WRITE or IO_WRITE_VFY functions. These functions reference disk parameters contained in the Input/Output Parameter Block (IOPB) to determine which disk drive to access, the number of physical sectors to transfer, the track and sector to read or write and the offset and segment of the DMA involved in the I/O operation. The BDOS initializes and places the IOPB on the stack before each call. The XIOS may modify the IOPB during the call as it is discarded by the BDOS.

If a physical error occurs during a disk I/O function, the call should perform at least 10 recovery attempts before returning an error condition to the BDOS. Table 5-1 lists the extended error codes returned by disk I/O functions. Register al for IO_SELDSK, Register ah for IO_READ, IO_WRITE and IO_WRITE_VERIFY.

Table 5-1 Disk I/O Extended Error Codes

Code	Meaning	Code	Meaning
80H	Attachment failed to respond	7H	Drive setup error
40H	Seek operation failed	5H	Cannot reset disk
20H	Controller has failed	4H	Sector not found
10H	Bad CRC	3H	Write protect disk error
BH	Bad track flag set	2H	Address mark not found
9H	DMA address error	1H	Bad command
8H	DMA overrun		

IO_SELDSK

Select the specified disk drive

Entry Parameters:

Register	AL:	09H (09)
	CL:	Disk Drive number
DL(Bit 0):		0 if first select

Returned Values:

Register	AX:	Offset of DPH if no error
	AH:	0 if Error and
	AL:	Error code
	BX:	Same as AX

ES, DS, SS, SP: preserved

If it is the first select then IO_SELDSK checks if the specified disk drive is valid.

If the drive is valid this function returns the offset of the corresponding DPH. If more than one media type is possible on that drive, then IO_SELDSK should read a sector to determine the disk type and set the DPB field in the DPH for this media type. The DPB may be dynamically calculated at this time, but it must not require larger memory table space (CSV, DDSC) than was originally allowed. If the media is CP/M (function called from utility rather than from the BDOS) then use a standard DPB, otherwise it will be an extended DPB. If a physical error occurs or the media is unknown, the return AH will be zero and AL is set to the error code (if appropriate).

If it is not the first select then the function returns the current DPH.

Listing 5-1. IO_SELDSK XIOS Function

```
;**************************************************
;*        DISK IO CODE AREA
;**************************************************
;==========
IO_SELDSK:              ; Function 7: Select Disk
;==========
;       entry:  CL= disk to be selected
;               DL= 00h for first select
;                 = 01h if previously selected
;        exit:  AH= 0 if illegal disk, AL = error code.
;                 = offset of DPH from XIOS Data Segment

      xor bx,bx                         ; Get ready for error
      cmp cl,15                         ; Is it a valid drive
      ja sel_ret                        ; If not just exit
        mov bl,cl
        shl bx,1                        ; Index into the Dph's
        mov bx,dph_tbl[bx]              ; get DPH address from table
                                        ; in XIOS Header
      or dl,dl                          ; First time select?
      jnz sel_ret                       ; No, exit
        mov ch,0                        ; Yes, set up DPH
        mov si,cx
        shl si,1
        call word ptr sel_tbl[si]

sel_ret:
      mov ax,bx
      ret

;--------------------------------------------------
```

IO_READ

Read sector(s) defined by the IOPB

Entry Parameters: Initialized IOPB (on stack)
Register AL: 0AH (10)

Returned Values:
AL: 0 Success
1 Physical Error
0FF Media density changed
AH: Extended error code (see Table 5-1)
BL: Same as AL
BH: Same as AH

ES, DS, SS, SP: preserved

IO_READ transfers data from disk to memory according to the parameters specified in the Input/Output Parameter Block (IOPB). The IOPB, which is located on the stack, indicates the drive, multisector count, track, sector, and DMA offset and segment. See Section 5.2, "IOPB Data Structure."

If the multisector count is equal to 1, the function should attempt a single physical sector read based upon the parameters in the IOPB. If a physical error occurs, the read function should return 1 in AL and BL and the appropriate extended error code in AH and BH. The XIOS should perform several retry attempts (10 are recommended) before returning an error condition.

If the hardware detects a media density change (for disk drivers with auto density select) IO_READ should immediately return 0FFH. To reinitialize the drive's parameter tables and avoid writing erroneous data to disk, the BDOS then performs an IO_SELDSK call for that drive.

If the multisector count is greater than 1, IO_READ must read the specified number of physical sectors before returning to the BDOS. IO_READ should read as many physical sectors as the specified drive's disk controller can handle in one operation.

If the disk controller hardware can handle only single physical sector operations, the function must read the number of single sectors defined by the multisector count. In any case, when more than one call is made to the controller, the function must increment the sector number and add the number of bytes in each physical sector to the DMA address for each successive read. If the sector number exceeds the last physical sector of the current track, the function must increment the track number and reset the sector number to zero. This operation is illustrated in the portion of a hard disk driver routine contained in Listing 5-2.

The BDOS will issue an IO_READ with a multisector count of 0 after a door open interrupt has been signalled on this drive (removable media only). The XIOS should return the following values in AL

00L	Disk may have changed
01L	Disk has not changed
FFL	Disk has changed

Listing 5-2. Multisector Operations

```
;*****************************************************
;*  Common code for hard disk read and write
;*****************************************************

hd_io:
            push es                              ;save UDA
            cmp mcnt,0                           ;if multisector count = 0
            je hd_err                            ;return error
hdio1:
              call iohost                        ;read/write physical sector
              mov al,retcode                     ;get return code
              or al,al                           ;if not 0
              jnz hd_err                         ;return error
              dec mcnt                           ;decrement multisector count
              jz return_rw                       ;if mcnt = 0 return
                mov ax,sector
                inc ax                           ;next sector
                cmp ax,maxsec! jb same_trak ;is sector < max
                  inc track                      ; no - next track
                  xor ax,ax                      ; initialize sector to 0
same_trak:
                mov sector,ax                    ;save sector #
                add dmaoff,secsiz                ;increment dma offset by
                                                 ; sector size
                jmps hdio1                       ;read/write next sector
hd_err:
              mov al,1                           ;return with error
return_rw:
            pop es                               ;restore UDA
            ret                                  ;return w/ error code in AL

;*****************************************************
;* IOHOST performs the physical reads and writes to
;* the physical disk.
;*****************************************************

iohost:
 ...
 ...
 ...  ret
;-----------------------------------------------------------
```

In Listing 5-2, the routine returns with an error if the multisector count is zero. Otherwise, it immediately calls the read/write routine for the present sector, puts the return code passed from it into AL, and, if there is no error, decrements the multisector count. When the multisector count equals zero, the read or write is finished and the routine returns. If the multisector count is not zero, the sector to read or write is incremented. If the sector number exceeds the number of sectors on a track (MAXSEC) the track number is incremented and the sector number is set to zero. The routine then performs the number of reads or writes remaining to equal the multisector count. Each time a sector is read or written, the size of a physical sector is added to the DMA offset passed to the disk controller hardware.

IO_WRITE/ IO_WRITE_VFY

Write (write/verify) sector(s) defined by the IOPB

Entry Parameters: Initialized IOPB (on stack)
Register AL: 0BH (11)
CL: 0 Deferred write
1 Nondeferred write
2 Deferred write, first sector unallocated block

Returned Values:
Register AL: 0 Success
1 Physical error
2 Read/only disk
0FF Media density changed
AH: Extended error code (Table 5-1)
BL: Same as AL
BH: Same as AH
ES, DS, SS, SP: preserved

IO_WRITE transfers data from memory to disk according to the parameters specified in the IOPB. This function works in much the same way as the read function, with the addition of a read/only disk return code and an entry parameter that specifies whether a deferred write is to be performed.

IO_WRITE should return the read/only code when the specified disk controller detects a write-protected disk.

If your XIOS performs disk cacheing, check CL for the type of write operation to be performed by IO_WRITE. There is no need to check the contents of CL if your XIOS does not perform disk cacheing.

IO_FLUSH

Write pending I/O system buffers to disk

Entry Parameters:
Register AL: 0CH (12)

Returned Values:

Register	AL:	0 Success 1 Physical Error 2 Read/only disk
	AH:	Extended error code (Table 5-1)
	BL:	Same as AL
	BH:	Same as AH

ES, DS, SS, SP: preserved

IO_FLUSH is called when a process terminates, a file is closed, or a disk drive is reset to indicate that all disk-cacheing buffers should be written to disk. The XIOS should perform 10 recovery attempts before returning the error codes for this function.

IO_FORMAT

Format a CP/M disk

Entry Parameters:
Register AL: 0FH (15)

Returned Values:

Register	AL:	0 Success 1 Physical error 2 Read/only disk
	AH:	Extended error code (Table 5-1)
	BL:	Same as AL
	BH:	Same as AH

ES, DS, SS, SP: preserved

IO_FORMAT formats the current track of the current drive with eight sectors per track using the standard gap and CP/M fill character (E5H). This function adds the gap and fill values to the I/O Parameter Block on the stack (offset 16). See Section 5.2.

In the example XIOS, IO_FORMAT shares the FORMAT_FLOPPY routine with IO_NEW_FORMAT (1DH). The FORMAT_FLOPPY routine is shown in Listing 5-3 (see the IO_NEW_FORMAT function below). This function is not called by the kernel and is used only by system-specific utilities.

IO_NEW_FORMAT

Variable disk format

Entry Parameters: Initialized IOPB (on stack)
Register AL: 1DH (29)

Returned Values:
Register AL: 0 Success
1 Physical error
2 Read/only disk
AH: Extended error code (Table 5-1)
BL: Same as AL
BH: Same as AH
ES, DS, SS, SP: preserved

IO_NEW_FORMAT writes either the CP/M or DOS format on the current track of the disk in the current drive as specified by the I/O Parameter Block.

The referenced IOPB is extended to include a word at offset 16 that contains the gap value in the low order byte and the fill character in the high order byte. See Section 5.2 for a description of the IOPB. This function is not called by the kernel and is used only by system-specific utilities.

IO_NEW_FORMAT shares the FORMAT_FLOPPY routine shown in Listing 5-3 with IO_FORMAT (0FH).

Listing 5-3. FORMAT_FLOPPY Routine

```
; Floppy disk track format entry:

format_floppy:
;-------------
    mov         dma_mode_storage,DMA_MODE_WRITE_FDC
    mov         fdc_command,FDC_CMND_FORMAT

    call        comp_dma_param                  ; set up dma pointers
    mov         ax,TRACK                        ; current track (on stack)
    mov         cur_trk,ax
    mov         bl,DRIVE                        ; current drive (on stack)
    mov         cur_disk$,bl                    ; save for seek

    sub         bh,bh
    cmp         drive_type$[bx],SLOW_DRIVE
     jz         format_flop_go                  ; skip for older drives

    mov         ax,gap_fill                     ; if we're using a 96 tpi gap
    cmp         al,F_GAP_96                     ; then set data rate fast
     jz         format_flop_hyper

    mov         data_rate$[bx],RATE_HYPER_48
    jmps        format_flop_go

format_flop_hyper:
    mov         data_rate$[bx],RATE_HYPER_96

format_flop_go:
    mov         media_retry,1                   ; do not change data rate
    call        set_drive_type                  ; to allow for hyper drive
    call        disk_init                       ; this does it
    call        check_disk_op                   ; did it work?
    cmp         al,'R'                          ; if retry requested
     jz         format_floppy                   ; then loop up

    jmp         flp_ret                         ; handles extended errors
```

5.2 I/O PARAMETER BLOCK

This section presents the tables and data structures within the XIOS that define the characteristics of the Concurrent disk system. You must code the XIOS DPHs and DPBs by hand, using values calculated from the information presented below.

The disk Input/Output Parameter Block (IOPB) contains the parameters required for the disk I/O functions. These parameters are located on the stack, and appear at the example XIOS IO_READ and IO_WRITE function entry points as described below. The IOPB example in this section assumes that the ENTRY routine calls the read or write routines through only one level of indirection; the XIOS has therefore placed only one word on the stack. RET_ADR is reserved for this local return address to the XIOS ENTRY routine.

Since the IOPB parameters are removed by the BDOS when the function call returns, the disk drivers may index or modify them directly on the stack. The IOPB fields are usually defined relative to the BP and SS registers. One of the first instructions of the IO_READ and IO_WRITE routines sets the BP register equal to the SP register for indexing into the IOPB.

Listing 5-4 contains a definition of the IOPB and shows how it is used in IO_WRITE. See Table 5-2 for descriptions of the IOPB fields.

Listing 5-4. IOPB Definition

```
;********************************************************
;    I/O PARAMETER BLOCK FORMAT
;********************************************************
;
;Equates for parameter passing for read and write from BDOS.
;
;At the disk read and write function entries, all disk I/O ;parameters are on the
stack. The stack at these entries appears as follows:
;
;                  al        ah
;
;[      +16     +-----------------+
;               | GAP        FILL |     Used by io_new_format only]
;               +--------+--------+
;       +14     | DRV        MCNT |     Drive no. and multisector count
;               +--------+--------+
;       +12     |      TRACK      |     Track number
;               +--------+--------+
;       +10     |     SECTOR      |     Physical sector number
;               +--------+--------+
;        +8     |     DMA_SEG     |     DMA segment
;               +--------+--------+
;        +6     |     DMA_OFF     |     DMA offset
;               +--------+--------+
;        +4     |     RET_SEG     |     BDOS return segment
;               +--------+--------+
;        +2     |     RET_OFF     |     BDOS return offset
;               +--------+--------+
;      SP+0     |     RET_ADR     |     Return address to ENTRY routine
;               +-----------------+
;
;These parameters may be indexed and modified directly on the stack by the read and
write routines. They are removed by the BDOS when the function completes.

DRIVE       equ       byte ptr 14[bp]
MCNT        equ       byte ptr 15[bp]
TRACK       equ       word ptr 12[bp]
SECTOR      equ       word ptr 10[bp]
DMA_SEG     equ       word ptr 8[bp]
DMA_OFF     equ       word ptr 6[bp]

GP_FL       equ       word ptr 16[bp] ; gap & fill used by format

;********************************************************
```

Table 5-2. IOPB Data Fields

Field	Description
DRV	This field specifies the logical disk drive for the function. Drive numbers range from 0 to 15 for drives A through P, respectively.
MCNT	This field contains the multisector count. To transfer logically consecutive disk sectors to or from contiguous memory locations, the BDOS issues an IO_READ or IO_WRITE function call with the multisector count greater than one. The maximum value of the multisector count is 255. This is further limited by the maximum transfer size of 64 kbytes.
TRACK	This field defines the logical track for the specified drive to seek. The BDOS defines the track number relative to 0. If your disk hardware defines physical track numbers relative to 1, increment the track number befor passing it to the disk controller.
SECTOR	This field defines the logical sector for a read or write operation on the specified drive. The sector size is determined by the PSH and PHM parameters in the Disk Parameter Block. See Section 5.5. The BDOS defines the sector number relative to 0. If your disk hardware defines sector numbers relative to 1, increment the sector number before passing it to the disk controller. If the drive uses a skewed-sector format, your XIOS must translate the sector number according to the translation table specified in the Disk Parameter Header. See Section 5.4.
DMA_SEG DMA_OFF	The DMA segment and offset define the address of the data buffer. The DMA address may reside anywhere within the address space of the microprocessor. If the disk controller for the specified drive can only transfer data to and from a restricted address area, IO_READ and IO_WRITE must block move the data between the DMA address and this restricted area before a write or following a read operation.
RET_SEG RET_OFF	The BDOS return segment and offset define the RETURN FAR address from the XIOS to the BDOS.
RET_ADR	The local return address returns to the ENTRY routine in the example XIOS.

5.3 MULTISECTOR OPERATIONS ON SKEWED DISKS

This section describes how to optimize performance on skewed disks with multisector I/O requests. Sector skewing is a method of logically numbering the sectors on a track so that they are not sequential. The example 8-inch disk format skews the sectors by a factor of 6. The 5.25-inch disk format has no skew.

For a disk format that uses a skew table, optimize multisector transfer times by first translating each logical sector number into its physical sector number. Next, create a table of each sector's DMA address. The physical sector provides the index to the sector's DMA address (see Figure 5-1). In this way, the DMA address table sorts the requested sectors according to their physical locations on the track allowing them to be transferred in as few disk rotations as possible. The data from each sector must be separately transferred to or from its proper DMA address.

Sector Indexes	Associated Physical DMA Address
00	DMA_ADDR_0
01	DMA_ADDR_1
.	.
.	.
.	.
N	DMA_ADDR_N

Figure 5-1. DMA Address Table for Multisector Operations

When the sector number exceeds the last physical sector of the current track during a multisector data transfer, increment the track number and reset the sector number to 0. You can then complete the operation for the balance of sectors specified in the IO_READ or IO_WRITE function call. See Listing 5-2 in the description of IO_READ.

If an error occurs during a multisector transfer, the XIOS should return the error immediately to terminate the read or write BDOS function call.

Listing 5-5 contains a coding example for multisector unskewing. The routine begins by calling IO_SELDSK for the DPH address. If the address is invalid (equal to zero), the routine returns with an error.

Next, the routine gets the translation table address, the number of sectors per track, and the physical shift factor from the Disk Parameter Header (DPH) and Disk Parameter Block (DPB) and stores them in local variables. This information is used to compute the physical record size and initialize the DMA address table with 0FFFFH word values. Notice that the DMA table is one word greater than the number of sectors per track, in case the sectors index is relative to 1 for that particular drive. If the multisector count is zero, the routine returns an error. Otherwise, the sector number is compared to the number of sectors per track to determine if the track number should be incremented and the sector number set to zero. If this is the case, the sectors for the current track are transferred, and the DMA address table is reinitialized before the next tracks are read or written.

The current sector number is moved into AX and same_trk checks the offset of translation table's address. If the offset value is zero, no translation table exists and translation is not performed. no_trans translates the sector number for use as an index into the DMA address table. The current DMA address, incremented by the physical sector size for multisector operations, is stored in the table for use by the rw_sects routine. Local values, beginning with i, are initialized for the various parameters needed by the disk hardware and the disk driver routine is called.

Listing 5-5. Multisector Unskewing

```
;*********************************************************
;*   DISK I/O EQUATES
;*********************************************************

xlt         equ      0          ;translation table offset in DPH
dpb         equ      8          ;disk parameter block offset in DPH
spt         equ      0          ;sectors per track offset in DPB
psh         equ      15         ;physical shift factor offset in DPB

;*********************************************************
;*   DISK I/O CODE AREA
;*********************************************************

read_write:                ;unskews and reads or writes multisectors
;           input:         SI = read or write routine address
;           output:        AX = return code

            mov cl,drive
            mov dl,1
            call seldsk                        ;get DPH address
            or bx,bx! jnz dsk_ok               ;check if valid
ret_error:
              mov al,1                         ;return error if not
              ret
dsk_ok:
            mov ax,xlt[bx]
            mov xltbl,ax                       ;save trans table address
            mov bx,dpb[bx]
            mov ax,spt[bx]
            mov maxsec,ax                      ;save maximum sector/track
            mov cl,psh[bx]
            mov ax,128
            shl ax,cl                          ;compute phys record size
            mov secsiz,ax                      ; and save it
            call initdmatbl                    ;initialize dma offset table
            cmp mcnt,0
            je ret_error
```

Listing 5-5. (Cont'd)

```
rw_1:
              mov ax,sector                     ;sector < max sector/track?
              cmp ax,maxsec! jb same_trk
                call rw_sects                   ; no, r/w sectors on track
                call initdmatbl                 ;reinit dma offset table
                inc track                       ;next track
                xor ax,ax
                mov sector,ax                   ;initialize sector to 0
same_trk:
              mov bx,xltbl                      ;get trans table address
              or bx,bx! jz no_trans             ;if xlt <> 0
                xlat al                         ; translate sector number
no_trans:
              xor bh,bh
              mov bl,al                         ;sector # used as the index
              shl bx,1                          ; into the dma offset table
              mov ax,dmaoff
              mov dmatbl[bx],ax                 ;save dma offset in table
              add ax,secsiz                     ;increment dma offset by the
              mov dmaoff,ax                     ; physical sector size
              inc sector                        ;next sector
              dec mcnt                          ;decrement multisector count
              jnz rw_1                          ;if mcnt <> 0, store next
                                                ; sector dma

rw_sects:                                       ;r/w sectors in dma table
;--------
            mov al,1                            ;preset error code
            xor bx,bx                           ;initialize sector index
rw_s1:
            mov di,bx
            shl di,1                            ;compute index into DMA table
            cmp word ptr dmatbl[di],0ffffh
            je no_rw                            ;nop if invalid entry
              push bx! push si                  ;save index and routine addr
              mov ax,track                      ;get track # from IOPB
              mov itrack,ax
              mov isector,bl                    ;sector # is index value
              mov ax,dmatbl[di]                 ;get dma offset from table
              mov idmaoff,ax
              mov ax,dmaseg                     ;get dma segment from IOPB
              mov idmaseg,ax
              call si                           ;call read/write routine
              pop si! pop bx                    ;restore routine addr, index
              or al,al! jnz err_ret             ;if error occurred return
```

Listing 5-5. (Cont'd)

```
no_rw:
            inc bx                              ;next sector index
            cmp bx,maxsec                       ;if not end of table
            jbe rw_s1                           ; go read/write next sector
err_ret:
            ret                                 ;return with ercode in AL

initdmatbl:             ;initialize DMA offset table
;----------
            mov di,offset dmatbl
            mov cx,maxsec                       ;length = maxsec+1 sect no.
            inc cx                              ; may be relative to 0 or 1
            mov ax,0ffffh
            push es                             ;save UDA
            push ds! pop es
            rep stosw                           ;initialize table to 0ffffh
            pop es                              ;restore UDA
            ret

;*****************************************************
;*   DISK I/O DATA AREA
;*****************************************************

xltbl       dw          0          ;translation table address
maxsec      dw          0          ;max sectors per track
secsiz      dw          0          ;sector size
dmatbl      rw          50         ;dma address table

;-----------------------------------------------------
```

5.4 DISK PARAMETER HEADER

It is recommended that disk structures (DPH, DPB, CSV and DDSC) are initialized and allocated at run time rather than at generation time. This permits BDOS more economic use of memory.

Each disk in the system must have a Disk Parameter Header (DPH). The DPH contains information about the drive and provides a scratchpad for certain kernel data. Figure 5.2 shows the DPH format; Table 5-3 describes the DPH fields.

00H	XLT	RESERVED	MF	RESERVED
08H	DPB	CSV	RESERVED	RESERVED
10H	RESERVED	DDSC		

Figure 5-2. Disk Parameter Header

Table 5-3. Disk Parameter Header Fields

Field	Description
RESERVED	All reserved fields must be initialized to 0s in the XIOS source (for static DPHs) or during DPH creation (for dynamic DPHs).
XLT	Translation Table Address: Set with the address of the vector for logical-to-physical sector translation. Set to 0000H if the physical and logical sector numbers are the same (no sector translation). Disk drives with identical sector skew factors can share the same translation tables. This address is not referenced by the BDOS. It is intended for use only by XIOS routines. The translation table usually contains one byte per physical sector. If the disk has more than 256 sectors per track, the sector translation must consist of two bytes per physical sector. Use the track address to compute the head number for disks with multiple heads.
MF	Media Flag: This field indicates whether the drive's door is closed (00H) or has been opened since the last access (0FFH). The Media Flag is set to zero by the BDOS when the drive is logged in. The XIOS must set this flag to 0FFH if it detects that the drive door has been opened. It must also set the global door open flag in the XIOS Header at the same time. If the flag is set to 0FFH, the BDOS checks for a media change before performing the next file operation on that drive. Note that the BDOS checks this flag when first making a system call and never during an operation. If the BDOS determines that the drive contains a new disk, it logs out the drive and resets the MF field to 00H. **Note:** The use of door open interrupts is mandatory on Concurrent. If your disk hardware does not produce interrupts on disk changes then use a simple-time out of about 2 or 3 seconds following any disk access before setting the door open flag. This can enhance a removable disk's performance to that of a permanent drive.
DPB	Disk Parameter Block: Set to the address of the drive's Disk Parameter Block. Section 5.5 describes the Disk Parameter Block. See Section 5.8 and the IO_SELDSK function in Section 5.1 for related information.
CSV	Checksum Vector: Set with the segment address of a scratch area used to detect a media change. Concurrent checksums the root directory and FAT to detect a disk change. The size of this table has to be at least (DPB_CKS + (DPB_NFATRECS * 2)) bytes. This area is not required for permanent media where (DPB_CKS = 8000h). Place FFFFH in here if it is required for GENSYS to pre-allocate this area. NOTE: in this case the DPB field must point to the largest possible media requirement.
DDSC	The address of a segment that is used internally by the BDOS to keep track of disk space allocation. Its size has to be at least (3 + (DPB_NFATRECS/8))) paragraphs for 16- bit FAT media and (3 + (DPB_NCLSTRS >> (DPB_PSH+6))) paragraphs for 12-bit FAT media. This entry must be zero filled if allocated dynamically by the XIOS. Place FFFFh in here if it is required for GENSYS to preallocate this area. NOTE: in this case the DPB field must point to the largest possible media requirement.

5.5 DISK PARAMETER BLOCK

Disk Parameter Blocks (DPBs) define the characteristics of disk drives. A Disk Parameter Header (DPH) points to a DPB to give the BDOS information on how to access a particular disk drive. Drives with the same characteristics can have the same DPB.

Note: When a drive supports both CP/M and DOS media, the IO_SELDSK routine must determine the type of media currently in the drive and return a DPH with a pointer to a DPB with the correct values. For CP/M media, use the standard DPB shown in Figure 5-3. For DOS media, use the extended DPB shown in Figure 5-4. Standard DPB fields are described in Table 5-4; extended DPB fields in Table 5-5. Use the worksheet in Section 5.5.1 to calculate a value for each DPB field.

00H	SPT		BSH	BLM	EXM	DSM		DRM
08H	DRM	ALO	AL1	CKS		OFF		PSH
10H	PRM							

Figure 5-3. Standard Disk Parameter Block Format

Table 5-4. Standard Disk Parameter Block Fields

Field	Description
SPT	Sectors Per Track: Set to the total number of physical sectors per track.
BSH	Block Shift Factor: Set to the value appropriate to your allocation block size. The BDOS used BSH to calculate a block number from a given logical record number by shifting the record number BSH bits to the right.
BLM	Block Mask Factor: Set to the value appropriate to your allocation block size. The BDOS uses BLM to calculate a logical record offset within a given block by masking the logical record number with BLM.
EXM	Extent Mask: Set to the the maximum number of 16K logical extents contained in a single directory entry. This value is a function of the allocation block size and the number of blocks.
DSM	Disk Storage Maximum: Set to the total number of allocation blocks for the drive, minus 1. This value times the allocation block size equals the total storage capacity of the drive. DSM must be less than or equal to 7FFFH. If the disk uses 1024-byte blocks, DSM must be less than or equal to 255.
DRM	Directory Maximum: Set to the total number of directory entries that can be kept in the allocation blocks reserved for the directory, minus 1. Each directory entry is 32 bytes long. The maximum number of blocks that can be allocated to the directory is 16, which determines the maximum number of directory entries allowed on the disk drive.

Table 5-4. (Cont'd)

Field	Description
AL0, AL1	Directory Allocation Vector: Set a bit, starting with the high order bit of AL0 and working successively to the low order bit of AL1, for each directory allocation block. This value is used to initialize the first 16 bits of the allocation vector built when a disk drive is logged in.
CKS	Checksum Vector Size: Set to the length, in bytes, of the directory checksum vector (CSV) addressed in the Disk Parameter Header. As open door detection is now mandatory the value should be 8000H plus one byte for each 4 directory entries (or 128 bytes) on the drive. A checksum vector is required for removable media only. Set CKS to 8000H for permanent drives. This indicates to the BDOS that no checksumming is necessary for this drive.
OFF	Track Offset: Set to the number of reserved tracks at the beginning of the disk. OFF is equal to the zero-relative track number on which the directory starts. It is through this field that more than one logical disk drive can be mapped onto a single physical drive. Each logical drive has a different Track Offset and all drives can use the same physical disk drivers.
PSH	Physical Record Shift factor: Set to the value appropriate to the drive's physical record size. The BDOS uses PSH to calculate the physical record number from the logical record number. The logical record number is shifted PSH bits to the right to calculate the physical record. Note that in this context, physical record and physical sector are equivalent terms.
PRM	Physical Record Mask: Set to the value appropriate to the drive's physical record size. The BDOS uses PRM to calculate the logical record offset within a physical record by masking the logical record number with the PRM value.

Listing 5-6 contains an assembly language definition of a DPB. Note that the parameter values in this listing are for a doubled-sided diskette.

Listing 5-6. DPB Definition

```
;*****************************************************************
;*   DPB Definition
;*****************************************************************

perm        equ         8000h
spt         equ         word ptr 0
bsh         equ         byte ptr 2
blm         equ         byte ptr 3
exm         equ         byte ptr 4
dsm         equ         word ptr 5
drm         equ         word ptr 7
al0         equ         byte ptr 9
al1         equ         byte ptr 10
cks         equ         word ptr 11
off         equ         word ptr 13
psh         equ         byte ptr 15
prm         equ         byte ptr 16

; Double-sided floppy parameter block:

dpbd$       dw                                  ; sectors per track
            db          4                       ; block shift
            db          15                      ; block mask
            db          1                       ; extnt mask
            dw          157                     ; disk size in 2k blocks
                                                ; less offset track(s)
            dw          63                      ; directory max
            db          10000000b               ; alloc0
            db          0                       ; alloc1
            dw          16 + perm               ; check size and
                                                ; door-open simulator
            dw          1                       ; offset
            db          2                       ; phys rec shift
            db          3                       ; phys rec mask

;-----------------------------------------------------------------
```

Figure 5-4 shows the extended DPB; Table 5-5 describes its fields.

00H	EXTFLAG		NFATS		NFATRECS		NCLSTRS	
08H	CLSIZE		FATADD		SPT		BSH	BLM
10H	EXM	DSM		DRM		ALO	AL1	CKS
18H	CKS	OFF		PSH	PHM			

Figure 5-4. Extended Disk Parameter Block Format

Table 5-5. Extended Disk Parameter Block Fields

Field	Description
EXTFLAG	Extended DPB Flag: Set to 0FFFFH for DOS media with a 12-bit File Allocation Table (FAT). Set to 0FFFEH for DOS media with a 16-bit FAT. For CP/M media, the first field in the DPB is SPT indicating that the DPB is not extended.
NFATS	Number of FATs: Set to the number of FATs contained on the DOS disk. Multiple copies of the FAT can be kept on the disk as a backup if a read or write error occurs.
NFATRECS	Number of FAT Records: Set to the number of physical sectors in the FAT.
NCLSTRS	Number of Clusters: Set to the number of clusters on the DOS disk. Cluster 2 is the first data cluster to be allocated following the directory; cluster NCLSTRS-1 is the last available cluster on the disk.
CLSIZE	Cluster Size: Set to the number of bytes per data cluster. This value must be a multiple of the physical sector size.
FATADD	FAT Address: Set to the physical record number of the first FAT on the DOS disk.

The remainder of the fields in the extended DPB are the same as their standard DPB equivalent. Note, however, that for DOS media, the EXM value must be 00H, and AL0 and AL1 are not used.

Listing 5-7 contains an assembly language definition of an extended DPB. Note that the parameter values in this listing are for a double-sided, nine sector DOS diskette.

Listing 5-7. Extended DPB Definition

```
;*********************************************************
;*  Extended DPB Definition
;*********************************************************

perm           equu     8000h
extflag        equ      word ptr 0
nfats          equ      word ptr 2
nfatrecs       equ      word ptr 4
nclstrs        equ      word ptr 6
clsize         equ      word ptr 8
fatadd         equ      word ptr 10
spt            equ      word ptr 12
bsh            equ      byte ptr 14
blm            equ      byte ptr 15
exm            equ      byte ptr 16
dsm            equ      word ptr 17
drm            equ      word ptr 19
al0            equ      byte ptr 21
al1            equ      byte ptr 22
cks            equ      word ptr 23
off            equ      word ptr 25
psh            equ      byte ptr 27
prm            equ      byte ptr 28

;FAT ID = FD, double-sided, 9 sector format (DOS 2.0 ONLY):
pcdpb9D    dw       0FFFFh               ; extended DPB flag
           dw       2                    ; NFATS
           dw       2                    ; NFATRECS
           dw       356                  ; NCLSTRS
           dw       512*2                ; CLSIZE
           dw       1                    ; FATADD
           dw       9                    ; sectors per track
           db       3                    ; CP/M block shift
           db       7                    ; CP/M block mask
           db       0                    ; CP/M extnt mask
           dw       359                  ; CP/M disk size in 1k blocks
                                         ; less offset track(s)
           dw       111                  ; dir max (ROOT DIR SIZE)
           db       0                    ; CP/M alloc0
           db       0                    ; CP/M alloc1
           dw       28 + perm            ; check size and
                                         ; door-open simulator
           dw       0                    ; offset
           db       2                    ; phys rec shift
           db       3                    ; phys rec mask
;---------------------------------------------------------
```

5.5.1 Disk Parameter Block Worksheet

The worksheet below is provided to help you create a Disk Parameter Block for your disk hardware. Instructions for calculating the fields common to both DPBs are presented first. The instructions for the extended DPB fields follow.

<A> **Allocation Block Size**
Concurrent allocates disk space in a unit referred to as an Allocation Block. The Allocation Block is the minimum allocation of disk space given to a file. Legal values are 1024 (400H), 2048 (800H), 4096 (1000H), 8192 (2000H), and 16384 (4000H) decimal bytes. Note that disks larger than 256K bytes require at least 2048-byte Allocation Blocks.

Note: For DOS media, use the Cluster Size (CLSIZE), <V>, instead of the Allocation Block size in the calculations below.

<B> **BSH -- Block Shift Factor**

<C> **BLM -- Block Mask Factor**
Table 5-6 lists values for BSH and BLM for each value of <A>.

Table 5-6. BSH and BLM Values

<A>	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

Note: For clusters of 512 and 1024, use a BSH value of 3 and a BLM value of 7.

<D> **Total Allocation Blocks**
To determine the total number of Allocation Blocks on the disk drive, divide the total data storage capacity of the disk by the Allocation Block Size. To get the total data storage capacity, multiply the total number of tracks on the disk minus those reserved for the operating system by the number of sectors per track and then physical sector size. If the result is not a whole number, round down to the next lowest integer.

<E> **DSM -- Disk Storage Max field**
The value of DSM is the total number of Allocation Blocks relative to zero or <D> - 1.

Note: The product of <A>*(DSM+1) is the total number of data bytes the drive holds. This value plus the amount of space in the tracks reserved for the operating system must be within the capacity of the physical disk.

<F> **EXM -- Extent Mask field**
For CP/M media, obtain the value of EXM from the values listed for <A> and <E> in Table 5-7. EXM must be zero for DOS media.

Table 5-7. EXM Values

<A>	If <E> is less than 256	If <E> is greater than or equal to 256
1,024	0	* N/A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

*
N/A = not applicable

<G> **Directory Blocks**
The number of Allocation Blocks reserved for the directory must be between 1 and 16.

<H> **Directory Entries per Block**
Given the Allocation Block size, <A>, use Table 5-8 to determine the number of directory entries per Directory Block.

Table 5-8. Directory Entries per Block Size

<A>	Number of Entries
1,024	32
2,048	64
4,096	128
8,192	256
16,384	512

<I> **Total Directory Entries**
Multiply <G> by <H> to determine the total number of Directory Entries.

<J> **DRM -- Directory Max field**
Determine DRM by subtracting 1 from <I>; the DRM field must contain this value at run time.

<K> **AL0, AL1 -- Directory Allocation vector 0, 1 fields**
Determine values for AL0 and AL1 according to the number of Directory Blocks, <G>, as listed in Table 5-9.

Note that DOS disks do not use these fields.

Table 5-9. AL0, AL1 Values

<G>	AL0	AL1	<G>	AL0	AL1
1	80H	00H	9	0FFH	80H
2	0C0H	00H	10	0FFH	0C0H
3	0E0H	00H	11	0FFH	0E0H
4	0F0H	00H	12	0FFH	0F0H
5	0F8H	00H	13	0FFH	0F8H
6	0FCH	00H	14	0FFH	0FCH
7	0FEH	00H	15	0FFH	0FEH
8	0FFH	00H	16	0FFH	0FFH

<L> **CKS -- Checksum vector size**
Set the size of the Checksum Vector with one of the following values:

* If the drive media is permanent, set CKS to 8000H.

* If the drive media is removable and the Media Flag is implemented (open door can be detected through interrupt), set CKS to: (((<I>-1)/4)+1)+ 8000H.

<M> **OFF -- Offset field**
Set to the number of tracks that are reserved at the beginning of the physical disk. The BDOS automatically adds this number to the value of TRACK in the IOPB. Set OFF to skip reserved operating system tracks and to partition a large disk into smaller logical drives.

<O> **Physical Sector Size**
Specify the disk drive's physical sector size. Note that the physical sector size must be greater than or equal to 128 and less than 4096 or the Allocation Block Size, <A>, whichever is smaller. Typically, this value is the smallest unit that can be transferred to or from the disk.

<P> **PSH -- Physical Record Shift Factor <Q>**
PRM -- Physical Record Mask
Set the values of PSH and PRM according to the physical sector size, <O>, as listed in Table 5-10.

Table 5-10. PSH and PRM Values

<O>	PSH	PRM
128	0	0
256	1	1
512	2	3
1024	3	7
2048	4	15
4096	5	31

<R> **EXTFLAG -- Extended DPB Flag**
Set EXTFLAG to 0FFFFH for DOS media with 12-bit FATs; to 0FFFEH for DOS media with 16-bit FATs.

<S> **NFATS -- Number of File Allocation Tables**
Set to the number of FATs on the disk currently in the drive.

<T> **NFATRECS -- Number of FAT Records**
Set to the number of physical sectors in the FAT. Calculate this value from the number of clusters, <U>, and the physical sector size, <O>, using the following formula:

```
<T> := (<U>*1.5)+(<O>-1)/<O>
```

<U> **NCLSTRS -- Number of Clusters**
Set to the number of clusters on the DOS disk.

<V> **CLSIZE -- Cluster Size**
Set to the number of bytes per cluster.

<W> **FATADD -- File Allocation Table Address**
Set to the physical sector number of the first FAT on the DOS disk.

5.5.2 Disk Parameter List Worksheet

<A> Allocation Block Size ____________

<B> BSH field ____________

<C> BLM field ____________

<D> Total Allocation Blocks ____________

<E> DSM field ____________

<F> EXM field ____________

<G> Directory Blocks ____________

<H> Directory Entries per Block ____________

<I> Total Directory Entries __________

<J> DRM field __________

<K> AL0,AL1 fields __________

<L> CKS field __________

<M> OFF field __________

<O> Physical Sector Size __________

<P> PSH field __________

<Q> PRM field __________

<R> EXTFLAG field (extended DPB) __________

<S> NFATS field (extended DPB) __________

<T> NFATRECS field (extended DPB) __________

<U> NCLSTRS field (extended DPB) __________

<V> CLSIZE field (extended DPB) __________

<W> FATADD field (extended DPB) __________

5.6 Buffer and Hash Control Blocks

5.6.1 Buffer Control Blocks (BCBs)

The disk buffering algorithm in Concurrent DOS 6.0 and Concurrent 386 2.0 is different from that of previous versions.

These BCBs consist of 16 bytes immediately preceding the actual data buffers. BCBs are always shared for all drives and therefore must be big enough for the largest sector size of all drives. For example, if the largest sector size supported is 512 bytes (32/20h paragraphs), then 33/21h paragraphs have to be allocated for each buffer. The segment address of the first BCB is stored in BCB_ROOT (SYSDAT:009Ah). There have to be at least three buffers in the system for the BDOS to work correctly. More buffers are recommended for better performance. Buffers are used for storing data, directory and FAT sectors. Figure 5-5 shows the Buffer Control Block format.

00H	LINK	DRV	RESERVED
08H	RESERVED		
10H	DATA . . .		. . .

Figure 5-5 Buffer Control Block format

BCB_LINK contains the segment address of the next BCB in the list or 0000h for the last BCB.

BCB_DRV has to be initialized to 0FFh for all BCBs in the list.

BCB_RESRVD is used internally by the BDOS and should be initialized to all zeroes (13 bytes).

BCB_DATA is where the actual data for the sector starts.

5.6.2 Hash Control Blocks (HCB)

Concurrent DOS 6.0 and Concurrent 386 2.0 uses directory hashing to speed up directory operations. By storing information about file names in the most recently used directories, slow disk accesses and memory consuming buffers can be saved. Concurrent uses a list of Hash Control Blocks (HCBs) which are maintained in a linked list. The root of this list called HASHROOT and is located in SYSDAT at offset 00B0h. HASHROOT contains 0000h if hashing is not supported or the offset of the first HCB.

The number of directory entries hashed by one HCB is up to HCB_CLS*8 (if the cluster size is the maximum supported). As a general rule, about 512 to 1024 directory entries should be hashed for optimal performance (4-8 HCBs with 4 K clusters, 8-16 HCBs with 2 K clusters). This will consume 1-2 K of memory plus the space taken up by the HCBs. Figure 5-6 shows the Hash Control Block format.

00H	LINK		CLS	DRV	RESERVED			
08H	SEG							

Figure 5-6 Hash Control Block format

HCB_LINK contains the offset of the next HCB in SYSDAT or 0000h for the last HCB.

HCB_CLS contains the maximum cluster size supported, divided by 256. For example, if hashing is to be supported on disks with block sizes up to 4 Kb (1000h), HCB_CLS has to be set to 16 (10h). This field has to be set to the same value for all HCBs. If the BDOS finds that the cluster size of a drive is larger than the maximum indicated by HCB_CLS, it will not use hashing on that drive.

HCB_DRV is used by the BDOS to keep track of which drive this HCB is currently used for. This field has to be initialized to 0FFh for all HCBs.

HCB_RESRVD is used internally by the BDOS and doesn't need to be initialized.

HCB_SEG contains the segment address of a memory block that is used by the BDOS to keep track of the file name information. The size of this block has to be HCB_CLS paragraphs.

5.7 MEMORY DISK APPLICATION

A memory disk may be implemented if sufficient memory is available. Usually this will be I/O addressed, extended or expanded memory. In some cases a loadable DOS driver is available for specific hardware add-on boards. If a memory disk driver is required in the XIOS then it should be implemented as a simple DOS hard disk. In this case the sectors are simulated by areas of memory. The initialization code must clear the directory and FAT sectors and create a DPH and extended DPB for the drive. Drive M: is reserved for use as a memory disk. To simplify calculations it is usual to have 512 byte sectors and clusters. For a 1 Mbyte disk there would be 2048 sectors on a single track. A 12 bit FAT would be sufficient.

5.8 MULTIPLE MEDIA SUPPORT

Disk access is controlled by a number of data structures that describe various disk parameters. Some of these parameters are set in the code of the XIOS; others are supplied by GENSYS. When a particular disk drive supports more than one type of media, some of these parameters must be set at run time. This section explains how these parameters are set up and which ones must be changed at run time.

Each disk drive is described by a Disk Parameter Header (DPH). The DPH contains the addresses of several data structures required for disk access, including the Disk Parameter Block (DPB). The DPB describes the disk in detail, and includes such information as the size of the directory and the total storage capacity of the drive. The information in the DPB varies according to the density and/or format of the disk currently in the drive.

The DPH is located by the DPH pointers in the XIOS Header (see Section 3.1, "XIOS Header"). Values for the DPH fields may be hard-coded in the XIOS or calculated at run time.

GENSYS can supply those that are dependant on the allocation of memory (CSV, DDSC) if they are set to 0FFFFH in the DPH.

Auto Density Support describes the ability to use different types of media on the same drive. Some floppy disk drives can read many different disk formats. Auto Density Support enables the XIOS to determine the density of the disk when the IO_SELDSK function is called, and to detect a change in density when the IO_READ or IO_WRITE functions are called.

To implement Auto Density Support or support for both CP/M and DOS media, the XIOS disk driver must include a DPB for each disk format, or routines that generate proper DPB values automatically in real time. The disk driver must also determine the type and format of the disk when IO_SELDSK is called for the first time, set the DPH to address the DPB that describes the media, and return the address of the DPH to the BDOS. If IO_SELDSK is unable to determine the disk format, it can return a zero to indicate that the select operation was not successful. On all subsequent IO_SELDSK calls, the XIOS must continue to return the address of the same DPH. A return value of zero is allowed only on the initial IO_SELDSK call.

Once IO_SELDSK has determined the format of the disk, IO_READ and IO_WRITE assume the format is correct until an error is detected. If an XIOS function encounters an error and determines that the media has been changed to another format, it must abandon the operation and return 0FFH to the BDOS. This prompts the BDOS to make another initial IO_SELDSK call to reestablish the media format. XIOS routines must not modify the drive's DPH or DPB until the IO_SELDSK call is made. BDOS can also determine that the media has changed and make an initial IO_SELDSK call even though the XIOS routines have not detected any change.

5.9 SYSDAT DATA SPACE ALLOCATION

If your XIOS is to provide support for DOS drivers as described in Section 1.12, you must implement the SYSDAT_ALLOC XIOS function.

SYSDAT_ALLOC

Allocate buffer space from SYSDAT

Entry Parameters:

Register	AL:	1C(28)
	CX:	Number of bytes to allocate

Returned Values:

Register	AX:	Buffer offset 0 If requested number of bytes unavailable
	DX:	Number of bytes available (If AX returns 00)
	CX:	Preserved

SYSDAT_ALLOC is an optional XIOS function that allocates buffer space in the SYSDAT area to DOS drivers loaded at boot time.

This function first attempts to use the area of SYSDAT occupied by the XIOS INIT routine. SYSDAT_ALLOC then attempts to satisfy the request by using the memory space reserved by the user with the SETUP utility. See the User's Reference Guide for a description of SETUP.

If both areas of memory have been previously allocated or not enough space remains to satisfy the number of bytes requested, SYSDAT_ALLOC returns zero in AX and the amount of space remaining in these areas in DX.

EXPANDED MEMORY SUPPORT

Concurrent provides generic, hardware-independent support for expanded memory. This support is provided through the Memory Manager (MEM), Real-time Monitor (RTM), and Interceptor modules and four XIOS functions. Note that the support provided through MEM and RTM is not hardware or driver software specific.

This section describes the data structures and XIOS function calls associated with Concurrent's support for expanded memory.

Expanded Memory Terminology

* Expanded memory is memory not enabled on the machine's motherboard or add-on board. Expanded memory is not recognized by the processor.

* Memory paging is the technique of managing and swapping pages of physical memory in and out of logical address windows within the 1M byte address space to effectively increase the usable memory of the machine.

* Pages are regions (usually 16K bytes) of the physical memory on the board that can be dynamically readdressed by the memory paging hardware.

* Logical address windows are the areas in the address space of the processor into which pages can be mapped. This document sometimes refers to these areas as windows.

6.1 MEMORY PAGING ENVIRONMENTS

Concurrent's support of expanded memory accounts for two dissimilar environments.

The first environment is generic. In this environment, the XIOS is responsible for managing the memory mapping hardware and allocating physical pages of memory. A special case of the generic environment is the Concurrent 386 system where the MEM module directly sets the 386 memory management registers so the XIOS functions are not required. In addition Concurrent 386 implements LIM (int 67) support directly and supports applications making LIM memory calls.

The second environment is that of an IBM personal computer (or compatible) with a memory board that conforms to the AST™ superset of the Intel(R)/Lotus above-board standard, such as the RAMpage!™ board. In this environment, certain applications may call the Expanded Memory Management (EMM) Driver to perform their own memory management tasks completely outside of the operating system. This second environment is referred to as the "EMM environment."

6.1.1 Generic Environment

In the generic environment, the Supervisor and applications call the Memory Manager (MEM) for all memory requests. MEM passes these calls to the XIOS as memory page allocation and release requests. During process dispatching, the Real-time Monitor (RTM) generates XIOS calls to save the current state of the memory mapping hardware and to restore the previously saved hardware to its original state. (Concurrent 386 does not pass on the XIOS calls.)

Figure 6-1 illustrates the generic environment's interfaces.

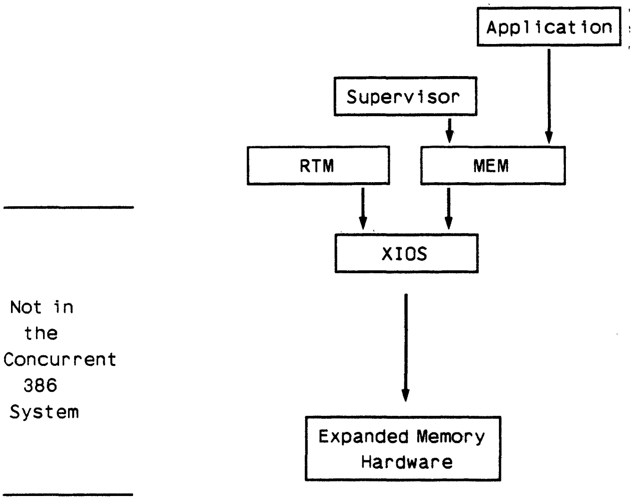


Figure 6-1. Generic Environment Memory Paging Interfaces

6.1.2 EMM Environment

In the EMM environment, the XIOS passes all of its memory page allocation and release requests along to the EMM Driver, which handles the page mapping hardware. Calls into the XIOS are translated to EMM Driver calls, and the EMM Driver is invoked with an Interrupt 67H.

An application running in the EMM environment may be making calls directly to the EMM Driver. The Interceptor module intercepts all calls to the EMM Driver so that the system can handle context switching and memory deallocation for aborted processes. To reserve some memory for system use, the Interceptor might not let an application know the total number of available memory pages.

The Interceptor's primary functions are creating and linking new Memory Page Allocation Descriptors (MPADs) as an application performs EMM Allocate calls. The Interceptor destroys these MPADs when the application performs an EMM Close call. These functions allow the Interceptor to track an application's calls to the EMM Driver for the system. The MPAD data structure is described below in Section 6.2.

Figure 6-2 illustrates the interfaces within the EMM environment.

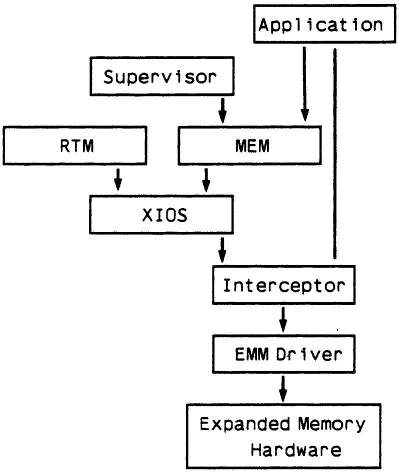


Figure 6-2. EMM Environment Memory Paging

6.2 EXPANDED MEMORY DATA STRUCTURES

Expanded memory support adds two new data structures to Concurrent. These are the Memory Window Descriptor and the Memory Page Allocation Descriptor. Both are structured as elements of linked lists, and both are five words long.

6.2.1 Memory Window Descriptor

The XIOS maintains one Memory Window Descriptor (MWD) for each logical address window of paged memory in the system. MWDs are arranged as a linked list pointed to by the Memory Window Descriptor Root (MWDR) located at offset 98H in the System Data Area (SYSDAT).

The MWD List is a static structure that can be initialized by GENSYS. It is used by the Memory Manager to locate memory for paged allocation. Figure 6-3 illustrates the format of an MWD. Table 6-1 lists and defines the MWD fields.

0	1	2	3	4	5	6	7	8	9
LINK		START		LENGTH		FLAGS		COUNT	

Figure 6-3. Memory Window Descriptor

Table 6-1. Memory Window Descriptor Fields

MWD	Field Description
LINK	Offset to next MWD; zero indicates end of list
START	Starting segment address of logical address window
LENGTH	Length of window in paragraphs (16 bytes)
FLAGS	Memory page attributes (to be defined)
COUNT	Number of processes currently using this window; 0FFFH if window is locked by a process

The MWD List describes the location of logical address windows that can be mapped to physical memory pages. If the MWD List has not been set up by GENSYS, the XIOS INIT routine must create it during system initialization. To create a complete MWD List, INIT first obtains Memory Descriptors from the linked list of unused Memory Descriptors pointed to by the MDUL field at offset 58H in SYSDAT DATA. It then links the newly created MWD's to the MWD List in address order.

If a default MWD List has been created by GENSYS, the INIT routine modifies the list to match the existing hardware configuration. It does this by unlinking unnecessary MWDs and returning them to the list of unused Memory Descriptors according to a procedure similar to that shown in the example XIOS for trimming the memory partition list.

6.2.2 Memory Page Allocation Descriptor

The Memory Page Allocation Descriptor (MPAD) is a dynamic structure created by MEM whenever paged memory is allocated to a process. MEM destroys the MPAD when the process frees the paged memory. MPADs are arranged as a linked list pointed to by the Memory Page Allocation Root (P_MPAR) field in the Process Descriptor of the process that owns this unit of paged memory. The P_MPAR field is located at offset 34H in the Process Descriptor. The Process Descriptor is defined in the Programmer's Guide.

The Interceptor creates and links a new MPAD when an application performs a successful allocation call directly to the EMM Driver. When the application makes a corresponding EMM Driver Close call, the Interceptor destroys the MPAD it had previously created.

Figure 6-4 shows the MPAD format. Table 6-2 describes the MPAD fields.

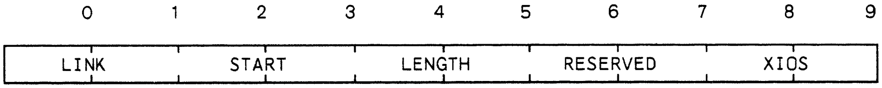


Figure 6-4. Memory Page Allocation Descriptor

Table 6-2. Memory Page Allocation Descriptor Fields

MPAD Field	Description
LINK	Offset address of next MPAD; zero indicates end of list
START	Starting segment address of allocation unit
LENGTH	Length of allocation in paragraphs (16 bytes)
XIOS	In the EMM Environment, this field stores the Process ID (PID) returned by the EMM Driver. In the generic environment, this field can be used as a pointer to a local XIOS data structure such as a linked list of words in which each word represents one physical page of expanded memory.

Note: Both the START and LENGTH fields of new MPADs created by the Interceptor contain zero values (the Interceptor has no knowledge of where the application will place its memory pages).

6.3 EXPANDED MEMORY FUNCTION CALLS

This section describes the four function calls that the XIOS uses to manage expanded memory. The MEM module calls two of these functions, IO_MPALLOC and IO_MPFREE, when a process allocates or releases pages of memory. The other two functions, IO_MPSAVE and IO_MPRESTORE, are called by the Real-time Monitor (RTM) when it suspends or returns a process to context.

All of the expanded memory function calls preserve registers DS, ES, SS and SP.

IO_MPALLOC

Allocate a bank of paged memory

Entry Parameters:
Register AL: 28H (40)
DX: Address of MPAD

Returned Values:
Register AL: 0 On Success
Error code on failure

MEM calls this function to allocate the physical pages of memory described in the referenced Memory Page Allocation Descriptor (MPAD). The MPAD's START field indicates where the memory pages are to be mapped within the logical address space; the LENGTH field indicates how many pages are to be mapped.

The XIOS should use the XIOS field of the MPAD to keep track of the physical pages associated with the allocation.

In the EMM environment, the XIOS must translate this function into two EMM calls, Allocate and Map, the first to acquire the necessary pages of physical memory, the second to map them into the appropriate logical address windows.

For an EMM Close call, the Interceptor deletes the MPAD only if it finds the START and LENGTH fields equal to zero (the MPAD is therefore one that it created).

If the allocation is unsuccessful for any reason, this function must return a nonzero value in AL.

IO_MPFREE

Free a bank of paged memory

Entry Parameters:
Register AL: 29H (41)
DX: Address of MPAD

MEM calls this function to release the physical pages of memory previously allocated through IO_MPALLOC. IO_MPFREE should use the MPAD_XIOS field to identify the physical pages involved.

In the EMM Environment, this call translates into an EMM Close call.

IO_MPSAVE

Save current state of paged memory hardware

Entry Parameters:
Register AL: 2AH (42)
DX: Address of MPAD List

The RTM calls IO_MPSAVE when it suspends the currently running process. This function causes the XIOS to preserve the current state of the memory mapping hardware for later restoration (IO_MPRESTORE).

If the memory mapping scheme involves restoring specific pages rather than the entire hardware state, this call may simply return without action.

IO_MPRESTORE

Restore previously saved hardware state

Entry Parameters:

Register	AL:	2BH (43)
	DX:	Address of MPAD List

The RTM calls IO_MPRESTORE to return the memory mapping hardware to the state it was in when the process that owns the referenced MPAD List was last in context.

The XIOS can perform this function by restoring the entire hardware state as saved by a previous IO_MPSAVE call, or by restoring the specific pages in each of the linked MPADs.

SECTION 7

PC/AT ROS Support

In order to improve performance and compatibility ROS interrupt support has been changed from previous revisions to give the XIOS greater control over handling the various functions.

Concurrent will intercept some of the ROS interrupts and emulate a limited sub-set of functions, the remaining functions will be passed onto the original interrupt service routines (ISRs). This means that the XIOS INIT routine should set all the interrupt vectors between 10h and 1Fh to point to XIOS code. These XIOS services routines should either handle the function requested or print an error message and terminate the current process. Obviously on a PC or compatible many of the functions can be passed on to the ROS.

The following ROS interrupts are intercepted by Concurrent.

INT 11h System Configuration
This function calls the PC_EQUIP XIOS function.

INT 12h Memory Size
The PCMODE returns the correct memory size based on the memory allocated to the current process.

INT 13h Disk I/O
Concurrent DOS uses the IO_INT13 XIOS function to support the Disk I/O ROS function.

INT 14h Serial I/O
Sub-functions 1 (Send Character), 2 (Get Character) and 3 (Status) are trapped by Concurrent and passed to the appropriate AUX devices. All other sub-functions are passed to the original ISR.

INT 16h Keyboard Services
Sub-functions 0 to 2 and 10h to 12h are emulated by Concurrent. All other sub-functions are passed to the original ISR. INT 16h is commonly used by clone manufacturers to access extra facilities provided by their hardware, ie Processor Speed Switching.

INT 17h Parallel Printer I/O
Sub-functions 0 (Print Character) and 2 (Status) are emulated by Concurrent all other sub-functions are passed to the original ISR.

7.1 XIOS Functions for ROS Video Support

IO_SCREEN (30) (not used by kernel)
IO_VIDEO (31) (not used by kernel)
IO_KEYBD (32) Used to indicate to XIOS the application requirements.

On Entry:-	AL:	020h (32)
	CL:	Bit 0: PC-Mode Keyboard Bit 1: 25 Line Support Bit 2: ANSI Support Bit 3: ROS Support Bit 4: Hardware Access
	DL:	Virtual Console Number
On Exit:-	AX:	0000
	BX:	Same as AX

Bit 0: PC-Mode Keyboard
This bit is set when a DOS process begins execution and is reset when it terminates. The XIOS uses this bit to control the keyboard decoding, ie. function key support is disabled when this bit is set.

Bit 1: 25 Line Support
When set this bit signifies that the current process requires a 25 line screen to function correctly. This bit is used by the example XIOS to disable the status line when a DOS application requiring all 25 lines is loaded.

Bit 2: ANSI Support
When set this bit signifies that the current process uses ANSI escape sequences. This bit can be used to select ANSI support in a serial terminal that supports several emulation modes.

Bit 3: ROS Support
When set this bit signifies that the current process uses the ROS video services software interrupt (INT 10).

Bit 4: Hardware Access
When set this bit signifies that the current process accesses the video hardware directly.

The PCMODE checks the ATTRIBUTES field in the CCB before loading a DOS process to check that the Virtual Console provides all the video services required by the DOS process.

7.2 XIOS Functions for ROS Disk Support

IO_INT13 (35) provides support for DOS interrupt 13 functions.

IO_INT13

DOS INT 13 Disk I/O

Entry Parameters:
Register AL: 23H (35)
DX: Offset of parameter structure

Returned Values:
Register AH: 00 Operation successful Status byte (if Carry Flag set, see Table 7-1)
AL: Number of sectors (on read, write, and verify operations) Status byte (on read disk status)
DS, BX, DX, CX: preserved

Parameter Structure: Offset

0	Number of sectors (BYTE)
1	Select byte 0 Reset disk system 1 Read disk system status into AL 2 Read sectors 3 Write sectors 4 Verify sectors 5 Format track
2	Offset of DMA buffer (WORD)
4	Sector number (1-17)
5	Track number (0 relative)
6	Physical drive number (0 relative)
7	Head number (0-1)
8	Segment of DMA buffer (WORD)
10	Reserved on entry, Flags on return

The DMA address may reside anywhere in the one megabyte address space of the processor. If the disk controller transfers data only to and from a restricted address area, the read or write functions must block-move the data between the DMA address and this restricted area after a read or before a write operation.

If the byte at offset 00 of the parameter structure contains 5 (format track), offsets 02 and 08 must point to a format block. The format block consists of one four-byte address field for every sector on the track to be formatted. The address field bytes contain the track number, head number, sector number, and the number of bytes per sector, respectively.

If IO_INT13 returns with the Carry Flag set, the status byte in AH indicates an error according to the values listed in Table 7-1.

Table 7-1. IO_INT13 Status Byte Values

Hex	Value
01	Bad command
02	Address mark not found
03	Disk write protected
04	Sector not found
08	DMA overrun
09	DMA crosses 64K boundary
10	CRC read error
20	Controller failure
40	Seek failure
80	Time out error

Note that IO_INT13 must update the parameter structure before returning.

7.3 XIOS Functions for ROS Keyboard Support

There are two functions described in this section: PC_KBD indicates whether or not PC Mode is active; PC_SHIFTS returns the keyboard shift status of a console operating in PC Mode.

PC_KBD

PC Mode Keyboard Switch

Entry Parameters:

Register	AL:	20H (32)
	CL:	1 Enable PC Mode 2 Disable PC Mode
	DL:	VC number

Returned Values:

Register	AX:	0 on success FFFFH on error

ES, DS, SS, SP: preserved

PC_KBD controls a variable that indicates when PC Mode is active. When PC_KBD is enabled, the console is running a PC DOS program and the keyboard's function keys and the 25th line on the screen must behave differently.

When PC_KBD is disabled, all non-ASCII keys are either ignored, initiate a screen switch, or return a string of characters (as does the sample XIOS). If PC_KBD is enabled, IO_CONIN must pass all 16-bit function key codes to the caller.

Many PC DOS applications use the 25th line of the display. The IO_STATLINE function (8) must not display the status line on a console that is in PC Mode. See Section 4.2, "Console I/O Functions," for information on IO_STATLINE.

The PC_KBD variable can also be used in the XIOS for any other functions that need to know if a console is in PC Mode.

When a Virtual Console is in PC Mode IO_CONIN (1) must return the full ROS character and scan code. IO_CONIN returns the ASCII code for characters in AL, and the scan code in AH.

PC_SHIFTS

Return Shift Status

Entry Parameters:
Register AL: 21H (33)
DL: VC number

Returned Values:
Register AL: Shift status bit map
ES, DS, SS, SP: preserved

PC_SHIFTS emulates subfunction 2 of DOS's interrupt 16. It returns a bit map that indicates the status of certain keys. This bit map is shown in Table 7-2.

Table 7-2. Keyboard Shift Status Bit Map

Bit	Meaning
7	Insert state active
6	Caps lock state toggled
5	Num lock state toggled
4	Scroll lock state toggled
3	Alternate shift key depressed
2	Control shift key depressed
1	Left shift key depressed
0	Right shift key depressed

7.4 XIOS Functions for ROS System Configuration Support

PC_EQUIP

Return Equipment Status

Entry Parameters:
Register AL:22H (34)

Returned Values:
Register AX: PC equipment bit map
ES, DS, SS, SP: preserved

PC_EQUIP emulates DOS interrupt 11 by returning the standard PC bit map that describes attached I/O devices. The bit meanings, from low to high order, are listed in Table 7-3. Note which values are to be interpreted as bit pairs or triplets.

Table 7-3. DOS Equipment Status Bit Map

Bit	Meaning
0	IPL from floppy - Set when system has floppy drives
1	Not used
3, 2	RAM size on system board:
	00 16K
	01 32K
	10 48K
	11 64K
5, 4	Initial video mode:
	01 40 x 25 color
	10 80 x 25 color
	11 80 x 25 monochrome
7, 6	Number of floppy disk drives:
	00 1 drive
	01 2 drives
	10 3 drives
	11 4 drives
8	Not used
11- 9	Number of RS232 ports
12	Game I/O attached
13	Not used
14,15	Number of parallel printers

XIOS TICK INTERRUPT ROUTINE

The XIOS Tick Interrupt routine forces process dispatches and "wakes up" the CLOCK RSP. To perform these functions, the routine continually sets (makes DEV_SETFLAG calls on) two reserved flags:

* On every interrupt, the routine sets the Flag #1, the system tick flag, if the TICK flag in the XIOS Header is 0FFH.

* Once every second, the routine sets Flag #2, the second flag.

If necessary, the routine must also reset the hardware timer interrupt.

After setting the flags, the Tick Interrupt routine must make a Jump Far call to the dispatcher entry point. This forces a dispatch. The double-word pointer to the dispatcher entry point is located at 0038H in the SYSDAT DATA. See Section 3.5, "Interrupt Devices," for more information on writing XIOS interrupt routines.

The system tick frequency determines the dispatch rate for compute-bound processes. If the frequency is too high, the system overhead required with each dispatch slows down system performance. If the frequency is too low, compute-bound processes monopolize the CPU resource for longer periods. The recommended time period for the system tick on systems with a power frequency of 60 Hz is 16.67 milliseconds. When operating on 50 Hz power, use a 20-millisecond period.

The CLOCK RSP calls DEV_WAITFLAG on Flag #2 (the second flag) to maintain the system time and day in the TOD structure in SYSDAT. When CLOCK "wakes up," it updates the TOD structure and calls the IO_STATLINE XIOS function to update the status line. If the system has more than one physical console, one physical console is updated each second. Thus if your system has four physical consoles, each one is updated once every four seconds.

DEBUGGING THE XIOS

The distributed source for Concurrent is based on the PC/XT/AT/PS2 hardware and clones of that hardware. Debugging techniques will vary depending on the extent of your modifications to this source and hardware differences.

In all cases it is recommended that you use an AT/PS2 or clone to generate the system. First it is most important to check that you can generate the distributed system. Copy all the source files and tools to your hard disk maintaining the sub-directory structure. It is advisable to check that there is no name conflict with any files that may already exist on your hard disk. Follow any instructions provided on the README file if it is present. This file reflects any special version dependant instructions or corrections to this guide.

Re-assemble the XIOS and link it using the batch and makefiles provided. Generate the system using GENSYS. Compare this new CCPM.SYS file with the distributed file. They should be the same. If there is a difference in length check again that no errors occurred during the build. Check that you are comparing the bootable CCPM.SYS supplied with the sources and not another version or release. If the files are the same length but have a difference it may be due to a field installable patch. Check the README file for details of patches.

The appropriate technique for debugging the XIOS depends on the extent of your modifications. We will take 3 possible scenarios.

1) Hardware compatible with PC/XT/AT/PS2, modification to GENSYS parameters required.

This system should not require debugging. Test to see that the requested changes have been made. The SYSTAT utility is useful for this. Use SID-86 to check SYSDAT and CCPM.SYS values if there is a problem.

2) Hardware compatible with PC/AT/PS2 or Clone hardware, Alteration required to XIOS to improve performance, fix bug or adapt for addon hardware.

Restrict the changes to one area/module at a time. Ensure that the changes will not prohibit rebooting the system. If necessary use a flag in the XIOS to switch in the modified code. The system can then be rebooted and the code enabled after SID-86 etc are loaded. Check the changes by examination of the XIOS variables using SID-86 and your XIOS symbols. Use debug code in your source to store intermediate results. It is possible to step through portions of the XIOS code either by calling directly from a test programme to the XIOS or by setting the SID-86 registers directly. REMEMBER that the XIOS is being used for all processes so confine yourself to specific code areas that do not affect the basic operating system. If this is insufficient then you will need to use an ICE or ROM Monitor or ROM dump facility.

3) Hardware incompatible with PC/XT/AT/PS2. New XIOS required.

In this case it will be necessary to build the XIOS incrementally as previously described. Use code from the example XIOS but enable the simple features first. Begin testing with all I/O devices in polled mode, all interrupts, including the system Tick, disabled. NOTE: the PIN processes calls the IO_CONIN and runs at higher priority than the TMP. It will be necessary to reduce PIN priority to enable console output from the TMP until the CONIN is changed from polling to flag_wait. Once the XIOS functions are implemented using device polling, change them to interrupt-driven I/O devices and test them one at a time.

Debug the Tick Interrupt routine last. The initial system can run without a Tick Interrupt, and console and disk I/O routines are much easier to debug. Disable the Tick Interrupt by changing the routine to execute an IRET instead of a JMPF to the dispatcher and not allowing it to set Flags #1 (system tick) and #2 (second tick). As long as the Tick Interrupt is disabled, you have no way to force CPU-bound tasks to dispatch. In fact, if problems arise after the Tick Interrupt is implemented, it is often helpful to simplify the environment by disabling the effects of the Tick Interrupt.

Note: Until the Tick interrupt is implemented, P_DELAY cannot be called. Instead, you must use an assembly language time-out loop for each instance of P_DELAY. Be sure to replace these time-outs with P_DELAY system calls after the tick routine is implemented and debugged. See the MOTOR_ON routine in the example XIOS for more details.

To debug this system it will probably be necessary to have hardware assistance in the form of an In Circuit Emulator. This can be used to trap Concurrent after it boots as the XIOS is initialised and as each function is called. Most ICE systems allow the CCPM.SYS file to be loaded directly into the target memory from the host system.

BOOTSTRAP ADAPTATION

This section describes an example bootstrap procedure for Concurrent implemented on the IBM PC/XT/AT/PS2 personal computers. You can customize this example for different hardware environments.

10.1 The Bootstrap Loader on Floppy disks.

Sector 0 of the floppy disk is used as a bootstrap loader. It contains enough code to read the directory and FAT information on the disk and read the CCPM.SYS file into the correct memory area and jump to it.

The Boot Sector is brought into memory on reset or power-on by the PC's ROM monitor. The Boot Sector then reads in Sector 0 and transfers control to the Loader.

The Loader depends on software interrupt 13 to the ROS to perform the physical disk I/O and hence is machine independent. The Loader is written to the disk when the floppy disk is formatted using the Concurrent FORMAT utilities.

OEMs may use the DR FORMAT utility to prepare disks with this loader but will need to provide support for interrupt 13 in their ROM.

10.2 The Bootstrap Loader on Hard disks.

In the case of hard disks the utility FDISK prepares the disk partition table and the logical partitions of the disk. One of these partitions is usually designated bootable. In the case of the hard disk a multi-stage bootstrap loader is used. Again it uses interrupt 13 to perform the physical I/O.

10.3 OTHER BOOTSTRAP METHODS

Many departures from the previously described methods of bootstrap loader operation and construction are possible. All are dependent upon the hardware environment and goals of the system implementor.

The Boot Sector can be eliminated if the system ROM (or PROM) can read in the CCPM.SYS file at reset.

Concurrent may be placed in ROM and executed from ROM itself as discussed previously in section 2.

Another possibility is that the CCPM.SYS file is read by the ROM over a network from another server.

10.4 ORGANIZATION OF CCPM.SYS

The CCPM.SYS file generated by GENSYS and read by the Loader consists of the required .CON files and any included .RSP files (see Section 2). CCPM.SYS is prefixed by a 128-byte CMD Header Record, which contains the two Section Descriptors shown in Figure 10-1.

S_TYPE	S_LENGTH	S_A-BASE	S_MIN	S_MAX
01H	xxxxx	SSSSH	xxxxx	xxxxx
02H	xxxxx	DDDDH	xxxxx	xxxxx

Figure 10-1. Section Descriptors - CCPM.SYS Header Record

The first Section Descriptor (S_TYPE 01H) represents the CCPM.SYS Code Group; (it also contains RSP and Network Data). The second Section Descriptor (S_TYPE 02H) represents the SYSTEM Data Group. The Code Section Descriptor has an A-Base load address at paragraph SSSSH, or "paragraph:byte" address of 0SSSS:0000H. The A-Base value in the Data Group Descriptor is 0DDDD:0000.

CCPM.SYS is read into memory by the Loader beginning at the address given by Code Section A-Base (in the example shown above, paragraph address SSSSH), and control is passed to the Supervisor INIT routine when the Loader Program executes a Jump Far (JMPF) instruction to SSSS:0000H. The Supervisor INIT must be entered with CS set to the value of S_A-BASE in the Code Section Descriptor, IP equal to 0, and DS equal to the value of S_A-BASE in the Data Section Descriptor.

OEM DISK UTILITIES

A commercially viable Concurrent system requires OEM-implemented and supported utilities. Fundamental services provided by such utilities are disk formatting and verbatim copying. The code that performs these functions is hardware-specific and typically uses direct XIOS calls or goes directly to the hardware. This section lists the precautions you should observe when making direct XIOS calls and describes DOS-media directory contents.

Note: The DSKMAINT, FORMAT and DISKCOPY utility provided with Concurrent formats DOS media in several sector sizes and copies disks with the same format. DISKMAINT retains CP/M media formatting for the CP/M utility. Separate modules are provided for target machines with 5.25″ and 3.5″ drives. If you have different media to those already supported and cannot use DSKMAINT, use its source modules as models for your own format and copy utilities. Concurrent Sources are not included in this kit but are available separately.

11.1 DIRECT-DISK ACCESS PRECAUTIONS

Programs that bypass the BDOS to make direct hardware calls circumvent Concurrent's normal disk control mechanisms. In a multiuser or multitasking environment, this can have disastrous consequences. The format program must include procedures that protect the user from formatting while a background process is using the disk and ensure that other users do not get access to the disk during the operation.

The following list contains the procedures required to provide safeguards under Concurrent.

1. Confirm that the Operating system BDOS version is 6.0 or later. (Use the S_OSVER function.) The following steps apply only to this version of Concurrent.

2. Lock the disk drive using the DRV_LOCK function.
   The program can now safely perform the format and copy operations on the disk system, independent of the BDOS. Only exit from the program according to the sequence of tasks outlined in steps 3 through 4.

3. Set the login sequence number in each affected DPH to 0 to allow the disk system to be reset. When the disk system is reset, these drives are reset even if they are permanent. The login sequence field is 06h bytes from the beginning of the DPH.

4. Unlock the drive with the DRV_UNLOCK function.

5. Reset the Disk System with the DRV_ALLRESET function.

6. Terminate.

END-USER DOCUMENTATION

OEMs must be aware that the documentation supplied by Digital Research for the generic release of Concurrent describes only the example XIOS implementations. If you decide to change, enhance, or eliminate a function that impacts the Concurrent operator interface, you must also issue documentation describing the new implementation. This is best done by purchasing reprint rights to the Concurrent DOS system publications, rewriting them to reflect the changes, and distributing them along with the OEM-modified system.

One area that is highly susceptible to modification by the OEM is the Status Line XIOS function. Depending upon the implementation, it might be desirable to display different, more, or even no status parameters. The documentation supplied with Concurrent assumes that the Status Line function is implemented exactly like the example XIOS presented in this document.

Another area that you might want to change is the default or current login disk. At system boot time, the default system disk, as specified in the system GENSYS session, is automatically logged-in and displayed in the first system prompt. However, a startup command file, STARTNN.BAT, where NN is the Virtual Console number, can be implemented for each Virtual Console. This file can switch the default logged-in disk drive to any drive desired. However, the User's Guide and the User's Reference Guide assume that the prompt will show the system disk. For more information on startup files, see the User's Guide, User's Reference Guide, and Programmer's Guide.

REMOVABLE MEDIA

All disk drives are classified under Concurrent as having either permanent or removable media. Removable-media drives support media changes; permanent drives do not. Setting the high-order bit of the CKS field in the drive's DPB marks the drive as a permanent-media drive.

The BDOS file system makes two important distinctions between permanent and removable-media drives. If a drive is permanent, the BDOS always accepts the contents of physical record buffers as valid. It also accepts the results of hash table searches on the drive.

BDOS handling of removable-media drives is more complex. Because the disk media can be changed at any time, the BDOS discards directory buffers before performing most system calls involving directory searches. By rereading the disk directory, the BDOS can detect media changes. When the BDOS reads a directory record, it computes a checksum for the record and compares it to the current value in the drive's checksum vector. If the values do not match, the BDOS assumes the media has been changed, aborts the system call routine, and returns an error code to the calling process. Similarly, the BDOS must verify an unsuccessful hash table search for a removable-media drive by accessing the directory. The BDOS can detect a media change only by reading the directory.

Because of the frequent necessity of directory access on removable-media drives, there is a considerable performance overhead on these drives compared to permanent drives. Another disadvantage is that since the BDOS can detect media removal only by a directory access, inadvertantly changing media during a disk write operation results in writing erroneous data onto the disk.

If, however, the disk drive and controller hardware can generate an interrupt when the drive door is opened, another option for preventing media change errors becomes available. By using the following procedure, the performance penalty for removable-media drives is practically eliminated.

1. Mark the drive as permanent by setting the value of the CKS field in the drive's DPB to 8000H plus the total number of directory entries divided by four. For example, you would set the CKS to 8018H for a disk with 96 directory entries.

2. Write an Open Door Interrupt routine that sets the DOOR field in the XIOS Header and the MF (Media Flag) field in the DPH for any drive that signals an open door condition.

The BDOS checks the XIOS Header DOOR flag on entry to all disk-related XIOS function calls. If the DOOR flag is not set, the BDOS assumes that the removable media has not been changed. If the DOOR flag is set (0FFH), the BDOS checks the Media Flag in the DPH of each currently logged-in drive. It then reads the entire directory of the drive to determine whether the media has been changed before performing any operations on the drive. The BDOS also temporarily reclassifies the drive as a removable-media drive, and discards all directory buffers to force all subsequent directory- related operations to access the drive.

In summary, using the DOOR and Media Flag facilities with removable-media drives offers two important benefits. First, performance of removable-media drives is enhanced. Second, the integrity of the disk system is greatly improved because changing media can at no time result in a write error.

GRAPHICS IMPLEMENTATION

Concurrent can support graphics on any Virtual Console assigned to a physical console that has graphics capabilities. Graphics support is provided in the sample XIOS for GEM.

GEM drivers may perform their own hardware initialization to put a physical console in graphics mode. A graphics process that is in graphics mode can not run on a background console, because this causes the foreground console to change to graphics mode. Keep the following points in mind when writing an XIOS for a system that is to support graphics:

- The GEM driver will send an escape sequence (PC Clones: use a Software interrupt 10) when it wants to change a Virtual Console to graphics or alphanumeric mode. If the Virtual Console is in the background and graphics is requested, the XIOS must flagwait the process. If the Virtual Console is in the foreground, change the screen_mode field in the XIOS VC Structure and allow the process to continue. You must reserve at least one flag for each Virtual Console for this purpose. The VC Structure is described in Section 4.3.2.

- IO_SWITCH (7) must flagset any process that was flagwaited by when its Virtual Console is switched to the foreground.

- IO_STATLINE (8) must not display the status line on a console that is in graphics mode. This can be done by checking the screen_mode varible in the screen structure.

- GEM drivers should detect if they are running under Concurrent and attach to the appropriate auxiliary device for MOUSE input from a serial device. The GEM driver sources demonstrate the use of this technique.

80386 SPECIFIC INFORMATION

Figure C-1 V386PTR Structure Definition

0	IDT SEGMENT	IDT LIMIT
+4	TSS SEGMENT	TSS LIMIT
+8	MP TABLE	NPAGES
+12	PTBL SEGMENT	V386 LIM
+16	LIM BASE	LIM MAXSIZE
+20	EXCEP OFFSET	EXCEP SEGMENT
+24	PAGE FAULT OFFSET	PAGE FAULT SEGMENT
+28	HI MEM ROOT	READ ONLY MEM ROOT

IDT SEGMENT: Segment base address of Interrupt descriptor table (inserted by XIOS INIT). XIOS must allocate enough table space for all interrupt descriptprs. Hardware specific interrupt descriptors must be inserted into the table. The 80386 interrupt descriptors will be allocated by the O/S.

IDT LIMIT - Total length in bytes of Interrupt descriptor table (inserted by XIOS INIT).

TSS SEG - Segment base address of Task state segment descriptor + I/O map if exception interrupts are required from certain I/O operations. The required bit must be set in the I/O map by the XIOS, and the rest of the TSS and I/O map cleared.

If the XIOS does not require a trap on any I/O operations then this field can be set to 0000. The O/S will subsequently build a TSS descriptor with an I/O bit map length of 1024 and no bits defined.

TSS LIMIT - Total length in bytes of Task state segment descriptor + I/O permission bit map (inserted by XIOS INIT).

MP TABLE - Segment base address of memory free space table initialised by XIOS INIT and segment address inserted.

The XIOS must allocate a table with one word per 16k page pf total system memory. All O/S reserved pages and I/O and ROM space pages must have their memory areas protected by mapping them out with 0FFFFh in their repsective locations in the MP TABLE (see V386.A86 module in example PCXIOS).

NPAGES - Total memory size in 16k pages. Inserted by XIOS.

PTBL SEGMENT - Master page table segment base address (inserted by O/S).

V386 - CDOS 386 flag

Bit	Description
7	1= Allows the MEM module to attempt to move SYSDAT above video memory (PC Clones)
6 - 1	Reserved (set to 0)
0	0= Processor is in real mode. 1= Processor is in virtual 8086 mode or protected mode.

LIM - Lotus/Intel/Microsoft expanded memory emulator flag

0FF	Emulation required
00H	Emulation not required (can be changed by XIOS INIT routine).

LIM BASE - LIM spec page frame base address. (Default = 0D00h - can be modified by XIOS INIT.)

LIM MAXSIZE - Maximum memory pages allocated per LIM application. (Default = 1 megabyte (1024/16).)

EXCEP OFFSET - Offset address of exception handler in XIOS (inserted by XIOS INIT). If 0000 the O/S will pass any underfined protection errors to INT 6 handler in XIOS, via interrupt vector page address.

EXCEP SEGMENT - Segment address of exception handler in XIOS (inserted by XIOS INIT). If 0000 the O/S will pass any underfined protection errors to INT 6 handler in XIOS, via interrupt vector page address.

PAGE FAULT OFFSET - offset address of exception handler in XIOS for all Page Faults. (Inserted by XIOS INIT.)

PAGE FAULT SEGMENT - Segment address of exception handler in XIOS for all Page Faults.

HI-MEM ROOT - Pointer to linked of available high memory. XIOS to initialise this list if it requires memory mapping in unused memory. This memory is available for use in the IO-protect function. The structure of the linked list is the same as the memory descriptor structure (MD).

READ ONLY MEM ROOT - Pointer to a linked list to be initialized by the XIOS INIT. The list contains the blocks of memory to be write-protected. The structure of the items in the list is the same as the memory descriptor structure (MD).

INDEX