MultiDos Plus (tm) 4.01

"Multitasking for DOS"

Shareware Version

by

Nanosoft Inc.
13 Westfield Road
Natick, MA 017

Voice (508) 651-0091
FAX (508) 655-8860
BBS (508) 650-9552

Table of Contents

CHAPTER I

INTRODUCTION

MultiDos Plus is a multitasking extension to MS/PC-DOS which enables you to run multiple DOS programs simultaneously. In order to use MultiDos Plus you will need an IBM PC/XT/AT or true clone running under DOS 3.0 or later operating system. You will also need enough memory to contain the operating system, MultiDos Plus (about 64K), and your application programs.

MultiDos Plus enables you to load multiple programs in your system at the same time and have them all run concurrently using a preemptive multitasking scheduler. The scheduler provides for two different types of prioritized scheduling.

Programs running under MultiDos Plus have full access to DOS services by means of standard DOS function calls. Users may use the full range of DOS commands by running COMMAND.COM under MultiDos Plus. In addition MultiDos Plus is compatible with most "well behaved" off-the-shelf software available for the IBM-PC. A set of commands are available to load and execute programs, set program priorities, suspend/resume/abort programs and more. An Application Program Interface (API) enables user written programs to do the following:

- Change program priority
- Change time slice interval
- Run subroutines or functions as separate tasks
- Execute MultiDos Plus commands from within a task
- Communicate with other programs
- Suspend for a timed interval
- Synchronize resource usage using semaphores
- Wait on and signal events

MultiDos Plus supports the execution of programs in expanded memory using the LIM 4.0 Expanded Memory standard. In most cases the program need not be aware that it is running in expanded memory.

MultiDos Plus can be run as a DOS SHELL program or as a conventional EXE DOS program. MultiDos Plus and all programs running under it may be started up automatically by means of a startup file. With this feature the system can be started with no operator intervention providing a platform for turnkey systems.

Programs running under MultiDos Plus can access the display and keyboard by means of standard DOS or BIOS calls. Each program is assigned a distinct virtual display, with only the "foreground" program allowed to write to the real display. The user may bring any program to the foreground by means of a hot key.

Usually a program and task are synonymous under MultiDos Plus. However, it is possible for a task to create another task to run in parallel and share the same code and data space. These are referred to as internal tasks or threads and are described later in more detail.

SYSTEM OVERVIEW

MultiDos Plus is an EXE program and may be run from the DOS command line like any other EXE program. You can exit MultiDos Plus, and the system will revert back to its normal state.

On startup, MultiDos Plus takes control of the system by altering various interrupt vectors to point to itself and also by obtaining all the system memory. From this point on MultiDos Plus controls the loading and execution of application programs. MultiDos Plus can load programs in accordance with commands typed at the keyboard, by other programs already loaded and running under MultiDos Plus, or by commands in a special startup file.

MultiDos Plus provides true preemptive multitasking scheduling with a choice of two different priority schemes. Under the default scheduling scheme, CPU time slices are assigned to the various programs in a round robin manner. Programs with higher priority (lower priority number) receive more time slices than programs with lower priority. The time-slice interval is based on the timer interrupt in the PC and is approximately 55 msecs. You can change the time-slice interval by making the appropriate MultiDos Plus Application Program Interface (API) call or by a command line parameter when MultiDos Plus is started.

MultiDos Plus provides an alternative scheduling scheme under which the highest priority program in the system continues to run until it is suspended waiting for some system resource to become available. This scheduling scheme requires that programs be aware that they are running in a prioritized multitasking environment. It would be necessary for high priority programs to suspend voluntarily, in order to permit other lower priority programs to run. This scheduling scheme is necessary for many applications in order to guarantee that certain high priority tasks get scheduled in a predictable manner to handle real time events. Using the alternative scheduling scheme requires careful design of the overall system. For most normal situations the default scheduling scheme should be adequate.

Since DOS is not reentrant, MultiDos Plus controls all access to DOS system services on a first come first served basis. Tasks need not concern themselves with the details of accessing DOS in concert with other tasks and can make calls to DOS at any time.

Programs which cannot execute for any reason (waiting for keyboard input, waiting for DOS or MultiDos Plus services, operator suspended, etc.) are not assigned time slices.

MultiDos Plus also controls access to the visible display and the keyboard. All programs under MultiDos Plus including MultiDos Plus itself can be in one of two states -- foreground or background. Only the foreground program can display on the monitor or read from the keyboard. At any given time only one program can be in the foreground state. If any of the background programs writes to the screen using DOS or BIOS calls, the output is actually written to an "invisible" screen associated with that program. This screen becomes visible if the program is brought to

the foreground. You can swap programs between foreground and background by means of commands entered at the keyboard or by means of MultiDos Plus API calls. It is important to note that background programs do continue to run, even if you do not see them do anything.

MultiDos Plus software consists of the following components:

MULTIDOS.EXE - the multitasking kernel.
MDBIO10.EXE - a display driver TSR which is automatically

loaded by MultiDos Plus on startup. This driver redirects interrupt 10H display calls issued by background programs to the appropriate virtual screens.

MDHLP - an ASCII text file containing online help information.
MDDEBUG.EXE - a monitor/debugger program useful for debugging application programs running under MultiDos Plus.

GETTING STARTED

Install MultiDos Plus in your system by copying the distribution diskette to any directory in your system. We recommend that this directory be specified in your DOS PATH environment variable.

The distribution diskette includes a number of sample programs. This section describes how to bring up MultiDos Plus and run two of the sample programs.

Change your current disk and directory to the directory containing the MultiDos Plus files. From the DOS prompt, perform the following steps:

1) Type MULTIDOS
   MultiDos Plus is started, and you should see the MultiDos Plus prompt.
2) Type RM 128
   This command assigns 128K of memory to the next task.
3) Type RU DEMO1
   The DEMO1 program is loaded into a 128K memory block.
4) Type RM 64
   Assign 64K of memory to the next task.
5) Type RU DEMO2
   The DEMO2 program is loaded into a 64K memory block.

The ALTZ hot key (ALT and Z simultaneously pressed) may be used to bring any program to the foreground. The ALTZ hot key also switches you from an application program to the MultiDos Plus command interpreter.

Typing a ? gives you a list of valid MultiDos Plus commands.

CHAPTER II

COMMAND LINE OPTIONS

MultiDos Plus may be started by simply typing MULTIDOS when you have a DOS prompt on your screen, or it may be automatically started from the autoexec.bat file, or you may specify MultiDos Plus to be the startup DOS SHELL. If MultiDos Plus is started with no command line arguments, certain default parameters are assumed for the system. These parameters may be modified by means of optional command line arguments. Certain arguments require an additional parameter. For example:

```
MULTIDOS /NALTZ /NUMTCB 5
```

This will start MultiDos Plus with the hot key disabled and the number of programs in the system limited to 5.

The remainder of this section lists the command line options and a detailed description of each. NOTE: All options begin with a

slash (/). Command line options or arguments may be entered in upper or lower case.

/AUTO [startup file name]

The /AUTO option allows you to specify the path name of a file which contains MultiDos Plus commands. If this option is not specified, a file called AUTO.MTX is assumed to be the startup batch file. If the startup batch file is not present, the system comes up in the interactive mode permitting the user to enter MultiDos Plus commands manually. See the Automatic Startup section in this chapter for more information.

/CTX

The /CTX option allows programs running under MultiDos Plus to maintain unique current directory and current disk context. That is, a program doing a change directory will have no effect on the current directories of other programs. NOTE: The use of this command adds a significant overhead to each DOS file system call because of the amount of context information which must be saved and restored. The default is to have the current directory and current disk as global values which can be changed by any task. If your programs use absolute path names or if your programs do not change the current directory or disk, this option is not required. Using this option has performance as well as a memory overhead (400 bytes per task).

/EMMPGS [# of 16K EMM pages]

This option should only be used in a system with LIM 4.0 expanded memory. Use the /EMMPGS option to limit the size of the LIM 4.0 expanded memory page frame created by MultiDos Plus for expanded memory program loading. The value specified is the number of 16K pages to allocate to the frame. If this option is not specified, MultiDos Plus will create the biggest possible page frame for program loading.

/MAPALL

This option should only be used in a system with LIM 4.0 expanded memory. The /MAPALL option will cause MultiDos Plus to permanently map the LIM page frame (usually at segment C000H) and

make this part of the conventional memory in the system. NOTE: Do not use this option if you plan on running applications which use LIM driver calls and explicitly map LIM memory. By default, MultiDos Plus will map all other "holes" in the first megabyte as conventional memory permitting access to a substantially larger amount of conventional memory. This option has no impact on the size and function of the MultiDos Plus Page Frame where applications are loaded and executed.

/NALTZ

Use the /NALTZ option to disable the ALTZ hot key feature. This option is most useful in turnkey systems where it may not be desirable to allow the user access to the MultiDos Plus commands. A turnkey system would use the FG command in the AUTO.MTX or user-specified startup file to establish a foreground program at system startup, preventing the system operator from using any MultiDos Plus commands.

/NOECHO

The /NOECHO option will inhibit the echoing of the commands in the startup file.

/NOPROMPT

The /NOPROMPT option will inhibit the display of the MultiDos Plus command prompt when MultiDos Plus is in the foreground.

/NOREDIRECT

Use this option to prevent redirection of standard output and input to and from the communication ports for tasks started with the COMM command. This option is useful only for certain specialized communication programs.

/NUMTCB [# of TCBs to allocate]

MultiDos Plus allocates a structure called a TCB for each task. All the TCBs are initially allocated when MultiDos Plus first starts. The default number allocated is 12 which is enough for 10 tasks. MultiDos Plus itself requires two TCBs for its own internal use. The rest are available for user programs. Each TCB occupies 224 bytes of memory. Allocating unnecessary TCBs decreases the memory available for application programs. Also, use of the /CTX option increases the size of each TCB by about

400 bytes. Up to 64 TCBs may be allocated.

/PR [level]

MultiDos Plus provides two different prioritized scheduling schemes. The /PR command is used to override the default scheme of a simple round robin. The [level] is a value from 0 to 255 which is the priority level initially assigned to tasks when they are loaded. The scheduling scheme invoked by the /PR option allows the task with the highest priority to execute until a task of greater priority is scheduled or the running task suspends itself. Tasks of equal priority are still scheduled in a round robin manner within their own priority level. NOTE: Both scheduling schemes permit the use of 256 priority levels.

See the section on Inside MultiDos Plus in Chapter V for more details on the two scheduling schemes.

/TSL [interval designator]

The /TSL option will tell MultiDos Plus to preset the time-slice interval at startup. If this option is not specified, the default interval will be set to 55 milliseconds. The [interval designator] is a single character and must be one of 2, 4, 8, A or B. See the API function code 10 for more details on the effects of changing the time-slice interval. The following table shows the relationship between the /TSL parameters and the length of the time slice.

Parameter	Interval Length (milliseconds)
2	27.5
4	13.75
8	6.88
A	3.44
B	1.72

Any other values will result in an error message, and MultiDos Plus will terminate.

OPERATOR COMMANDS

A set of commands called Operator Commands control the loading

and execution of application programs under MultiDos Plus. These commands are used to load, execute, abort, suspend, and restrict the memory usage of the programs. A special startup file may also contain these commands to automatically load and execute programs when MultiDos Plus is started.

Commands may be entered in upper or lower case, and you must terminate each command with a carriage return.

It is possible for a running task to issue operator commands by using the Application Program Interface. See the section on Execution Control Functions in Chapter IV for details.The commands are grouped into four sections. The commands in the first section are those you may need to specify before loading and executing a program. All commands in this section are optional since they all have default values associated with them. The second section consists of a single command (RU) which is actually used to load the program. The third section deals with commands to control the execution of programs, and the fourth section contains miscellaneous commands.

Preload Commands

As noted earlier, all commands in this section are optional and are used to override defaults used in loading a program under MultiDos Plus. All parameters revert back to the default state as soon as a program is loaded. Some commands in this section are provided only to undo the effects of a previous command and are infrequently used.

AM - Additional Memory

When MultiDos Plus loads a program, the minimum amount of memory required to load the program is computed. If the AM command is used, the value specifed is added to the minimum amount of memory required to load the program. Enter the command using the following syntax:

```
AM <additional memory required in K>
```

The <additional memory required in K> parameter is a decimal value in units of 1,024 bytes.

CD - Change the Current Directory

The CD command is provided so that the user can change the current working directory for MultiDos Plus. Other DOS internal commands are available only if COMMAND.COM is loaded under MultiDos Plus.

CMP - Program Uses Conventional Memory Only

Use the CMP command to undo the effects of a previous XMP command. The CMP command indicates that the next loaded program will not make any calls to a LIM 4.0 expanded memory manager. CMP is the default unless an XMP command has been executed. NOTE: This command has no bearing on whether the program runs in expanded memory or not but only instructs MultiDos Plus that the program will not use expanded memory on its own.

COMM - Specify the CRT Port for the Program

Use the COMM command to specify a CRT communication port for the program's standard input and output. This will cause MultiDos Plus to redirect the program's standard keyboard and display data to and from the communications port. The command syntax is:

    COMM <port> <init-parms>

The parameter <port> specifies the communication port where the CRT is connected. A value of 1 designates COM1, value of 2 COM2, etc. The number of COM ports supported depends on the communications driver in your system. The <init-parms> is a hexadecimal value used to initialize the port when the task is loaded and is used to set the baud rate, parity, etc. See the Initialize Port function under the Software Interface for Terminal Communication section in Chapter V for more information. The <init-parms> value corresponds to the contents of register AL documented in that section.

It is IMPORTANT that the NANSI command be executed for the task after the task is loaded. The built-in ANSI.SYS functions in MultiDos Plus do not support programs running on CRTs connected to your PC. Therefore, it is necessary to disable this function for programs running on CRTs.

DI - Allocate Background Display Memory

When the next task is loaded by the RU command, the task will be given a display memory for background display output. This command is rarely needed since this is the default mode. This command is provided so that you can undo an ND command typed by mistake.

IRQ - Associate the LIM Map of a task with an External Interrupt

The IRQ command tells MultiDos Plus that when an external interrupt occurs the LIM map associated with the program should be mapped before the ISR associated with the interrupt is invoked. When the ISR returns, the original map is automatically restored. This command is required only for expanded memory programs performing asynchronous I/O with external devices.

The syntax for the IRQ command is:

    IRQ <level>

<level> is a number from 1 to 15 and indicates the hardware interrupt to be mapped. A value of 1 will map interrupt vector 9 which is normally the keyboard hardware interrupt. Values 1 thru 7 map to vector numbers <level> + 8, and values 8 thru 15 map to vector numbers <level> + 68H.

LC - Load in Conventional Memory

The LC command tells MultiDos Plus to load the next program into conventional memory. On systems with no expanded memory this is the default. If a LIM 4.0 board and driver are present in your system, programs are loaded in expanded memory by default. Use this command only if you have LIM 4.0 in your system and you wish to load your program in conventional memory.

LO - Program Execution Starts Only After Foreground Selection

By default, programs loaded under MultiDos Plus are not started until the program is brought to the foreground the very first time. The XQ command, described later, permits a program to start running as soon as it is loaded. The LO command is provided to undo a previous XQ command.

LX - Load Program in Expanded Memory

Use the LX command to reverse the effect of an LC command. The LX command tells MultiDos Plus to load the next program into expanded memory. This command is only meaningful if your system has expanded memory and a LIM 4.0 driver. If LIM 4.0 is present, the default is to load programs into LIM memory unless an LC command is entered.

ND - No Display Memory

If the next task to be loaded will not perform any output to the screen, the ND command can be entered to inhibit the allocation of a background display memory buffer for the task. This will save either 4K or 16K depending on the display adapter. If the ND command is used, the XQ command must also be used in order to start running the program. This is necessary because programs with no display memory cannot be brought to the foreground. Programs with no display memory will not show up when the ALTZ hot key is used.

RM - Reserve Memory

When MultiDos Plus loads a program, it automatically determines the amount of memory to assign to the program. This is done by examining the size of the program file and, in the case of EXE programs, the information in the EXE header. However, many programs request additional memory by making the DOS memory allocation calls. Programs written in high level languages may do so for stack and heap space even if you find no explicit memory calls in your program. It is important to note that this memory is allocated only from the initial memory reserved for the program. Therefore, many programs may not operate correctly unless sufficient memory is reserved for the program. The amount of memory needed by a program may be determined by reserving a large amount of memory and checking the memory usage of the program by using MDDEBUG.

The syntax for the RM command is:

    RM <size in K>

For example:

    RM 128

Use RM 128 to create a 128K memory partition for the next loaded program. The memory is not reserved until the program is loaded. If there is not enough memory to create the partition when the RM command is issued, no error message will be issued. The error message will be issued when the command to load the program is executed.

NOTE: RM and AM commands perform similar functions. RM lets you specify total size needed. AM specifies additional memory.

SBK - Suspend Task in the Background

The program will run only when it is in the foreground state. A typical use for this command is to prevent programs which write directly to the display memory from interfering with the foreground display. The foreground display is preserved between foreground to background and back to foreground switches.

XMP - Program Uses Expanded Memory

Use the XMP command to indicate that the program about to be loaded will use expanded memory. With this option, MultiDos Plus will save the LIM map context at the end of every time slice for the program and restore it when the program is ready to run. Use of this option has no bearing on whether the program is loaded in expanded or conventional memory. It only instructs MultiDos Plus that the program will make LIM EMM calls on its own.

XQ - Start Execution of Next Program Immediately After Loading

The XQ command will cause the next program loaded to start running immediately after it is loaded. If this command is not used, a program will start running only when it is brought to the foreground for the first time. If you use the ND command, you must use the XQ command before loading the program.

Load Program

RU - Load the Specified Program into Memory

The RU command is used to actually load a program into memory and ready it for execution. The program is loaded using the parameters established by previously entered commands like RM, XQ, ND and DI. After the program is loaded, the parameters are reset to their initial default values. A loaded program becomes a task under MultiDos Plus.

The following is the basic syntax of the RU command:

RU <program name> <prog cmd line args>

The <prog cmd line args> are the command line arguments that would normally be supplied to a program if it were executed directly from DOS.

MultiDos Plus does not require you to type RU. You may load a program simply by typing the name of the program as long as the program name does not conflict with any MultiDos Plus command.

eg. RU PROG ARG1 (or PROG ARG1)

If the name of a program conflicts with a MultiDos Plus command and the RU is not specified, the MultiDos Plus command definition takes precedence.

The program name may be any valid DOS path name of a COM or an EXE file. If a simple file name is specified, the DOS environment PATH variable is used to locate the program.

Once loaded, the program name is also used for the task name. The first seven (7) characters up to the period (.) of the simple file name are used for the task name. If a file with a path name of \ABC\DEFGHIJK.EXE were loaded, the task name would be DEFGHIJ. The directory specification of \ABC\ is dropped and so is the K. The .EXE or .COM extension is never included. If the RU is not specified, the name is also converted to upper case.

The standard input/output redirection and pipes normally supported by the DOS COMMAND.COM program are not available when running under MultiDos Plus.

Execution Control Commands

AB - Abort Task

Abort the named task. The task is actually aborted the next time it gets scheduled for execution. If the task is suspended for any reason (waiting for keyboard, waiting for a semaphore, etc.), it will be aborted only when it gets out of the suspended state.

The syntax for the AB command is:

    AB <task name>

When MultiDos Plus aborts the task, the memory reserved for the program is returned to the free memory pool.

CAUTION: If the task has "hooked" into any interrupt vectors, the consequences of leaving interrupt vectors pointing to unused memory may cause unexpected results.

FG - Move Background Task to Foreground

Use the FG command to bring the named task into the foreground. The syntax for the FG command is:

    FG <task name>

If the task has just been loaded and is waiting to begin execution, it will begin execution as soon as it has control of the foreground display.

Information on possible uses of the FG command is given in the sections on Foreground/Background Program Selection and Automatic Startup in this chapter.

GO - Resume Execution of a Task

A task which was previously suspended using the SS command can be restarted at the point where it was suspended. The syntax for the GO command is:

    GO <task name>

The task is removed from the Suspend queue and placed on the Ready queue.

PR - Set or Display Task Priority

Each task in the system is assigned an execution priority. The priority determines how often the task is given a slice of time by the task scheduler. The lower the priority number, the more frequently the task will be scheduled.

The PR command allows a task priority to be set to a value from 0 to 255. If no value is specified in the command, the current priority of the task is displayed. If you do not set any priority, loaded tasks will default to priority 0 or the level specified by the /PR command line option.

The following is the syntax of the PR command:

    PR <task name> <priority value>

SS - Suspend Task

The execution of the specified task is suspended until a GO command is issued to reactivate the task. The syntax for the SS command is:

SS <task name>

It is important to note that even though a task is marked for suspension, it may not be suspended immediately. Tasks waiting for keyboard input, semaphores, events and messages are not suspended until their current requests are satisfied. A task performing a DOS function call is not suspended until the DOS call is complete.

Miscellaneous Commands

ANSI - Activate ANSI.SYS for Task

The ANSI command activates the ANSI.SYS driver functionality built into MultiDos Plus for the specified task. ANSI.SYS can be enabled or disabled on a per task basis. The syntax for the ANSI command is:

ANSI <task name>

The <task name> is the name of the task to be activated. The default when a task is loaded is to have ANSI.SYS enabled. If a task should not have the ANSI.SYS functionality activated for it (for example, a task loaded with the COMM option), the NANSI

command can be used to disable this functionality.

ECHO - Echo Text

Any text following the ECHO keyword is echoed to the display. This command may be useful in AUTO.MTX files to display messages during a turnkey system startup. The following line:

```
ECHO This is a test
```

will display "This is a test" on the CRT display.

The command "ECHO *" will clear the display.

EXIT - Terminate MultiDos Plus

This command terminates MultiDos Plus and restores the system to its original state. All running tasks are immediately terminated. Caution should be used when using this command as some tasks may have taken certain interrupts during execution, and the EXIT command will not restore the interrupt vector to its original state.

The EXIT command has no effect if MultiDos Plus is running as a DOS SHELL.

NANSI - Disable ANSI.SYS

Use the NANSI command to disable ANSI.SYS for a task. When a task is loaded, the default is to have ANSI.SYS in effect for a task. The ANSI.SYS functions can be reactivated using the ANSI command. The syntax for the NANSI command is:

```
NANSI <task name>
```

<task name> is the task name of the task for which ANSI.SYS is to be disabled.

FOREGROUND/BACKGROUND PROGRAM SELECTION

As described earlier, there is always only one foreground program; and the rest of the programs are in the background. You can swap programs between foreground and background by doing ALTZ (pressing ALT key and Z key at the same time). If you do ALTZ when you have an application program in the foreground, your foreground program will go to the background; and MultiDos Plus will come to the foreground.

If you do ALTZ with MultiDos Plus in the foreground, you will see the name of one of your background programs. At this point you will be invited to either hit a return or do another ALTZ. Hitting the return will banish MultiDos Plus to the background and bring the named program to the foreground. Doing another ALTZ instead of hitting the return will list the next program (if any) in the background.

A program loaded without a background display memory may never be brought to the foreground. For this reason, programs without display memory will not have their name shown when the ALTZ task list is being displayed.

Another method of establishing the foreground program is to use the MultiDos Plus FG command. This command can be invoked directly from the MultiDos Plus command line or another program can issue the command using the API function to invoke MultiDos Plus commands from a program.

EXECUTING DOS COMMANDS

A set of DOS commands, known as DOS internal commands, are built into the DOS command processor, COMMAND.COM. This includes

commands to list a directory, copy files, delete files, etc. Also, the ability to execute batch files requires COMMAND.COM.

MultiDos Plus users who require these features should, therefore, first load and execute COMMAND.COM. The amount of memory to reserve for the COMMAND task depends on what you intend to do with the command processor. Reserving 64K is adequate for executing internal commands. Running batch files which invoke other programs may require you to reserve sufficient memory for their execution as well as the memory needed to execute COMMAND.COM.

AUTOMATIC STARTUP

Many applications require startup with little or no human intervention (a turnkey system). MultiDos Plus has all the features neccessary to be used in a turnkey system.

To start MultiDos Plus automatically, modify the AUTOEXEC.BAT file to invoke MULTIDOS.EXE. NOTE: The command to start MultiDos Plus should be the very last command in the batch file.

Now create a standard ASCII text file with the MultiDos Plus commands needed to load and begin execution of your application programs. Although the name of the file is not important, if you name it AUTO.MTX and place it in the current directory, MultiDos Plus will automatically look for it and execute the commands placed therein. If you choose a different file name or place the file in a different directory, use the /AUTO command line option to tell MultiDos Plus where the file is located.

Normally, MultiDos Plus will echo each command it reads from the AUTO.MTX file just before it executes the command. You may use the command line option /NOECHO to suppress the echoing of those commands. You may also want to use the /NOPROMPT option to inhibit the display of the MultiDos Plus command prompt.

In a turnkey system you may want to disable the ALTZ hot key facility. If your system operators do not have access to the MultiDos Plus command level, they will not be able to abort tasks, load other programs, or any other nasty things you may not want them to do. Use the /NALTZ command line option to disable the ALTZ facility. The /NALTZ option will disable the ALTZ before any other operation occurs.

Since you will probably want to disable the ALTZ facility, you will need an alternative method of starting your programs. Also, you will probably want one specific task to become the first foreground task. The XQ command will force programs to begin execution without being brought to the foreground. The FG command should be the last command in the file and will establish the desired foreground program.

Before each program is loaded, be sure to specify the XQ command so that the program will begin execution immediately after loading. This way the program need not be brought to the foreground to initiate its execution. This command would also be needed to start any programs which have no display.

The very last command in the AUTO.MTX file should be the FG command to start the foreground task. The foreground task need not be loaded with the XQ command option. The FG command will cause the task to begin execution.

RUNNING MULTIDOS PLUS AS A DOS SHELL

In many turnkey systems there may be no need for the DOS COMMAND.COM software. If this is the case in your system, MultiDos Plus can be executed as the DOS SHELL.

If you want to run MultiDos as a DOS SHELL, enter the following line in your system's CONFIG.SYS file:

```
SHELL=MULTIDOS <options>
```

This assumes that the MULTIDOS.EXE file is located in the root directory of the boot drive. The <options> are any of the MultiDos Plus command line options which may be needed for your configuration.

Because SHELL programs are not given an environment, MultiDos Plus will create an environment with only a single null string to pass on to loaded programs.

CHAPTER III

USING MDDEBUG

Included with the MultiDos Plus software is a program named MDDEBUG which may be used to observe the state of the system during execution. For example, the execution context of a program, its PSP, environment and other parameters can be examined. MDDEBUG will also allow you to display memory, system variable contents, check resource ownership, and list the tasks in the system. The following describes how to execute MDDEBUG and its various commands. Used in conjunction with an hardware emulation debug facility, MDDEBUG is invaluable for debuging a multitasking application. If you are using a system with a "dumb" terminal connected, you can use the COMM command to run MDDEBUG from the terminal if you have a communication driver installed as described in the Software Interface for Terminal Communication section in Chapter V.

Note that MDDEBUG makes reference to tasks and not programs. Although a program loaded under MultiDos Plus becomes a task, it is possible for a program to contain more than one task. MDDEBUG will show all the tasks runing under MultiDos Plus. See the API functions in Chapter IV for more information on creating "internal" tasks or "threads".

To run MDDEBUG, just type:

```
MDDEBUG
```

at the MultiDos Plus command prompt. If your system is configured with LIM 4.0 expanded memory, use the LC command to ensure that it is loaded into conventional memory. It must run in conventional memory in order to access data in programs loaded in LIM memory.

Use ALTZ to select and run MDDEBUG.

*** MultiDos Plus 4.01 Copyright (c) 1991 Nanosoft Inc. ***

You will be presented with a signon message and the MDDEBUG prompt which looks like:

```
MDDEBUG>
```

You may now enter one of the following single character commands. Enter a ? to get on-line help.

```
(C)heck Message Queue
(D)ump Memory
(E)nvironment of a Task
(F)ree Memory for Task
(H)andles and File Names for a Task
(I)ssue a Message to a Message Queue
(L)ist Task Names and Numbers
(M)ultiDos Command
(P)SP Information of a Task
(Q)uit
(R)esource Semaphore Owners and Requestors
(S)ystem Block
(T)ask Status
(X)panded Memory Information
```

When a command is entered, the operator is prompted for any additional parameters required to process the command. The command is executed, and the MDDEBUG prompt is again displayed.

The remainder of this chapter describes the commands in detail.

(C)heck Message Queue

The "C" command is used to check the contents of a message queue. When you press the "C" key, you will be asked to enter a queue number. The number must be a decimal value between 0 and 63. The status of the message queue is then displayed. Tasks waiting on the queue, if any, are listed. If any messages are on the queue, the contents of the messages are displayed. All messages are displayed in both hexadecimal and ASCII.

(D)ump Memory

The "D" command is used to display the contents of memory in the system. When you press the "D" key, MDDEBUG will ask for a memory address to display. If the address falls in the LIM expanded memory page frame, the number of the task whose memory is of interest is requested. Use the "L" command to obtain a list of tasks and their numbers. MDDEBUG will make sure the task is mapped in during the display process.

When the address of the memory to be displayed is requested, it must be entered in the form SEGMENT:OFFSET and the values entered must be hexadecimal.

Memory is displayed in 32-byte chunks. Each line displayed shows 16 bytes: first in hexadecimal format and then in printable ASCII. If a byte is not a printable ASCII character, it is displayed as a period (.). After the 32 bytes are displayed, a prompt is issued for another memory address. If a blank line is entered, the next 32 bytes are displayed. If an address is entered, 32 bytes are displayed starting with the new address.

(E)nvironment of a Task

The "E" command is used when you need to display the environment of a task. When you press the "E" key, the task number will be requested. Use the "L" command to obtain a list of tasks and their numbers. Enter the number of the task in which you are interested. The individual environment strings for the task will then be displayed. The path name of the task's program file will be displayed last.

(F)ree Memory for Task

The "F" command is used to display the memory allocation information of a task. When you press the "F" key, the task number will be requested. Use the "L" command to obtain a list of tasks and their numbers. Both the task's available and in-use memory is displayed. The starting paragraph of each memory block and the size of the block are shown. The starting paragraph is displayed in hexadecimal, and the block size is displayed in decimal. The size is the number of bytes (not paragraphs).

(H)andles and File Names for a Task

The "H" command is used to display the file handles in use by a task and the name of the file associated with the handle. When you press the "F" key, you will be asked to provide the number of the task. Use the "L" command to obtain a list of tasks and their numbers. The task name and the maximum number of files the task may have open is displayed. The file handle numbers are displayed next, one handle to a line. Following each handle number is the DOS file name associated with the handle. Only the name of the file is displayed. The directory and disk of the file are not provided.

(I)ssue a Message to a Message Queue

The "I" command is used to place a text message on a selected message queue. When you press the "I" key, you will be prompted for the queue number and then for the text of the message. The message will be placed on the specified queue. The message is limited to text characters entered from the keyboard.

(L)ist Task Names and Numbers

The "L" command is used to request a list of the names and numbers of all the tasks in the system. When you press the "L" key, all the tasks (and their associated numbers) in the system

are listed, one per line. Each task name is preceeded by a number which is its task number. The task number is used to reference a specific task in many of the other MDDEBUG commands. Following each task name is a number which is the priority of the task.

(M)ultiDos Command

The "M" command is used to send MultiDos Plus an Operator Command from MDDEBUG. When you press the "M" key, you will be asked to enter a command string. Up to 80 characters may be entered, terminated by a carriage return. The command string is sent to the MultiDos Plus command interpreter, and the string "Command sent" is displayed acknowledging that the command was sent. The command may be any valid MultiDos Plus command. Any error messages are displayed on the foreground display. See the API function code 15 (Execute a MultiDos Plus Command) in Chapter IV for more information on the effects of sending MultiDos Plus commands from a task.

(P)SP Information of a Task

The "P" command is used to display information from a task's Program Segment Prefix (PSP). When you press the "P" key, you are asked for a task number. Use the "L" command to obtain a list of tasks and their numbers. Information contained in the selected task's PSP is displayed. For more information on the contents of the displayed fields refer to your DOS Technical Referance manual.

(Q)uit

The "Q" command is used to terminate execution of MDDEBUG. When you press the "Q" key, execution is terminated immediately.

(R)esource Semaphore Owners and Requestors

The "R" command is used to display a snapshot of the MultiDos Plus Resource Semaphores. When you press the "R" key, the Resource Semaphore number, current owner task name and any

requesting task's names are displayed for all the Resource Semaphores having any activity.

(S)ystem Block

The "S" command is used to display the contents of the MultiDos Plus System Block data structure. When you press the "S" key, the contents of the System Block will be displayed. See the section on Useful System Data Structures in Chapter IV for a description of the fields in the System Block.

(T)ask Status

The "T" command is useful for obtaining information about a running task. When you press the "T" key, you will be asked for a task number. Use the "L" command to obtain a list of task numbers. Once you enter the task number, the information about the task will be displayed. The following describes the meaning of the displayed information:

The Task Name is the seven (7) character name of the selected task. Following the name is the priority of the task and the ANSI.SYS emulation state. If ANSI.SYS emulation is turned off for the task, the state will be 0. Otherwise it will be non-zero.

The next line displays the SS and SP registers for the task followed on the next line by the remainder of the task's registers.

The environment segment from the task's PSP is displayed in hexadecimal followed by the communications port number of the task. If the task is not communicating with an RS-232 terminal, the port number will be zero.

Twelve fields from the task's TCB are displayed, four to a line. They are CURIRQ, DISMEM, TOPMEM, TSKSEG, EMSPTR, EMMPRG, EMMHND, ALTMAP, PARTCB, TSKPID, DISTYP, and ADDR68. All values are displayed in hexadecimal.

Field	Description
CURIRQ	value set for task by the IRQ command
DISMEM	display memory segment
TOPMEM	tasks top of memory in K
TSKSEG	PSP segment for the task
EMSPTR	segment of LIM map
EMMPRG	task uses LIM flag
EMMHND	LIM handle for task
ALTMAP	LIM alternate map set for task
PARTCB	parent tasks TCB offset

TSKPID	current DOS process ID
DISTYP	0 = MDA, 1 = CGA
ADDR68	base i/o address of video controller

The next line displays the current display mode for the task, the number of display columns, and the length of the display memory.

The following line contains the current cursor positions for the four display pages (if MDA, only the first position is valid), and the next line displays the current active display page and the current cursor mode. The MDBIO10 driver only updates this information for background tasks. The foreground task information is kept in the low memory BIOS data area.

(X)panded Memory Information

The "X" command is used to display information about your system's expanded memory. When you press the "X" key, the raw page size, number of alternate register sets, the context save area size, and the number of available 16K EMS pages are displayed. Refer to your LIM 4.0 driver specification for a complete description of these values. If no expanded memory is configured in your system, the message "No expanded memory present" will be displayed.

CHAPTER IV

THE APPLICATION PROGRAM INTERFACE (API)

Programs can interact with each other or control their own execution by means of calls to the Application Program Interface (API). This is done through the software interrupt 15 (INT 15H). A program must load the function code of the function to be

performed into the AH register and any required parameters in the proper registers and execute an INT 15H instruction. If a function code not recognized by MultiDos Plus is used in register AH, control is passed to the interrupt handler present before MultiDos Plus was loaded.

NOTE: No registers are destroyed unless specifically noted.

This chapter has been divided into four sections: Execution Control Functions, Resource Control Functions, Inter-Task Messaging Functions, and Event Trigger Functions. The function descriptions in each section are organized in function code order. The following tables lists the function codes for each section.

Execution Control Functions

Function Code	Definition
0	give up time slice
3	suspend task for interval
7	spawn internal task
8	terminate internal task
9	change priority
10	change time slice interval
11	force display output to real video memory
12	restore old video memory
13	turn off time slicing
14	turn on time slicing
15	execute a MultiDos Plus command
17	turn off ALTZ
18	turn on ALTZ
19	return TCB address
20	return foreground/background flag
21	return pointer to System Block
22	initialize ADOS
23	map interrupt request
24	unmap interrupt request
25	unmap all interrupt requests

Resource Control Functions

Function Code	Definition
1	get resource semaphore

2	release resource semaphore
16	test a resource semaphore
26	map semaphore name to number
27	get resource semaphore by name
28	release resource semaphore by name
29	check if resource semaphore available by name

Inter-Task Messaging Functions

Function Code	Definition
4	send message
5	check queue for a message
6	get message

Event Trigger Functions

Function Code	Definition
30	event semaphore functions
30-0	clear event counter
30-1	signal event
30-2	wait for event

NOTE: Most compiler systems available for DOS have a facility for invoking software interrupts. See the technical reference manual of your particular compiler for details.

EXECUTION CONTROL FUNCTIONS

The Execution Control Functions are provided to give tasks a means of controlling various aspects of execution. Functions are provided for a task to control its own execution, to examine and alter system wide parameters, and to control the execution of other tasks. Several of the functions defined here must only be used by MultiDos Plus. They are only listed for completeness.

Give Up Time Slice

Function Code = 0

Entry:
AH = function code

The task making this call wishes to give up its time slice. The next ready task is given the remainder of the time slice.

Tasks which are performing certain types of timing loops should use this function to yield processor time to other tasks in the system.

If the /PR command line option was specified when MultiDos Plus was started, and the highest priority task in the system issues this function, and there are no other tasks of equal or greater priority, control will return immediately to this task.

Suspend Task for Interval

Function Code = 3

Entry:
AH = function code
DX = number of time ticks to suspend

The task making this call is suspended for the specified number of time ticks. A time tick is the basic time-slice interval. Be aware that this value can be changed by using function code 10 (Change Time Slice Interval).

This function should be used by tasks which want to delay execution for a specific period of time. The normal practice of executing a loop to maintain timing is not recommended in a multitasking environment. This technique will waste processor time which can be productively used by other tasks.

If the /PR option was invoked, this function can provide very accurate execution timing for high priority tasks. High priority tasks should use this function to yield processor time to lower priority tasks when they no longer need it.

Spawn Internal Task

Function Code = 7

Entry:
AH = function code
BX = CS register of new task
CX = IP register of new task
DX = stack size for new task in paragraphs

Exit:
AH = 0 if no error
1 if no free task control blocks
2 if no free memory for the new task's stack

A new task, called an internal task, is created and will start executing at the specified CS:IP. Control is returned immediately to the calling (parent) task which continues execution. The memory for the stack, specified in DX, is obtained from the parent task. The parent task must, therefore, have sufficient free memory to start an internal task.

See the section on Inside MultiDos Plus in Chapter V for more details on the operation of internal tasks.

Terminate Execution of an Internal Task
Function Code = 8

Entry:
AH = function code

An internal task terminates itself by invoking this function call. Its stack space (originally obtained from its parent task) is returned to the free memory pool of the parent task.

It is important that an internal task terminate itself using this function and not one of the DOS termination functions.

Change Priority
Function Code = 9

Entry:
AH = function code
AL = new priority

Change the task priority. Any value between 0 and 255 is allowed.

The meaning of the priority depends on whether or not the /PR command line option was specified when MultiDos Plus was started. See the /PR command line option in Chapter II for more

information. The Inside MultiDos Plus section in Chapter V also has more information on task priority.

Change Time Slice Interval
Function Code = 10

Entry:
```
     AH =       function code
     AL =       00H = 55 milliseconds (default)
                80H = 27.5 milliseconds
                40H = 13.75 milliseconds
                20H = 6.88 milliseconds
                10H = 3.44 milliseconds
                08H = 1.22 milliseconds
```

This function changes the length of the basic time-slice interval. Unless specifically stated, MultiDos Plus begins execution with a default time-slice interval of 55 milliseconds. This is the normal interval programmed into the timer for interrupt 8. MultiDos Plus uses interrupt 8 to preempt running tasks.

Although MultiDos Plus will change the interrupt interval for interrupt 8, a count of the number of times the interrupt has occured is maintained and after 55 milliseconds has elapsed, the interrupt 8 handler, which was present before MultiDos Plus, is executed. This preserves the BIOS time-of-day and any other timing required by your PC BIOS and TSR's.

When MultiDos Plus terminates execution, the timer is reset to its original 55 millisecond interval.

Force Display Output to Real Video Memory
Function Code = 11

Entry:
```
        AH = function code
```

The pointer to the task display memory is forced to point to the real hardware display memory. The old pointer is saved for the task and may be restored using function code 12 (Restore Old Video Display Memory).

This function is useful if a task wants to force a message to the CRT display regardless of its foreground/background state.

In order for this function to work properly, the invoking task's display mode must be the same as the foreground task's mode. Also, the text is displayed at the current cursor position

defined for the issuing task which may not be the same as the cursor position of the foreground task.

If a task issues this function and writes to the display when it is a background task, the displayed data will be saved to the foreground task's background virtual display if it is switched to the background.

Restore Old Video Display Memory

Function Code = 12

Entry:
AH = function code

The old video display memory pointer saved by function code 11 (Force Display Output to Real Video Memory) is restored.

NOTE: Function codes 11 and 12 are useful if a background task wants to display a message in the foreground display.

This function must not be called unless a function code 11 has been executed by the task.

The execution of function codes 11 and 12 does not affect the display of a foreground task.

Disable Multitasking

Function Code = 13

Entry:
AH = function code

When the timer interrupt signals the end of the time slice for this task, MultiDos Plus gives the next time slice to the same task. This effectively turns off the time slicing. The task will continue to receive all the CPU time until a function code 14 (Enable Multitasking) is executed.

This function, which disables multitasking, is useful for processing time-critical events. It may also be useful while executing regions of nonreentrant code shared by multiple tasks. A critical region of code would be bracketed by a function code 13 and a function code 14 which enables multitasking.

Hardware interrupts remain enabled even when multitasking is

turned off.

Enable Multitasking

Function Code = 14

Entry:
        AH = function code

This function reverses the effect of function code 13 (Disable Multitasking). If function code 13 is issued and is not followed by function code 14, no other task would receive time slices.

See function code 13 for more information on disabling and enabling multitasking.

Execute a MultiDos Plus Command

Function Code = 15

Entry:
        AH = function code
        DS:BX = pointer to null terminated command string

The string pointed to by DS:BX is executed as a MultiDos Plus command as if the command had been typed in at the MultiDos Plus prompt. The string must be terminated with an ASCII null character and must not include a carriage return or line feed at the end. Any lower case characters in the command are converted to upper case, up to the first space, before the command is executed. The converted characters are placed back into the command buffer. You can use this call to start other programs, suspend programs, etc.

When a task invokes this function, the task is suspended and placed in a First-In-First-Out (FIFO) queue. The queue is serviced by MultiDos Plus periodically. When MultiDos Plus finds a request to execute a command, the command is executed before the task is allowed to resume execution. If several tasks issue requests at the same time, the requests are serviced in a FIFO order.

Turn Off ALTZ Toggle

Function Code = 17

Entry:
        AH = function code

This function disables the ALTZ foreground/background selection key. Once this function has been issued, pressing the ALT and the Z key together will have no affect on MultiDos Plus. The only way to switch between foreground and background would be to issue an FG command to MultiDos Plus using INT 15H function code 15 (Execute a MultiDos Plus Command).

Turn On ALTZ Toggle (default)

Function Code = 18

Entry:
AH = function code

This function enables the ALTZ foreground/background selection key. This is the default mode when MultiDos Plus first begins execution unless the command line option /NALTZ was specified. If the ALTZ hot key was disabled, this function will reenable it.

Get Task Control Block Address
Function Code = 19

Entry:
AH = function code

Exit:
AX = offset of the task control block
BX = segment of the task control block

The segment and offset of the Task Control Block (TCB) of the calling task is returned. A TCB is a structure created by MultiDos Plus for each task in the system. See the section on Useful System Data Structures in Chapter V for more information on the contents of the TCB.

All TCBs reside in the same data segment which can be found in the first field of the System Block and is labeled as the segment of the System Control Block.

MultiDos Plus Foreground/Background Flag
Function Code = 20

Entry:
AH = function code

Exit:

```
        AX =    0 if MultiDos Plus is the background task
                1 if MultiDos Plus is the foreground task
```

Using this function, a task can check to see if the MultiDos Plus command task is in the foreground or background.

A task can check if it is the foreground task by testing the flag at offset 72 in its TCB. If the word at that location is binary zero, the task is in the background. The TCB of the current foreground task can be located using the pointer at offset 22 in the System Block.

Get System Block Pointer
Function Code = 21

```
Entry:
        AH = function code

Exit:
        AX = offset of the System Block
        BX = segment of the System Block
```

This function returns a pointer to the System Block structure. See the section on Useful System Data Structures in Chapter V for more details on the position of data in the System Block. This structure contains information which may be useful to an application.

MultiDos Plus Initialization
Function Code = 22

```
Entry:
        AH = function code
```

This function is used by MultiDos Plus during initialization and should not be invoked by any other task. It is documented here for the sake of completeness. Any attempt to use it by an application program will cause unpredictable, erratic results.

Map IRQ
Function Code = 23

```
Entry:
        AH = function code
        AL = interrupt request number to map (1 - 15)
        BX = offset of Task Control Block (TCB) to associate
             to interrupt
```

Exit:
AX = 0 if successful, not 0 if invalid value in AL

The LIM map of the task identified by the TCB is associated with the hardware Interrupt ReQuest level specified in the AL register. The value of the AL register is determined by subtracting 8 from the interrupt number. For example, a value of 1 corresponds to interrupt 9 which is normally the keyboard hardware interrupt. Mapping interrupt 8 (IRQ 0) is not allowed and returns a nonzero value in AX. Values in register AL which are greater than 7 are determined by subtracting 68H from the interrupt number. For example, a value of 8 corresponds to interrupt 70H which is IRQ 8 on the second PIC of an AT.

A task may only be associated with one IRQ at a time. If the IRQ is changed for a task, be sure to use function code 24 (Unmap IRQ) to disassociate an existing IRQ relationship.

This function is normally issued by the MultiDos Plus command task when a program is loaded with an IRQ command in effect.

Unmap IRQ
Function Code = 24

Entry:
AH = function code
AL = interrupt request number to unmap (1 - 15)

Exit:
AX = 0 if successful, not 0 if invalid value in AL

The Task Control Block (TCB) is disassociated from the specified interrupt request level. The association must have been previously established using function code 23 (Map IRQ).

Unpredictable results may occur if no relationship exists for the interrupt specified in the AL register when function code 24 is

invoked.

Unmap all IRQ'S
Function Code = 25

Entry:
AH = function code

Exit:
AX = destroyed

This function is used by MultiDos Plus to restore all the hardware interrupt vectors to the values present when MultiDos Plus first began execution.

WARNING: This function is for MultiDos Plus internal use only.

RESOURCE CONTROL FUNCTIONS

Tasks running under MultiDos Plus can synchronize their execution and also share common resources by using "semaphores". Semaphores are also referred to in this document as "resource semaphores".

Thirty-one semaphores numbered 1 through 31 are available for MultiDos Plus applications. These semaphores are created when MultiDos Plus starts, and programs may use them at any time.

WARNING: Semaphore 0 is present but is used internally by MultiDos Plus to control access to DOS services. Programs which get or attempt to release semaphore 0 may interfere with DOS access.

An additional set of 32 semaphores are also available but are referenced by name rather than a number. Names may be anything that programs choose to use but are limited to a maximum of 8 characters in length. Once used, names are remembered by MultiDos Plus and do not disappear until MultiDos Plus is terminated or the system is rebooted. Named semaphores are used internally by MultiDos Plus when applications post read/write calls to DOS drivers which support IOCTL calls. Consequently, the number of named semaphores available for application use may be less than 32.

Semaphore operations include the ability to get (function codes 1 and 27), release (function codes 2 and 28), or check the status (function codes 16 and 29) of a given semaphore. A program issuing a call to get a semaphore will return immediately if the semaphore is free or available. The semaphore is then considered to be owned by the program until the program releases it at a later point. A program attempting to get a semaphore will be suspended if the specified semaphore is owned by another program.

Semaphores may be used to handle a variety of synchronization problems but are generally used in the following two situations:

1) Two cooperating tasks may wish to synchronize their execution and attempt to get a semaphore in a prearranged sequence. The second task attempting to get the semaphore will be suspended. The first task may schedule the execution of the second by releasing the semaphore at the appropriate time.

2) The most common use of semaphores is to control the access to certain system resources. These resources may be hardware or software resources. An example of a hardware resource may be a device such as a communications port which may be used by any program but must be accessed in an exclusive manner. A software resource could be a function or a driver which is not reentrant. Programs accessing such resources should control their access by getting a semaphore prior to accessing the resource and releasing it, after the use of the resource. This technique will guarantee exclusive use of the resource when it is in use.

Get Ownership of a Resource Semaphore

Function Code = 1

Entry:
```
        AH = function code
        AL = resource semaphore number (0 - 63)
```

Exit:
```
        AH =    0 if no error
                2 if resource semaphore number is invalid
```

The ownership of the resource semaphore is tested and if not owned by another task, ownership is given to the calling task and control returns immediately. If the resource semaphore is owned by another task, the calling task is suspended until the owner gives up control. If several tasks request the same resource semaphore, the tasks are suspended and placed in a FIFO queue to wait for the resource semaphore to become available.

Release Ownership of a Resource Semaphore
Function Code = 2

Entry:
```
        AH = function code
        AL = resource semaphore number (0 - 63)
```

Exit:
```
        AH =    0 if no error
                1 if the task does not own the resource semaphore
                2 if the resource semaphore number is invalid
```

This function releases ownership of the specified resource semaphore. If a task is waiting for the resource, the waiting task will become the new owner and will be scheduled for execution. If several tasks are waiting for the resource semaphore, the first task that requested it will be given ownership. The other waiting tasks will remain on the queue and wait for their turn at ownership.

WARNING: Do not use this function in an interrupt service routine. The task which was interrupted may not be the owner of the resource semaphore.

Test Ownership of a Resource Semaphore
Function Code = 16

Entry:
AH = function code
AL = resource semaphore number (0 - 63)

Exit:
AH = 0 if resource is not reserved
1 if resource is reserved
2 if the resource semaphore does not exist
3 if the task already owns the resource semaphore

This function tests if the specified resource semaphore is already owned. You will probably want to use this function in conjunction with function codes 13 (Disable Multitasking) and 14 (Enable Multitasking) to disable the time slice scheduling while your task is checking the resource semaphore. If the resource is available, then the task can issue the function to get the resource before some other task can get the resource. This may be important if a task wants to get ownership of a resource without getting suspended.

Map Name to Number
Function Code = 26

Entry:
AH = function code
DS:SI = pointer to name string

Exit:
AL = 0H no error
4H out of string space
AH = resource semaphore number if no error

This function is used internally by function codes 27, 28 and 29 to equate a string with a resource semaphore number. The name string pointed to by DS:SI consists of 8 bytes all of which are significant, and it is not null terminated. If the name already exists, the number associated with that name is returned in register AH. If the name does not yet exist, it is entered into the name table; and a free number is assigned to the name. Up to 32 names may be created, and the numbers assigned to them will start with 32 and range up to 63. Once a name is created, it cannot be destroyed.

Function codes 27, 28, and 29 are the equivalent of invoking function code 26 followed by function code 1, 2, and 16 respectively.

Get Resource Semaphore by Name
Function Code = 27

Entry:
AH = function code
DS:SI = pointer to name string

Exit:

AH =	0H	resource semaphore is now reserved
	2H	illegal resource semaphore number
	3H	invoking task already owns semaphore
	4H	out of string space

The ownership of the resource semaphore is tested; and if unowned, ownership is given to the calling task and control returns immediately. If the resource is owned by another task, the calling task is suspended until the owner gives up control. If several tasks request the same resource semaphore, the tasks are suspended and are placed in a FIFO queue to wait for the resource to become available. See function code 1 (Get Ownership of a Resource Semaphore) for more details on getting resource semaphore ownership.

Release Resource Semaphore by Name
Function Code = 28

Entry:
AH = function code
DS:SI = pointer to name string

Exit:

AH =	0H	resource semaphore is available
	1H	resource semaphore is not available
	2H	illegal resource number
	3H	invoking task already owns semaphore
	4H	out of string space

This function releases ownership of the named resource semaphore. See function code 2 (Release Ownership of a Resource Semaphore) for more details about releasing ownership of a resource semaphore.

Test Resource Semaphore by Name
Function Code = 29

Entry:
AH = function code
DS:SI = pointer to name string

Exit:

AH =	0H	resource semaphore is available
	1H	resource semaphore is not available
	2H	illegal resource number
	3H	invoking task already owns semaphore
	4H	out of string space

The ownership of the resource semaphore named in the string pointed to by DS:SI is tested, and the results of the test are placed in the AH register on return. See function code 16 (Test Ownership of a Resource Semaphore) for more information on testing resource semaphores.

INTERTASK MESSAGING FUNCTIONS

The Intertask Message Facility is a convenient mechanism for programs running under MultiDos Plus to rapidly exchange data. API calls are available for tasks or programs to send (function code 4), receive (function code 6) and check (function code 5) for the presence of messages.

MultiDos Plus provides 64 message queues (or mail boxes) numbered 0 through 63. These queues are created when MultiDos Plus is started and are available at any time for programs to send or receive data. Unlike other multitasking environments, there is no need to create, use and subsequently destroy message queues.

When a task sends a message to a message queue, the contents of the message buffer are copied to a buffer in a message buffer pool. This buffer is queued on a data structure associated with the message queue. The task sending the message is free to continue its execution and also to reuse its message buffer.

The buffer pool, where copies of the messages are maintained by MultiDos Plus, is the unused conventional memory pool in the system. Therefore, if all the memory is assigned to various application programs (by means of RM commands and/or loading programs until memory runs out), there will be no memory available for the message buffer pool and intertask communication will not be possible.

The size of an individual message is limited to the smaller of 64K or the size of the largest free memory block.

Tasks which attempt to read a message from a queue are suspended if there is no message pending on the queue. If a message is pending, its contents are immediately copied to the receive buffer and execution continues. Multiple tasks may post reads on a single message queue. These tasks are queued up in a FIFO manner and receive the messages in the order in which they are received on the queue.

If any task is waiting to receive a message on the specified queue, the message buffer contents are copied to the receiving task's buffer area, after which the task that was waiting for the message is scheduled.

Tasks which read a message queue must specify the length of the receive buffer so that MultiDos Plus can ensure that the receive buffer is not overrun. MultiDos Plus will always transfer as many bytes as it can without overrunning the receive buffer.

As noted earlier, applications do not create queues; and therefore, there is no notion of "ownership" of the queues. Programs are free to write to and read from message queues as they please. It is, therefore, necessary for the system designer of the various applications to coordinate the use of the various message queues.

Cooperating programs may set up shared memory areas by sending the address and the size of the area to another program via a message queue. Since Multidos Plus runs under the real mode, there is no memory protection; and any program may access any part of the first megabyte. NOTE: This technique may not be possible for programs running in LIM memory.

Send a Message to Another Task
Function Code = 4

Entry:
        AH = function code
        AL = message queue number (0 - 63)
        DS:SI = message buffer pointer
        CX = length of message

Exit:
        AH =   0 if okay
                1 no system memory for message
                2 if illegal message queue number

Send a message to another task via a message queue. The contents of the buffer pointed to by DS:SI are copied to a system buffer allocated from the MultiDos Plus available memory pool, and the system buffer is added to the end of the specified message queue. As soon as this function returns to the invoking task, the task's message buffer may be used for another message without disturbing the previous message.

Check Message Queue for a Message
Function Code = 5

Entry:
        AH = function code

```
        AL = message queue number (0 - 63)

Exit:
        AH=     0 if no error
                2 illegal message queue number
        DX=     0 if no message, else the length of
                  the first message in the queue
```

Check if the specified message queue has an available message. Control is returned immediately. The function does not wait for a message to be placed on the message queue. In addition to checking for the presence of a message, this function is also useful for determining the size of a message before actually attempting to get the message. The task can then insure that the buffer it passes to get the message will be large enough to hold the entire message.

Get a Message from a Message Queue
Function Code = 6

```
Entry:
        AH = function code
        AL = message queue number (0 - 63)
        ES:DI = buffer pointer to put message in
        CX = number of bytes in buffer

Exit:
        AH = 0 no error
        CX = number of bytes transferred
        DX = actual message length
                or
        AH = 2 illegal message queue number
```

Read a message from the specified message semaphore. If no message is present, the task is suspended until one is available.

If a task does not know the maximum size of a message, it should issue a function code 5 (Check Message Queue for a Message) to check a message queue for the size of a message before actually

reading the message. If the buffer is not large enough to hold the message, the message is truncated before it is placed in the calling task's buffer. The truncated part of the message is lost.

EVENT TRIGGER FUNCTIONS

A useful feature provided by MultiDos Plus is a mechanism for having a task wait for a specific event to occur. While the task is waiting, it is suspended on a queue and will not use any processor time. When an event occurs, the process triggering the event (such as an interrupt handler) invokes a MultiDos Plus API call to signal that the event has occurred. If a task is waiting for the event, it is scheduled. If no task is waiting, a counter for the event is incremented.

There are 64 Event Triggers defined in MultiDos Plus. Each one consists of a 16-bit counter and an associated task waiting queue. When an Event Trigger function is invoked, a number from 0 to 63 is specified to indicate on which counter/queue to operate.

The Event Trigger functions are all available through the single interrupt 15H function code 30. Each individual function is specified by a subfunction code in register AL. Three subfunctions are defined to clear the counter (subfunction code 0), wait for an event (subfunction code 2), and to trigger the event (subfunction code 1). The following describes the interface to these subfunctions.

Clear Event Counter
Function Code = 30 subfunction = 0

Entry:
     AH = function code
     AL = subfunction code
     DX = event/trigger number (0 - 63)

Exit:
     AH = 0 no error

This function clears the counter for the specified event/counter number. This function can be invoked by either a task or an interrupt service routine.

NOTE: A counter will roll over after 65,536 events have occurred and none have been serviced.

Signal Event
Function Code = 30 subfunction = 1

Entry:
       AH = function code
       AL = subfunction code
       DX = event/trigger number (0 - 63)

Exit:
     AH =   0 no error

1 invalid event/trigger number

This subfunction is used to signal a waiting task that an event has occurred. This function may be invoked either by another task or by an interrupt service routine. If a task is waiting for the event, it is scheduled for execution.

If no task is waiting, then the counter is incremented by one to indicate that an event occurred. If more than 65,536 event signals are received, the counter will roll over to zero.

Wait for Event
Function Code = 30 subfunction = 2

Entry:
AH = function code
AL = subfunction code
DX = event/trigger number (0 - 63)

Exit:

AH =	0	no error
	1	invalid event/trigger number

A task issues this function to wait for an event to occur. If the value of the counter is zero, the invoking task is suspended until another task or an interrupt service routine invokes subfunction 1 (Signal Event). If the counter is not zero, control is returned immediately to the calling task; and the counter is decremented by one.

A potential use for an Event Trigger is a cooperative arrangement between a task (or several tasks) and an interrupt service routine. If an interrupt service routine has the job of collecting data and placing it in a buffer, it would notify a task waiting for the event when the buffer is full. The waiting task would then service the full buffer (when it is its turn to receive a time slice) and return to wait for the next event.

CHAPTER V

INSIDE MULTIDOS PLUS

The file MULTIDOS.EXE is the only file required to use MultiDos Plus. It contains the multitasking kernal which provides all the MultiDos Plus features except for those provided by the display driver (MDBIO10.EXE). If your application will only have one task which will ever write to the CRT, there is no need to install MDBIO10 on startup. Therefore, the only files you may ever need to run MultiDos Plus are MULTIDOS.EXE and perhaps MDBIO10.EXE. This chapter will describe some of the inner workings of the MULTIDOS.EXE program which may help you to better understand how to use MultiDos Plus for your application.

MultiDos Plus Startup

When the MULTIDOS.EXE program begins execution, it goes through a complex initialization process. During initialization, a number of data structures are created and initialized followed by the "hooking" of a number of the interrupt vectors. All the command line parameter options are examined; and the appropriate internal flags, etc. are set for later use.

During intialization, the memory for the Task Control Blocks (TCB) is allocated. The value specified with the /NUMTCB option is used as the number of TCBs to create. If no /NUMTCB option is specified, a default value of 12 is used. When specifying the number of TCBs, keep in mind that two additional TCBs are required by MultiDos Plus for internal use. Also, when a task executes a DOS child process function, each active child process requires an additional TCB.

The size of the TCB is determined by the /CTX option. If /CTX is not specified, the TCB will only be about 224 bytes. If /CTX is specified, the TCB will be about 624 bytes. The actual size may vary slightly depending on the version of MultiDos Plus you are using.

Once the TCBs have been allocated, the first two are used for two internal tasks, the MULTIDO and the IDLE tasks. If you use the "L" command in MDDEBUG, the first TCB is MULTIDO and the second is IDLE. These two tasks are created during initialization.

The interrupt vectors "hooked" into by MultiDos Plus are 8, 9, 12H, 15H, 16H, 20H, 21H, 22H, 23H, 24H, 25H, 26H, 27H. If MDBIO10 is loaded, interrupt 10H is also "hooked". If your application needs to make use of any of these interrupts, you must take care to pass control to the MultiDos Plus interrupt handler if the interrupt is not applicable to your application. For the most part MultiDos Plus tries to preserve the DOS or BIOS meaning of the interrupt with multitasking in mind.

Once the INT 8 interrupt vector is "hooked", the multitasking is initiated. The initialization routine becomes the MULTIDO task which is described below.

The Command Task

The Command Task is the main MultiDos Plus task and performs several different functions. The task name in the TCB is MULTIDO which can be examined with various MDDEBUG commands. As mentioned above, the Command Task is a continuation of the initialization process and comes into being as soon as the multitasking is turned on by "hooking" interrupt 8.

The first function performed is the loading of the MDBIO10 driver, if required. Once MDBIO10 is loaded, a delay of 18 ticks is executed to give MDBIO10 plenty of time to perform its initialization. To make the new interrupt vector a part of the Command Task display context, the content of the System Block's "pointer to new interrupt 10H" is placed into the actual interrupt 10H vector.

Next is the processing of the AUTO.MTX startup file, if one is present. During the execution of the AUTO.MTX, only command processing takes place. If a started task attempts to use any of the other services provided by the Command Task, they will not be performed until the AUTO.MTX processing is finished.

The Command Task will now enter its main function loop which continually checks several queues for requests and then reads a character from the keyboard. If there is no character to read, the Command Task suspends itself for at least one timer tick. When the MULTIDO task is in the background, it suspends itself for 4 ticks before scanning the queues and looking for an ALTZ key.

Besides interpreting MultiDos Plus commands, the Command Task also provides services for inter-task communication, program loading, program termination, foreground/background switching and task command execution. All these functions are initiated when the Command Task finds a request for these services on one of the request queues. These queues are checked each time the Command Task wakes up.

The "IDLE" Task

The "IDLE" Task is a very simple task which exists solely for the purpose of giving the system something to do when there is nothing else that needs to be done. The "IDLE" Task is always in a loop which continually gives up its time slice by issuing an API function code 0 (Give Up Time Slice) call.

The General Application Environment

When a COM or EXE program begins execution as a loaded program under MultiDos Plus, it can usually execute as if MultiDos Plus was not in the system and the program was started by DOS.

Each program has a PSP block built for it just as if the program had been loaded by DOS. This gives each task its own environment string (which is a copy of the environment present when the MULTIDOS.EXE first began execution) and separate file handle table. The PSP will also have the command line string which is

taken from the command line that loaded the program. As mentioned in earlier chapters, no standard input/output redirection is provided by MultiDos Plus. The first five file handles of a task are inherited from the MultiDos Plus PSP. The DOS CLOSE HANDLE function will not close a handle less than 5. If attempted, the function returns a normal completion code with no error to the calling task.

DOS Services

Tasks are normally free to make use of DOS interrupt 21H services as if they are the only program in the system. MultiDos Plus uses Resource Semaphore 0 to control access to DOS without the tasks being aware of the process. In addition, a number of DOS services are provided directly by MultiDos Plus so as not to tie up DOS. These are normally functions which read from the keyboard and write to the CRT display.

When a task issues an INT 21H for a DOS function, several functions are performed just before entry to the old DOS interrupt vector. MultiDos Plus sets the DTA and the Process ID for the task. The Process ID is the task's PSP segment and is set using DOS function code 50H. Also, if the /CTX option was specified when MultiDos Plus was invoked, the directory context of the task is restored. When the task returns from DOS, the directory context will be saved in the task's TCB.

Task Scheduling

MultiDos Plus provides a preemptive multitasking system which offers a choice of two different scheduling schemes. The interrupt 8 timer is used to preempt executing tasks. Tasks waiting for execution are held in a Ready Queue and receive slices of processor time. When the interrupt 8 timer expires, the currently executing task is suspended and placed at the end of the Ready Queue. The next task on the queue is then allowed to execute until the timer expires again.

The default scheme of selecting the next task to execute from the Ready Queue removes the task at the head of the queue, checks its scheduling counter and if zero, the task is executed. If the scheduling counter is not zero, it is decremented by one and placed at the end of the Ready Queue. If the scheduling counter is decremented to zero, the counter is reset to the task's priority level; and the task is allowed to execute. Using this type of scheduling scheme, all tasks will eventually receive a time slice. When a task is loaded, its priority level is set to zero.

A second type of scheduling scheme may be selected by using the /PR command line option when MultiDos Plus is started. The /PR

option changes the way MultiDos Plus selects tasks from the Ready Queue for execution. A task is selected by scanning all the tasks on the Ready Queue. The task with the highest priority level (lowest value) is selected for execution. If several tasks have the same priority level and they have the highest priority, the task closest to the head of the queue is selected. Thus, tasks

with equal priority will execute in a round robin manner. The use of this scheme can provide high priority tasks with highly reliable scheduling for critical real time events. NOTE: High priority tasks should make use of API function code 3 (Suspend Task for Interval) to avoid monopolizing all of the system's processing time. The API Event Triggers are also good for controlling high priority tasks.

The scheme for selecting the next task for execution also applies when an executing task gives up its processing time by giving up its time slice, suspending itself, requesting a resource semaphore, getting a message from a message queue, or waiting for an event.

Task Context Switch

There are a number of instances where MultiDos Plus will save a running task's context, place the task's TCB on a queue, find the next task ready to run, switch in the new task's context and resume execution with the new task. A context switch occurs when a task executes an interrupt 15H API function which causes the task to be blocked from execution or when the interrupt 8 timer interrupt occurs. It is the interrupt 8 timer interrupt which preempts the running task and the context switch is performed before the old timer interrupt handler code is executed.

A task's context information consists of registers AX, BX, CX, DX, BP, DI, SI, DS, ES, CS, IP, FLAGS, SS, and SP. All the register context is saved on the task's stack with the exception of the SS and SP registers which are saved in the task's TCB. The interrupt 10H interrupt vector is also saved in the TCB; and if the task uses LIM memory, the LIM context is also saved in a block of memory attached to the task's TCB.

Internal Tasks (Threads)

MultiDos Plus supports the concept of internal tasks or threads within the same program. A program may start subroutines or functions as separate processes called internal tasks.

The concept of internal tasks offer several obvious advantages. The various tasks can easily share data without resorting to intertask communication or other means. The other major advantage is that reentrant code can be shared between the tasks, resulting in significantly smaller memory requirements.

Greater care is needed in implementing a system based on internal tasks. The major consideration is to examine the code that will be shared by the various tasks and determine whether it is reentrant. Though this is fairly easy to determine in the case of assembler code, determining the reentrancy of higher level language code may be difficult. Even if the code written by the MultiDos Plus user is reentrant, it should be noted that the various tasks may reference library run-time functions which may or may not be reentrant.

Internal tasks share the same Program Segment Prefix with the parent, with the result that files opened by one task are

available to the parent as well as other children. The child internal tasks inherit other properties of the parent including the memory control blocks. Memory allocation done by a child internal task results in memory being allocated from the parent.

MultiDos Plus automatically assigns names for internal tasks. Unique names are assigned to internal tasks by incrementing the last character of the parent task's name. Internal tasks are not shown when ALTZ is done and cannot be brought to the foreground.

It is very important that an internal task terminate its execution by using the API function code 8 (Terminate Internal Task). Using a DOS termination function will result in unpredictable results.

RUNNING WITH A LIM 4.0 EMM DRIVER

MultiDos Plus fully supports any LIM EMS memory you may have in your system. A LIM driver version 4.0 or greater must be present in your system in order for MultiDos Plus to take advantage of the LIM memory.

On 386 PCs no hardware LIM boards need to be present as long as you have a suitable software emulator which conforms to LIM 4.0. For 286 as well as 8088 based systems you must have a LIM board with the appropriate LIM 4.0 driver. NOTE: Certain 286 PCs have LIM 4.0 emulators which enable you to use extended memory as LIM 4.0 memory. However, this technique will not be suitable for use under MultiDos Plus as the map switching will not be fast enough.

Certain older LIM boards such as the Intel Above Board may be controlled by a LIM 4.0 driver but provide mappable memory of only 64K. These boards will severely restrict the size of the programs you can run in LIM memory.

Newer LIM boards which provide hardware support for the LIM 4.0 specification such as the Intel Above Board PLUS or the AST Rampage Plus are highly desirable for use under MultiDos Plus. However, even with these boards, your hardware configuration may limit the amount of mappable EMS memory. The ideal configuration for MultiDos Plus is a 386 PC with a LIM 4.0 memory manager or a PC/AT with conventional memory limited to 256K with the rest supplied by a LIM 4.0 board with the ability to backfill conventional memory to 640K. For faster context switches, boards supporting alternate map sets are desirable.

Nanosoft Inc. has tested various configurations but does not explicitly endorse any single product due to the large number of boards and emulators available in the market. It is up to the

user to determine the best configuration to suit your needs.

MultiDos Plus permits your normal DOS programs to run in expanded memory with no changes to your programs. This permits large multitasking systems to be developed without being limited by memory. The code size of a single program is still limited by the size of the mappable page frame in your system (unless overlays are used). Programs running in expanded memory may use LIM memory by means of standard LIM driver calls to map memory in the LIM page frame. MultiDos Plus saves and restores map contexts for all programs with no special coding required on the part of application programs.

Certain terms need to be fully understood in order to make effective use of LIM memory under MultiDos Plus. The MultiDos Plus Page Frame is defined as the area of mappable LIM memory used to load and execute application programs. It is the largest contiguous area of mappable LIM memory available in the system and generally starts at the first 16K boundary after MultiDos Plus and extends up to your display memory. The LIM page frame is a 64K area of memory usually starting at C000 or beyond and is used by applications to map LIM memory by normal LIM driver calls. A normal DOS program which uses LIM memory will run unchanged under MultiDos Plus whether it is loaded in

conventional or LIM memory (in the MultiDos Page Frame).

In addition to the two page frames discussed, most systems have "holes" in the first megabyte of memory where LIM memory can be mapped. On startup, MultiDos Plus permanently maps LIM memory into these "holes" and links them to the conventional memory in your system. This may result in significant increase in the amount of conventional memory in your system. By default, the "hole" corresponding to the LIM page frame is not mapped on the assumption that you may be running applications which explicitly map LIM memory and need the use of the LIM page frame. If this is not the case the startup option /MAPALL will map all available holes including the LIM page frame. NOTE: This permanent mapping does not preclude applications from running in LIM memory in the MultiDos Plus Page Frame.

As noted earlier, programs may run in expanded memory without any special consideration. There are, however, some exceptions. Programs running in expanded memory performing data transfers to/from external devices triggered by hardware interrupts will not function correctly. This is due to the fact that the data transfer may take place when the map set associated with the program is not active. To handle this situation, the command IRQ lets you associate an IRQ number with a specific program's map set. Simply specify this parameter when you load the program, and MultiDos Plus will automatically map the program when the specified interrupt occurs. After the ISR is complete MultiDos Plus will remap the map context in effect when the interrupt

occurred. Currently there is no provision to associate multiple IRQs with a single program.

Programs running in LIM memory may not set up shared memory areas with other programs running in LIM. They may do so with programs running in conventional memory provided that the programs explicitly handle the mapping when necessary.

MultiDos Plus uses alternate map sets (if available) to perform rapid context switching of LIM maps. If alternate map sets are used up (or if none are available to start with), MultiDos Plus saves and restores map contexts in internal save areas associated with each program.

DRIVERS UNDER MULTIDOS PLUS

The discussion in this section applies to installable DOS device drivers as well as Interrupt Service Routines and TSRs.

In general, drivers, TSRs and ISRs written for DOS will work under MultiDos Plus with no changes. There are, however, some restrictions when expanded memory is used in conjunction with MultiDos Plus. This section offers general guidelines for implementing and using existing drivers under MultiDos Plus. Having a clear understanding of the MultiDos Plus architecture will permit the user to circumvent most of the limitations.

In general, all TSRs and ISRs must be loaded prior to MultiDos Plus. DOS installable drivers are of course always loaded when the system is booted. Though MultiDos Plus permits TSRs to be

loaded as tasks under MultiDos Plus, in most cases this approach is not recommended. TSRs which change interrupt vectors used by MultiDos Plus MUST be loaded before MultiDos Plus.

Any software which gets invoked as a result of external interrupts may not normally reside in mappable expanded memory. The external interrupt would cause the processor to transfer control to code which may not be currently mapped. The IRQ command described in Chapter II may be used to handle this situation but will result in significant performance overhead, especially if the interrupts occur at a rapid rate.

TSRs and ISRs may be loaded in expanded memory which is permanently mapped (the "holes" in high memory which are permanently mapped by MultiDos Plus). Any driver, TSR or ISR which uses expanded memory (by making explicit LIM driver calls) must save and restore the current task's map context.

Standard DOS drivers are usually synchronous drivers (control is transferred to the application only after the driver completes the request). Some drivers, especially network related drivers, are implemented as asynchronous drivers. Such drivers may need to

be redesigned if they need to service application programs running in expanded memory. When a request is initiated by a program, the driver must determine the TCB of the program, and later map the LIM map associated with the program to perform data transfers to/from driver address space to program address space. If drivers are implemented to service multiple application programs, the driver must be reentrant.

Drivers and ISRs which are triggered by external interrupts should not use most of the MultiDos Plus API calls. The exceptions are calls to determine the current TCB and certain event trigger calls. It should be noted that ISRs may get invoked at any arbitrary time and runs under the context of the user task which was interrupted. Using MultiDos Plus system service calls may, therefore, affect the execution of the task which was interrupted with unpredictable results.

Event trigger calls have been designed for use by ISRs and drivers and should be the primary vehicle for communication from a driver to tasks running under MultiDos Plus.

THE DISPLAY AND INTERRUPT 10H

Well behaved DOS programs either use DOS interrupt 21H functions or BIOS interrupt 10H functions to place data on the visual CRT display. Even the DOS interrupt 21H functions wind up invoking the BIOS interrupt 10H functions for display functions. Based on this, MultiDos Plus includes a software driver which intercepts BIOS interrupt 10H. The driver will only allow the foreground program to write data to the real CRT memory, while background programs have their display data written to a different piece of memory. This driver is the file MDBIO10.EXE which is included on the distribution diskette.

MDBIO10 supports the standard BIOS INT 10H functions for background tasks for the monochrome and the color graphics

displays. Other display adapter modes are not supported for background tasks. Foreground tasks can use any display mode supported by your adapter hardware. If the foreground task is using an unsupported mode and the task is moved to the background, some of the contents of the display are lost.

MDBIO10 is a Terminate Stay Resident program which performs some initial setup and then invokes DOS INT 21H function code 31H to terminate. The initialization sets the interrupt 10H vector to point to the MDBIO10 interrupt handler and sets the new interrupt 10H pointer in the System Block before terminating. Subsequent execution is initiated by MultiDos Plus tasks invoking interrupt 10H.

When the driver is entered as a result of an interrupt 10H call, MDBIO10 first determines if the calling program is in the

foreground or in the background. Display calls issued by the foreground program are passed through to the original BIOS display handler.

Display output of a background task is written to its virtual screen. The virtual screen is on a paragraph boundary, and its segment address is in offset 18 of the TCB.

Data associated with the display for each task is maintained in its TCB. This area is identical to that maintained by BIOS in low memory.

USING THE MATH COPROCESSOR

In the interest of performance, MultiDos Plus does not save and restore the context of the 8087/80287/80387 math coprocessors when a task's context is switched at the end of a time slice. If more than one task requires the use of the coprocessor, one of the resource semaphores should be used to control access to it (See the README file for more information about 8087 context).

The first step is for all tasks to agree on which resource semaphore number to use. Whenever a task needs to use the coprocessor, it would issue INT 15 function code 1 (Get Ownership of a Resource Semaphore) to get control. When function code 1 returns with no error, the task may begin using the coprocessor. When the task is done, it must issue INT 15 function code 2 (Release Ownership of a Resource Semaphore) to release control of the coprocessor. The coprocessor would now be available for use by other tasks.

Even if your system does not have a math co-processor, the run-time package supplied with your compiler may load a driver to emulate the 8087. You should still use a resource number to control access to the driver since the driver will not know which task is using the driver at any given moment. This will also prevent non-reentrant drivers from being used by more than one task at the same time.

USEFUL SYSTEM DATA STRUCTURES

The Task Control Block

A Task Control Block (TCB) is a data structure which contains task related information needed by MultiDos Plus to keep track of the task. Each task running under MultiDos Plus has a TCB. A task can obtain a pointer to its own TCB by issuing an interrupt 15H with a function code of 19 (Get Task Control Block Address). All TCBs are allocated in the same segment, and they all start on

a paragraph boundry. The first TCB starts at byte offset 16. The TCB size in the system block can be used to calculate the start of any TCB in the system. The following table contains the definitions of the fields contained in the TCB. All offsets and sizes are in bytes.

Offset	Size	Definition
0	2	link to next TCB
4	8	null terminated task name (7 characters max)
14	2	task starting segment (PSP)
16	2	abort/suspend flags
18	2	current display memory segment
20	2	priority level (0 - 65,534)
22	2	time slice counter
26	2	suspend timer value
28	2	stack segment
30	2	stack pointer
32	2	display type
34	2	display memory
38	2	termination count
40	2	equipment flag for BIO10 driver
42	1	background CRT mode

Offset	Size	Definition
43	2	number of CRT columns
45	2	length of CRT display memory

47	2	segment of real CRT memory
49	16	8 CRT cursor positions
65	2	current cursor mode
67	1	active CRT page
68	2	6845 address
72	2	foreground task flag
80	2	save video memory segment
82	4	INT 22H context
86	4	INT 23H context
90	4	INT 24H context
94	2	top of memory for this task
100	2	DTA segment
102	2	DTA offset
108	1	current attribute (ANSI.SYS)
109	1	current horizontal coordinate (ANSI.SYS)
110	1	current vertical coordinate (ANSI.SYS)
111	1	current display state (ANSI.SYS)
112	1	maximum columns (ANSI.SYS)
113	1	current page (ANSI.SYS)
114	2	saved cursor position (ANSI.SYS)
116	1	parameter buffer index (ANSI.SYS)
117	1	current screen mode (ANSI.SYS)
118	1	wrap flag (ANSI.SYS)
119	6	parameter buffer (ANSI.SYS)
125	1	keyboard DSR state (ANSI.SYS)
126	7	keyboard DSR buffer (ANSI.SYS)
136	16	request header for DOS driver calls
166	2	segment of LIM map if LIM task
168	2	task makes LIM calls flag
170	2	LIM handle for this task
172	2	keyboard shift status
186	2	parent TCB if child process
188	2	termination code
190	2	COM port number
196	2	current IRQ number
200	2	miscellaneous flag word
204	4	INT 10H vector context
208	2	LIM alternate map set number
210	414	DOS current disk and directory context

The System Block

The System Block is a data structure inside MultiDos Plus which contains pointers to other MultiDos Plus internal data structures. It is used by other programs to retrieve information for debugging and setting up a display driver for BIOS interrupt 10H.

The pointer to the System Block is obtained by issuing the MultiDos Plus service interrupt with a function code of 21 (Get System Block Pointer). The pointer returned points to a structure of the following format:

Offset	Size	Definition
0	2	segment of System Control Block
2	2	redirection flag set by /NOREDIRECT
4	2	no BIOS 10H flag set by /NO10
6	4	pointer to old interrupt 10H
10	4	pointer to new interrupt 10H
14	4	pointer to word with current TCB offset
18	4	pointer to word with Idle task TCB offset
22	4	pointer to word with foreground TCB offset
26	4	pointer to word with MultiDos Plus TCB offset
30	2	TCB size
32	2	number of TCBs
34	2	expanded memory present flag
36	2	base segment of expanded memory frame
38	2	size of expanded memory frame in 16K pages
40	2	base segment for conventional memory tasks
42	2	number of conventional memory paragraphs
44	4	pointer to list of queue pointers

SOFTWARE INTERFACE FOR TERMINAL COMMUNICATION

When a program is loaded with the COMM command option, MultiDos Plus will direct certain keyboard and display functions to/from the serial communication ports. This section describes the specific functions performed and the software interface to the communication ports. By using this facility it is possible to run many standard DOS programs from a "dumb" terminal, including COMMAND.COM.

In order to use this feature you must install a special driver which will handle the low level BIOS communication functions. This driver is currently not supplied with MultiDos Plus; a detailed description of the driver is provided here so that the user can develop a suitable driver.

Terminal redirection is enabled by the COMM command described in Chapter II. NOTE: The communications driver must be resident prior to using the command. It may be loaded before MultiDos Plus, or you can have MultiDos Plus load and execute the driver. The driver must "hook" onto software interrupt 14H to handle functions issued to it by MultiDos Plus. A basic requirement of the driver is that it use the hardware interrupts to collect input data from the communications port. If the driver is not interrupt driven, it will most likely function improperly because the program was not assigned a time slice when the port needed servicing.

MultiDos Plus interfaces to the driver by means of software interrupt 14H. The function codes used by the driver are distinct from the BIOS RS-232 services.

Basically, MultiDos Plus invokes INT 14H in three places. When a task is loaded, INT 14H function codes 20H, 24H and 25H are issued to initialize the specified communications port. When a task reads from the keyboard, INT 14H function codes 22H and 23H are issued and if there is a buffer overflow, function code 25H is issued to clear the error. Function code 27H is issued if INT 16H is issued to check for a key press. When a task issues an INT 10H function code 0EH to display a character, the MDBIO10 driver will invoke INT 14H function code 21H to send the character to the task's communication port.

When the task is loaded, the communications port number is placed in the task's TCB at offset 190. Later read and write functions check the TCB field to determine which communication port to use for the operation. The port number placed in the TCB is zero if no communication redirection should be done for the task. If the port is a one (1), then COM1 is assumed. The highest port number allowed depends on what is supported by the port driver.

When the task is loaded and the port is initialized, the INT 14H functions are issued in the order 25H, 20H, and then 24H to initialize the port.

Initialize Port

Function Code = 20H

Entry:
AH = function code (20H)
AL = initialization parameters
DX = port number 0 based (COM1 = 0)
Exit:
AH = 0H command successful
41H no port address
64H monitor mode active

The RS-232 port specified in the DX register is initialized for the parameters specified in the AL register. The bit settings in AL have the following meaning:

Bits	Definition
5 - 7	baud rate
000	110
001	150
010	300
011	600
100	1200
101	2400
110	4800
111	9600
3 - 4	parity
00	none
01	odd
11	even
2	stop bits
0	one stop bit
1	two stop bits

```
     0 - 1      word length

      10        7 bits
      11        8 bits
```


Transmit Character

Function Code = 21H

Entry:
- AH = function code
- AL = character to send
- DX = port number

Exit:

AH =	
0H	command successful
39H	no DSR or CTS
3CH	no DSR
3BH	no CTS
42H	monitor mode inactive
41H	invalid port address
97H	time-out

In order to use this function, the monitor mode must have been activated by issuing an INT 14H function code 24H.

The character in the AL register is sent to the communications port as soon as the port is able to receive data for transmission.

Receive Character

Function Code = 22H

Entry:
- AH = function code
- DX = port number

Exit:

AH =	
0H	command successful
3DH	framing and parity error

```
          3EH   overrun error
          3FH   framing error
          40H   parity error
          41H   invalid port number
          42H   monitor mode not active
          96H   ring buffer overflow
          97H   time-out
     AL = character received
```

The receive ring buffer is examined to determine if any characters have been received from the specified communication port. If a character is present, it is returned in the AL register. If no characters have been received, the function waits for a character until a time-out occurs. The time-out duration is implementation specific.

Return Port Status

Function Code = 23H

Entry:
     AH = function code
     DX = port number

Exit:
     AH = line status
     AL = modem status

The line and modem status registers are read, and control is returned to the caller. The following describes the meaning of the bit settings in the AH and AL registers. The definition is true if the bit is set to one.

LINE STATUS

Bits	Definition
7	time-out error
6	transfer shift register
5	transfer holding register
4	break-detect error
3	framing error
2	parity error
1	overrun error
0	data ready

MODEM STATUS

Bits	Definition
7	received line signal detect

```
  6     ring indicator
  5     data set ready
  4     clear to send
  3     delta receive line signal detect
  2     trailing edge ring detect
  1     delta data set ready
  0     delta clear to send
```


Set Monitor Mode

Function Code = 24H

Entry:
```
     AH = function code
     AL = 0     no individual status
          1     individual status
     DX = port number
```

Exit:
```
     AH =       status
           0H  successful
          3AH  illegal value in AL
          41H  invalid port address
          64H  monitor mode already on
```

This function sets up the specified serial port to start accepting interrupts. Any input or output buffers are cleared. The setting in register AL indicates how the port status is to be maintained. A value of 0 causes the status to be maintained as a single value. A value of 1 will maintain status for every byte in the receive buffer.

Clear Buffers

Function Code = 25H

Entry:
```
     AH = function code
     AL = 0     clear buffers only
          1     clear buffers and deactivate
     DX = port number
```

Exit:
```
     AH =      status
           0H   successful
          3AH   illegal value in AL
          41H   invalid port number
          42H   monitor mode off
```

Any buffers associated with the port are cleared. If the AL register value is one, the interrupts for the port are turned off by resetting the PIC.

Return Buffer Character Count

Function Code = 27H

Entry:
```
AH = function code
DX = port number
```

Exit:
```
AH = status
0H              successful
41H             invalid port number
42H             monitor mode not active
AL = number of elements in input buffer
```

The number of characters in the input buffer are returned in the AL register. No other action is taken on the port.