CHAPTER 9: INTERRUPT MANAGEMENT & MICE

Introduction

In an IBM® PC-AT™ type of computer system there are 15 interrupt signal lines of the IRQ type. The term IRQ stands for Interrupt ReQuest. An IRQ is typically generated to signal the CPU that an I/O operation needs attention. When one of these hardware interrupts occur, the CPU temporarily sets aside whatever it is doing and executes a short set of instructions associated with the interrupt.

Do not confuse this type of interrupt with software interrupts such as INT10h or INT21h. When the CPU encounters a software interrupt instruction, calling the sub-procedure associated with the interrupt vector is the next thing to do in the normal flow of execution, so there is nothing to set aside. An IRQ type of interrupt occurs from an external stimuli produced by a devices such as a serial and parallel ports, buss mouse cards, disk and tape drive controllers, certain video display cards and other special purpose I/O adapters.

These devices use interrupt-based operation because it is the only practical way for one CPU to manage a number of events in a timely manner. For the CPU to have to poll a range of I/O devices is inefficient and inflexible. The following list shows the use of each IRQ, its associated interrupt vector number, and how it is accessed.

IRQ0 (Interrupt Vector 8)

This is the timer tick interrupt. It occurs approximately 18 times a second from a clock circuit within the computer system and is used to maintain a system clock. PC-MOS uses this interrupt as one point at which to instigate a task switch.

IRQ1 (Interrupt Vector 9)

An IRQ1 is generated by the master console's keyboard controller chip to indicate that a keyboard scan code is available.

IRQ2 (Interrupt Vector 0A)

This IRQ is typically available for general use through the expansion bus.

IRQ3 (Interrupt Vector 0B)

When a 02F8 serial port is installed and programmed in an IRQ mode, IRQ3 is used. It can also be accessed through the expansion bus.

IRQ4 (Interrupt Vector 0C)

IRQ4 is used by a 03F8 serial port when that chip is programmed in an IRQ mode. It is also accessible through the expansion bus.

IRQ5 (Interrupt Vector 0D)

This IRQ is typically available for general use through the expansion bus.

IRQ6 (Interrupt Vector 0E)

IRQ6 is used by some disk controllers and is accessed through the expansion bus.

IRQ7 (Interrupt Vector 0F)

This IRQ is typically available for general use through the expansion bus.

For IRQ lines which are available on the expansion bus, add-in adapter cards can use these signal lines to gain the CPU's immediate attention. Although it is technically possible for different devices to use the same IRQ signal line, this type of sharing is typically not done due to the wide variability in the types of hardware devices involved and the associated software drivers. Each device which requires the use of an IRQ line must be given exclusive use of that line.

Excepting IRQ0 and IRQ1, when a device driver or application acquires an IRQ interrupt vector, it does so with the presumption that it is the only one. Whereas with IRQ2 through IRQ7, sharing (or chaining) is not done, the obverse is true with IRQ0 and IRQ1 since the hardware involved is strictly defined. An IRQ0 occurs roughly 18 times a second from the system's clock circuitry and an IRQ1 occurs each time a scan code is generated from the master console's keyboard.

In certain cases, where both the hardware and supporting software are properly designed, more than one device may share an IRQ. The most common case of this is when a group of UART's (serial port chips) are all on the same board and will all be managed by the same software driver (e.g. $SERIAL.SYS). In such cases, this IRQ must be selected, often by a dip switch on the board, so it is unique -- not conflicting with any other IRQ's used in the system.

IRQ Handlers and the Need for Distribution

When an I/O device generates an IRQ interrupt, the CPU is compelled to set aside the process it is currently performing and branch to the software routine located at the address in the corresponding interrupt vector. For example, if interrupt vector INT0F contains the address 5020:0105 then the program logic at that address will receive control each time an IRQ7 occurs. It is up to the interrupt handling software to save and restore any significant aspects of the machine state so that when the IRQ event has been processed, a return can be made to the process which was in progress when the IRQ occurred.

Hardware interrupts are asynchronous in that they can occur at virtually any time unless they have been temporarily disabled. When it is necessary to disable interrupts, it will only be done for a very short period of time. IRQ interrupts cannot be deferred or ignored, as doing so could result in the loss of data or the loss of control of some external device.

For MOS to provide the access control necessary to manage IRQ type interrupts in a multitasking environment, the kernel must always receive control first when an IRQ occurs. The kernel acts as a distributor and determines whether an interrupt call is to be passed on to the currently active task, another task, or a device driver in the SMP. In most cases, MOS can determine this through monitoring changes to interrupt vectors. When a change is detected in a vector, MOS will store the new value and re-assert its own address again. It is also possible, and sometimes necessary, for a user to make an explicit declaration as to which task is to be associated with an IRQ.

Consider the case where a telecommunications program is being run in task #3 and has programmed the 03F8H serial port to generate an IRQ4 each time a character is received. Since the reception of serial communications data will be asynchronous to MOS's time slicing there will be times when task #3 is the currently active task when an IRQ4 occurs, and there will be times when some other task will be switched in. In order to service the IRQ event, the appropriate software handler must receive control which means that MOS must sometimes do a special purpose task switch to map in task #3.

When an SMP loaded device driver takes control of an IRQ vector there is no need for a special task switch. Upon determining that the recipient of the interrupt call is in non-switched memory, the kernel will pass control of the IRQ event on to the device driver's software handler. However, this would not be the case with a task specific device driver. Being in switched memory, a task specific driver which contains an IRQ handler must rely on the special task switching control of the kernel for proper operation.

Automatic Routing and When it Won't Work

As long as there is no ambiguity, the kernel can automatically route IRQ's to the appropriate task. For example, if task #1 is the only one that has changed, the vector for INT0C then the kernel can safely presume that IRQ4's should go to task #1. When more than one task has set a new value for an IRQ vector, and no explicit IRQ reservation has been registered, there is no way for the kernel to determine which task to give the IRQ event to. In this case, a critical error window will pop up to indicate that an IRQ has occurred for which no safe presumption can be made.

This could happen when, after using a communications program in one task for awhile, you hang up the phone but don't terminate the program. Later, when you bring up this same program in a different task, not remembering it's already up or not realizing that it matters, you will create a situation where two tasks will be trying to control the same serial port and the corresponding interrupt vector. This is due to the fact that, although inactive, the first copy of this program still has the port programmed in IRQ mode and has the interrupt vector hooked.

When the kernel's distribution logic finds two tasks with the same interrupt vector set it will pop up a critical error window which effectively says "You need to tell me which task gets this IRQ". To rectify such a situation, press a key to turn the critical error window off, terminate both programs, and then restart with just one copy.

General IRQ Reservations

Automatic IRQ routing is convenient, especially when only one task is running, but its more reliable and self documenting to issue explicit IRQ reservations. Three subcommands of the MOS.COM system utility are used to establish a reservation, relinquish an existing one, and review the current reservation status. These are: MOS USEIRQ n, MOS FREEIRQ n and MOS IRQ respectively. This set of commands allows the user to manage a system data table which the kernel will use in its IRQ distribution function. It is only necessary to make a reservation for an application or a task specific device driver. If an SMP loaded device driver takes over an IRQ vector, the kernel automatically marks this reservation in the system table.

A MOS USEIRQ n command should be issued before an application is run which changes an IRQ. With the exception of the port specific IRQ reservation method described below, only one task can reserve a given IRQ at one time. If you try to reserve an IRQ that is already reserved by another task or by a device driver, MOS.COM will produce an error message to warn that your reservation did not succeed. In addition, when a failing MOS USEIRQ command is issued from within a batch file an ERRORLEVEL 1 will be produced.

To guard against the conflicts which would arise when more than one user tries to access the same IRQ device, you could create a batch file which would be used when starting the corresponding IRQ based application. For example, to control access to the telecommunications package XYZCOMM, the following batch file could be used:

```
MOS USEIRQ 4
IF ERRORLEVEL 1 ABORT
XYZCOMM
MOS FREEIRQ 4
```

In the case where an application does not establish an IRQ handler itself but, rather, interacts with a device driver which performs the interrupt processing, this method cannot be used. See the flag file technique described in Chapter 4.

Port Specific IRQ Reservations

MOS provides two interface methods to support multiple mice. The port specific reservation method allows serial ports which share an IRQ to be split across tasks. For example, you can cause MOS to pass an interrupt call to task #1 when the 06A0h serial port is the source of an IRQ and pass it to task #2 when the 06A8h port is responsible. In this way, even though devices share an IRQ, they can be made to appear to a task in an isolated manner. Note that the other method, the newer INT 14H interfaced method of multiple mouse support, does not require this task specific IRQ reservation method.

The task receiving the interrupt call for port 06A0h is not affected by the fact that port 06A8h is using the same IRQ. The use of the port specific IRQ reservation scheme and the non port specific is mutually exclusive for a given IRQ. For example, you may reserve IRQ2 for task #0 and you may register IRQ5 with port 06A0h for task #2 and IRQ5 with port 06A8h for task #3. You could not, however, have a general reservation for IRQ5 in any two tasks at the same time. An example of the use of port specific IRQ reservation will be detailed later in this chapter, in the section on mouse support.

Releasing a Reservation

The reservation data for each type of IRQ reservation (general and port specific) are kept in separate tables, so in order to rescind a reservation the corresponding type of FREEIRQ command must be used. For example, to free the reservation made by a MOS USEIRQ 2, the command MOS FREEIRQ 2 would be required. Likewise, in order to counteract a MOS USEIRQ 5 06A0, a MOS FREEIRQ 5 06A0 would be required.

Mice

There are two basic types of mice, the serial mouse which connects through a serial port, and the bus mouse, which uses a small adapter card installed in an expansion slot. The focus here will be on the serial mouse since it is more flexible in terms of possible system configurations. Serial port connections can be extended with standard cables and, when the appropriate steps are taken, multiple mice can be supported in one system through multiport serial cards. In addition, it only takes one expansion slot for a multiport serial card as opposed to the slot-per-mouse with the bus based interface.

A bus mouse can only be used in a PC-MOS system if it is the only one of that type installed. Do not try to plug in more than one bus mouse card thinking you can set them up for different IRQ's. The cards will have the same I/O addresses, so a conflict will exist.

As a security consideration, note that using the system keyswitch to disable the master console's keyboard does not disable a mouse. If you must leave your workstation running a mouse based application, lock the mouse in a drawer. Do not unplug a mouse while a system is running, as this can generate electrical pulses which will upset a computer. Also, since a mouse does not follow a PAMswitch, be careful when a mouse-based application is running in one task while your workstation is watching some other task. The mouse is still active for the unwatched mouse program! If you move it, you will not be able to see what you are actually doing with it.

Mouse Drivers

A mouse driver is used to provide a standard interface to applications which use a mouse. This mouse driver program is basically a hardware interrupt handler coupled with a user interface. When the mouse is moved, interrupts are generated which causes the mouse driver's code to get executed. The driver maintains information on the mouse's position and the state of the mouse buttons.

As supplied by the mouse manufacturer, this driver code can exist in two different forms: a device driver and a TSR program (Terminate and Stay Resident). Its basically the same set of software instructions, there are just two different ways to load it into a computer's memory. An SMP loaded device driver is globally accessible meaning that it is available to all tasks. The TSR version is loaded within the switched memory of the task's TPA area and is only accessible to the task in which it is loaded. The device driver version is typically in a file called MOUSE.SYS and the TSR version in a file called MOUSE.COM.

Even if you only plan to support one mouse on your system, you should use MOUSE.COM rather than MOUSE.SYS. Using MOUSE.SYS can cause confusion unless your system will never have more than one task (e.g. no multitasking or multiuser environment). To understand this, consider the case where one user is operating a mouse with a graphics drawing program at their workstation and another user is running a word processor at theirs. Some applications programs, including certain word processors, will automatically respond to a mouse when they detect an installed mouse driver (e.g. MOUSE.SYS). Having two applications, in two different tasks, both responding to the same mouse is an un-usable environment.

When there is a need to support more than one mouse in a system, isolation factors become very important. MOUSE.SYS provides no isolation and thus allows both programs to "see" the same mouse. If you install MOUSE.COM only in the task where the mouse is to be used, all other tasks will be unaware that there is a mouse in the system.

This is fine for the case of a single mouse per system. However, trying to install more than one copy of MOUSE.COM in a system can again result in confusion, since the mouse driver is designed to find the mouse at a particular serial port. Most serial mouse drivers are designed to operate a mouse connected to either the 03F8h serial port or the port at address 02F8h. An optional command line parameter is used to select which port to use. If you only need to support two mice and the 03F8h and 02F8h ports are available, you can issue a MOS USEIRQ 4 in one task, followed by the installation of MOUSE.COM for COM1. Then do the same thing in the other task for IRQ3 and COM2.

Installing Multiple Mice

In the case where these ports aren't available or more than two workstations will require a mouse, another solution must be found. Prior to release 4.00, multiple mice were supported by customizing or patching the MOUSE.COM TSR program that was supplied with the mouse. This method provided for the need at the time, but has since been superceded by a more flexible approach.

Beginning with release 4.00, a new method has been devised which allows mice to be connected to serial ports which are under the control of an interrupt driven serial driver (e.g. $SERIAL.SYS). This method is more flexible and provides for better overall system throughput. Since the patching method will still be in use for some time however, a discussion of both methods is included.

Patched Mouse Drivers

The SETMOUSE.COM utility is provided to allow a MOUSE.COM TSR driver to be customized with respect to the serial port address and IRQ level that the driver will use. This utility will read the stock copy of MOUSE.COM, make the required modifications and produce a new COM file with a unique file name such as M06A0I2.COM for a driver which uses port 06A0H and IRQ2. For each task which will use a mouse, a customized copy of MOUSE.COM with a unique port address must be created. Through the use of port specific IRQ reservations, it is possible for a group of mice to share an IRQ level (see the following example).

Some dumb multiport cards provide only the bare minimum number of serial port connections (TX, RX and ground -- a three wire set). This is all that is required for a serial terminal type of workstation since a software form of handshaking can be used, such as XPC. However, a serial mouse will typically need more than just a 3-wire port to operate properly. You must use a dumb multiport card which supports all of the signal lines required by your particular type of mouse. In addition, since the mouse driver's interrupt handler will need to be able to interact directly with the serial port, a dumb type of serial card must be used for a mouse when using the patched mouse driver method.

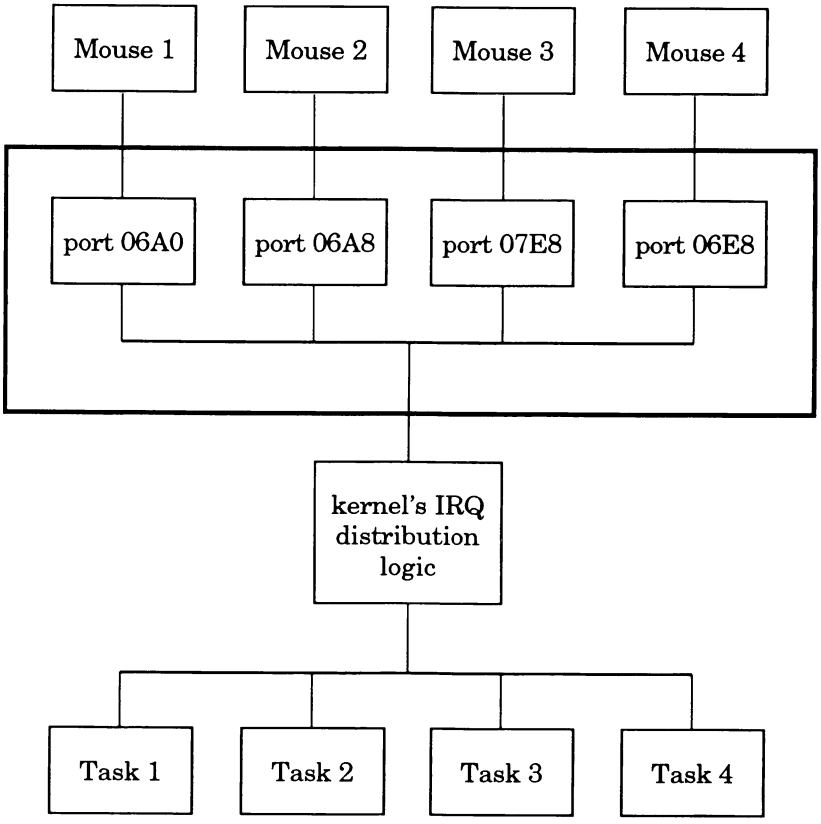

Figure 9 - 1

Through the use of MOS's port specific IRQ reservation scheme, multiple mice may share an IRQ level.

Figure 9-1 shows a system which supports four serial mice. Task #1's startup batch file would include the command MOS USEIRQ 2 06A0 to establish the port specific IRQ reservation. Following this would be a M06A0I2 to invoke a patched version of the MOUSE.COM driver. Task #2's startup batch file would contain MOS USEIRQ 2 06A8 and M06A8I2, with task's 3 and 4 following suit for their respective ports. In this way, each task presents mouse aware applications with the standard mouse driver interface which it expects to see, but each driver is isolated from its replica in the other tasks.

A variety of peculiarities bear mention in a discussion of mouse usage. In the case where an application contains its own code to interface to a mouse, rather than make use of the services of a standard mouse driver, there could be problems. One such application used internal mouse driver code for a serial mouse but used MOUSE.COM when configured for a bus mouse. To operate under MOS with a multiple mouse configuration, this package could be setup for a bus mouse just to make it interface with MOUSE.COM. Even though the installed MOUSE.COM was actually interfacing to serial mouse and was patched for a different port address and IRQ level, the application was none the wiser.

NOTE for release 4.00: A change has been made which pertains to the support of mice through the patched mouse driver method. Be sure to read the section at the end of this chapter on the MOSADM VIRQ command.

INT 14H Interfaced Mouse Drivers

This method takes advantage of a new interface technique which began when support was added for the PS/2 series of machines by mouse manufacturers. Note that this does not mean that a PS/2 machine must be used for this mouse support technique to work. Through MOS's emulation capabilities, any hardware platform on which MOS will operate can be used. MOS provides this emulated interface through a device driver called $MOUSE.SYS. If you are using an older version of a mouse driver which does not have PS/2 compatibility, you will need to upgrade to a version which does.

It is now possible to support multiple tasks, each with their own MOUSE.COM installed, without having to patch each driver. There are also added benefits. Overall system throughput is increased since no asynchronous task switches must be done to handle mouse interrupts. Also, in contrast to the patched mouse driver scheme where the MOUSE.COM TSR must have direct access to the serial port, this method uses ports which are interfaced through a serial communications device driver.

While $SERIAL.SYS is a typical example of this type of driver, it will not be the one most often used with this mouse interface method. When SunRiver workstations are used, the serial device driver interface code is included within the SRTERM.SYS driver and when VNA/IONA workstations are used, the serial interface code is contained within the VNA.SYS driver. These device drivers contain the equivalent of a $SERIAL.SYS which has been customized to manage the particular types of serial ports within these workstations. A $SERIAL.SYS device driver, or its equivalent, can also be used where appropriate.

The $MOUSE.SYS device driver must be installed from a CONFIG.SYS declaration which occurs after all serial drivers have been declared. Then, in order to activate this type of mouse support within each task which will use it, an unpatched copy of the MOUSE.COM TSR must be installed. Next, the following command must be issued, where "p" is the logical port number and "r" is the baud rate (typically 1200):

```
MOS MOUSE p,r
```

The MOSADM VIRQ Command

When MOS must make an asynchronous task switch to handle an IRQ event which is for a task other than the current task, this task switch has involved a remapping of the video buffers as well as the task RAM and context area. The switching of video buffers was done primarily for the sake of the patched mouse driver method of multiple mouse support. When multiple mice are supported through the INT 14H interface method, there is no need to do any form of task switch to handle mouse interrupts.

One consequence of switching the video buffers during an asynchronous task switch is that the PAMswitching procedure must disable interrupts for relatively long periods of time. The PAMswitching process must protect the integrity of the video state when making certain transitionary adjustments to that state. Consequently, when a PAMswitch is done it is possible for IRQ's to be lost. While this will not noticeably affect the operation of mice, it can be disastrous when IRQ's are being generated in a telecommunica- tions session.

In order to prevent a PAMswitch from interfering with communications, a change has been made in release 4.00 of MOS. An IRQ-based task switch no longer makes any dependencies on the video state. This means that the patched mouse driver scheme will no longer work (unless special action is taken -- see below) but PAMswitching will not cause IRQ's to be missed.

If you still wish to use the patched mouse driver method rather than convert to the INT 14H method, you will need to issue a new MOSADM utility command known as MOSADM VIRQ. Placing MOSADM VIRQ ON in the AUTOEXEC.BAT file of your foreground task will restore MOS's IRQ handling to the condition required for patched mouse drivers.