Mouse Input

From OSDev Wiki

Contents

PC Mouse Interfaces

Current PCs generally use PS2 mice, or a similar format that emulates a PS2 mouse. Serial mice are a much older technology that is no longer common.

USB Mouse

A USB mouse generally emulates a PS2 mouse, except that it generates IRQs from the USB bus, and not IRQ 12.

PS2 Mouse -- Basic Operation (Microsoft compliant)

Once a mouse has been initialized (see below), a mouse sends 3 or 4 byte packets to communicate mouse movement, and mouse button press/release events. These packets show up asynchronously as data on IO port 0x60. Even though the data shows up on port 0x60 (the keyboard port), it does *not* trigger an IRQ1. The only way that you will know that mouse data has arrived is if you handle an appropriate IRQ (generally IRQ12), or if you occasionally poll bit number 0 (value=1) of port 0x64, to see if data is available on port 0x60. It is not necessary to handle all of the following things inside the driver, but doing so can make things work faster.

Keyboard/Aux Data Bit

Since both keyboard and mouse data is showing up to be read on port 0x60, it is necessary to be able to tell which is which. To tell if there is any available data on port 0x60 at all, it is necessary to read a byte from port 0x64. In that byte from port 0x64, bit number 0 (value=1) indicates that a byte is available to be read on port 0x60. An additional bit, bit number 5 (value=0x20), indicates that this next byte came from the mouse, if the bit is set. If you look at RBIL, it says that this "mouse bit" is MCA specific, but this is no longer true. All PCs that support PS2 mice use this bit to indicate that the incoming byte was generated by the auxiliary PS2 input port.

PS2 Mouse Subtypes

There are several types of mice, but they can be separated into two groups, depending on whether they have a scroll wheel. Mice with scroll wheels can send an additional byte in each mouse packet, indicating the status of mouse wheels and extra buttons.

Mice with no scroll wheels use 3 byte packets exclusively. Mice with scroll wheels and up to 5 buttons (currently) can send 4 byte packets, if they are initialized properly.

Mouse Packet Info

For even greater detail on Microsoft compliant PS2 mouse packets than what is presented next: reference the links below, or search for the Adam Chapweske article. For mice that are not MS-compliant, see the Linux article.

Timing/Generation of Mouse Packets

A mouse is usually initialized to generate movement packets at a particular rate. The default rate is 100 packets per second if the mouse is being moved. A mouse also generates a packet if a button is either pressed or released. If the mouse is not moving and no buttons are being clicked a mouse will not generate any automatic packets.

Format of First 3 Packet Bytes

Even if your mouse is sending 4 byte packets, the first 3 bytes always have the same format. The first byte has a bunch of bit flags. The second byte is the "delta X" value -- that is, it measures horizontal mouse movement, with left being negative. The third byte is "delta Y", with down (toward the user) being negative. Typical values for deltaX and deltaY are one or two for slow movement, and perhaps 20 for very fast movement. Maximum possible values are +255 to -256 (they are 9-bit quantities, two's complement).

byte 1:

Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn

byte 2:

X movement

byte 3:

Y movement

The top two bits of the first byte (values 0x80 and 0x40) supposedly show Y and X overflows, respectively. They are not useful. If they are set, you should probably just discard the entire packet.

Bit number 5 of the first byte (value 0x20) indicates that delta Y (the 3rd byte) is a negative number, if it is set. This means that you should OR 0xFFFFFF00 onto the value of delta Y, as a sign extension (if using 32 bits).

Bit number 4 of the first byte (value 0x10) indicates that delta X (the 2nd byte) is a negative number, if it is set. This means that you should OR 0xFFFFFF00 onto the value of delta X, as a sign extension (if using 32 bits).

Bit number 3 of the first byte (value 0x8) is supposed to be always set. This helps to maintain and verify packet alignment. Unfortunately, some older mice (such as 10 year old Microspeed 2 button trackballs) do not set this bit. RBIL claims that this bit should be 0, but it is wrong.

The bottom 3 bits of the first byte indicate whether the middle, right, or left mouse buttons are currently being held down, if the respective bit is set. Middle = bit 2 (value=4), right = bit 1 (value=2), left = bit 0 (value=1).

Formats of Optional 4th Packet Byte

If the mouse has been initalized so that its mouseID is 3 or 4, it will send a 4th byte in each packet. On all current mice, the top two bits should be ignored. On some mice, the bits will flip between 0 and 1, based on scroll wheeel movement. If the mouse has a 4th and 5th mouse button, then their state is indicated by bit 4 (value=0x10), and bit 5 (value=0x20), respectively. Note: if the buttons *do not* exist, then these bits may flip based on scroll wheel movement! (ie. Be careful that this does not generate spurious "mouse button click" events for buttons that do not exist.)

The bottom nibble (4 bits) of the 4th byte will be one of the following values:

- 0 -- no scroll wheel movement
- 1 -- vertical scroll up one click
- 0xF -- vertical scroll down one click

- 2 -- horizontal scroll right one click
- 0xE -- horizontal scroll left one click

When only the vertical scrollwheel exists or has been activated, the 4th packet byte simply contains:
byte 4:

Z movement

However, if the mouse has extra buttons and gets initialized properly, then it will look like this:
byte 4:

Mostly 0	Mostly 0	5th btn down (pressed)	4th btn down (pressed)	Z3	Z2	Z1	Z0

Where Z0 though Z3 is a 4 bit signed value of the Z movement (scrollwheel), with the values mentioned above (0, 1, 2, 0xE, 0xF) and with the meanings listed above.

Non-Linear Movement

When a user is trying to point at something on a screen, they will quickly move the mouse in the general direction of the target, and then slow down to accurately point at it. The deltaX/Y values from the mouse packets can be used directly, but doing that will force the user to move the mouse very far to get the cursor from one area of the screen to another.

A better answer may be to scale the mouse movement by additional multiples, based on how fast the mouse is moving. If the mouse is moving slowly, the scale factor can be 1 -- to allow the highest possible sensitivity. When the mouse is (or has been) moving fast, the scale factor could be 5 or 10, to allow the cursor to move across large distances on the screen quickly.

There are many ways to do such scaling, obviously. If you are coding the driver in assembler, one suggestion might be to use the BSR command to generate an approximate log base2 of deltaX plus deltaY, and use that as your scaling factor.

Doubleclicks

If you intend to implement a doubleclick, you will need to record timestamps of mousedown events to high accuracy. The only way to know if a doubleclick has occurred is to know how long ago the previous click of the mouse button happened. If the delay between the two clicks was less than some threshhold, then it was a doubleclick. The problem here is that you need to be able to measure very short delays, and there may be extremely long delays between clicks (days, perhaps) -- long enough that your timestamp measurement might overflow. This needs to be handled carefully.

Timeouts/Mouse Disconnect

If a mouse is operating normally but is not being moved or clicked, it will not send any packets. If a mouse has been unplugged, it will also not send any packets. If you want to support "hot plugging" of PS2 mice, then you need to know when a mouse has become disconnected, because you will need to reinitialize it. If you have not seen any packets from the mouse for awhile, you can query the mouse to see if it is still alive. A convenient way to do this is to send a special request for a single mouse packet (mouse command 0xEB). You will get back an ACK (0xFA) from the mouse, and then a mouse packet (with everything set to 0, probably). Please note that you need to make sure that the 0xFA does not cause a misalignment of your input mouse packet. Also please note that if the contents of the mouse packet are *not* 0, then that means that somehow your mouse packets have become disabled.

In general, it is a good idea to also have timeouts everywhere that the system is waiting for a response from the mouse, because it may never come.

PS2 Mouse Commands

After the PS2 Aux port has been enabled, you can send commands to the mouse. It is recommended to disable automatic packet streaming mode while "reprogramming" the mouse. You can do this by either sending command 0xF5 to the mouse, or disabling the "master mouse clock" by setting bit 5 of the Compaq Status byte (see below).

Waiting to Send Bytes to Port 0x60 and 0x64

All output to port 0x60 or 0x64 must be preceded by waiting for bit 1 (value=2) of port 0x64 to become clear. Similarly, bytes cannot be read from port 0x60 until bit 0 (value=1) of port 0x64 is set. See PS2 Keyboard for further details.

0xD4 Byte, Command Byte, Data Byte

Sending a command or data byte to the mouse (to port 0x60) must be preceded by sending a 0xD4 byte to port 0x64 (with appropriate waits on port 0x64, bit 1, before sending each output byte). Note: this 0xD4 byte does *not* generate any ACK, from either the keyboard or mouse.

Wait for ACK from Mouse

It is required to wait until the mouse sends back the 0xFA acknowledgement byte after each command or data byte before sending the next byte (Note: reset commands might not be ACK'ed -- wait for the 0xAA after a reset). A few commands require an additional data byte, and both bytes will generate an ACK.

Useful Mouse Command Set

- Note: remember that the mouse responds to all command bytes and data bytes with an ACK (0xFA).

Hex value	Meaning	Description
0xFF	**Reset**	**The mouse probably sends ACK (0xFA) plus several more bytes, then resets itself, and always sends 0xAA.**
0xFE	**Resend**	**This command makes the mouse send its most recent packet to the host again.**
0xF6	**Set Defaults**	**Disables streaming, sets the packet rate to 100 per second, and resolution to 4 pixels per mm.**
0xF5	**Disable Packet Streaming**	**The mouse stops sending automatic packets.**

0xF4	**Enable Packet Streaming**	**The mouse starts sending automatic packets when the mouse moves or is clicked.**
0xF3	**Set Sample Rate**	**Requires an additional data byte: automatic packets per second (see below for legal values).**
0xF2	**Get MouseID**	**The mouse sends sends its current "ID", which may change with mouse initialization.**
0xEB	**Request Single Packet**	**The mouse sends ACK, followed by a complete mouse packet with current data.**
0xE9	**Status Request**	**The mouse sends ACK, then 3 status bytes. See below for the status byte format.**
0xE8	**Set Resolution**	**Requires an additional data byte: pixels per millimeter resolution (value 0 to 3)**

Additional Useless Mouse Commands

Hex value	Meaning	Description
0xF0	**Set Remote Mode**	**The mouse sends ACK (0xFA) and then reset its movement counters, and enters remote mode**
0xEE	**Set Wrap Mode**	**The mouse sends ACK (0xFA) and then reset its movement counters, and enters wrap mode**
0xEC	**Reset Wrap Mode**	**The mouse sends ACK, and then enters the last mode, before entering wrap mode, it also resets its movement counters**
0xEA	**Set Stream Mode**	**The mouse sends ACK (0xFA) and then reset its movement counters, and enters reporting mode**
0xE7	**Set Scaling 2:1**	**The mouse sends ACK and sets non-linear scaling "2:1"**
0xE6	**Set Scaling 1:1**	**The mouse sends ACK and sets normal linear scaling "1:1"**

The status bytes look like this:
Byte 1:

Always 0	**mode**	**enable**	**scaling**	**Always 0**	**left btn**	**middle**	**right btn**

Byte 2:

resoltion

Byte 3:

sample rate

Mode: if it is 1, the current mode is remote mode; if 0 then it is stream mode
Enable: if it is 1, then data reporting is enabled; if 0 then data reporting is disabled
Scaling: if it is 1, scaling 2:1 is enabled; if 0 then scaling 1:1 is enabled.

Resolution, Scaling and Sample Rate

Definitions:

- Resolution: DeltaX or DeltaY for each millimeter of mouse movement.
- Scaling: Apply a simple non-linear distortion to mouse movement (see Non-Linear Movement, above).
- Sampling Rate: Packets the mouse can send per second.

Resolution:

value	resolution
0x00	1 count /mm
0x01	2 count /mm
0x02	4 count /mm
0x03	8 count /mm

Scaling can either be "1:1" (linear = no scaling) or "2:1" (non-linear). This is the non-linear scaling:

Movement Counter	Reported Movement
0	0
1	1
2	1
3	3
4	6
5	9
more than 5	2 * Movement Counter

Sample Rate can have the following values: (all values are Decimal, NOT hex)

value	Samples pr second
10	10
20	20
40	40

60	**60**
80	**80**
100	**100**
200	**200**

- Note: human eyes don't see movement faster than 30 samples per second, and human fingers cannot click a button

that fast either. A sample rate lower than 30 will cause visibly jerky mouse movement, and may miss mousedown events. A sample rate significantly higher than 30 will waste precious I/O bus bandwidth. You may test these things for yourself, but sample rates of 10, 20, 100, or 200 are generally not recommended.

Initializing a PS2 Mouse

The PS2 mouse port on a PC is attached to the auxiliary input of the PS2 keyboard controller. That input might be disabled at bootup, and needs to be enabled. It is usually also desirable to have the mouse generate IRQ12 interrupts when it sends bytes through the keyboard controller to IO port 0x60. Additionally, it is necessary to tell the mouse to enable the transmission of packets. Optionally, you may also want to enable additional mouse features, such as scroll wheels, faster response times, increased resolution, or additional mouse buttons.

PS/2 Device Unplugging/Hot Plugging

Some idiot who created the PS/2 device specs did not specify that PS/2 devices can be unplugged and replugged while the computer remains turned on ("hot plugging"). A long time ago, some other idiots actually designed motherboards that would be slightly damaged if PS2 hot plugging occurs. However, mice and keyboards have cords that were made to be tripped over, and sometimes it is very logical to try moving a mouse from one machine to another, temporarily, without powering both systems down. So all computers made in the last 15 years should, in fact, support hot plugging of PS2 devices. When a mouse is plugged into a running system it may send a 0xAA, then a 0x00 byte, and then go into default state (see below).

Set Compaq Status/Enable IRQ12

On some systems, the PS2 aux port is disabled at boot. Data coming from the aux port will not generate any interrupts. To know that data has arrived, you need to enable the aux port to generate IRQ12. There is only one way to do that, which involves getting/modifying the "compaq status" byte. You need to send the command byte 0x20 ("Get Compaq Status Byte") to the PS2 controller on port 0x64. If you look at RBIL, it says that this command is Compaq specific, but this is no longer true. This command does *not* generate a 0xFA ACK byte. The very next byte returned should be the Status byte. (Note: on some versions of Bochs, you will get a *second* byte, with a value of 0xD8, after sending this command, for some reason.) After you get the Status byte, you need to set bit number 1 (value=2, Enable IRQ12), and clear bit number 5 (value=0x20, Disable Mouse Clock). Then send command byte 0x60 ("Set Compaq Status") to port 0x64, followed by the modified Status byte to port 0x60. This might generate a 0xFA ACK byte from the keyboard.

Aux Input Enable Command

Send the Enable Auxiliary Device command (0xA8) to port 0x64. This will generate an ACK response from the *keyboard*, which you must wait to receive. Please note that using this command is not necessary if setting the Compaq Status byte is successful -- but it does no harm, either.

Mouse State at Power-on

When the mouse is reset, either by applying power or with a reset command (0xFF), it always goes into the following default state:

- packets disabled
- emulate 3 button mouse (buttons 4, 5, and scroll wheels disabled)
- 3 byte packets
- 4 pixel/mm resolution
- 100 packets per second sample rate

MouseID Byte

During initialization, a mouse indicates that it has various features (a scroll wheel, a 4th and 5th mouse button) by changing its mouseID in response to initialization commands. So you send a set of mouse commands, and then ask for the mouseID byte with the Get MouseID command (0xF2). If the mouseID changed from its previous value, then the mouse has changed modes. The mouseID byte is always the next byte sent after the ACK for the Read MouseID command. At initialization, the mouseID is always 0. Other current legal values are 3 and 4.

Init/Detection Command Sequences

If you would like more than just 3 buttons, you will have to use the following sequence(s). If the first sequence is accepted: the number of bytes in the mouse packets changes to 4, the scroll wheel on the mouse becomes activated, and the mouseID changes from 0 to 3.

The first magic sequence goes like this:

- set sample rate to 200
- set sample rate to 100
- set sample rate to 80
- get the new id to verify acceptance

After using the above sequence to activate the scroll wheel, the 4th and 5th mouse buttons can be activated with the following additional magic sequence:

- set sample rate to 200
- set sample rate to 200
- set sample rate to 80
- get the new id to verify acceptance

If this second sequence is accepted, the returned mouse ID value changes from 3 to 4.

Enable Packets

After the mouse has been initialized to your desired mouseID value, its Samplerate is probably 80 samples per second, its Resolution is probably 4 pixels/mm, and packets are still disabled. You may want to modify the Samplerate and Resolution, and then send a 0xF4 command to the mouse to make the mouse automatically generate movement packets.

Streaming Advantages and Disadvantages

Instead of enabling automatic streaming packet mode, it is possible to request mouse packets one at a time. Doing this has some advantages over streaming packet mode. You may see that you need to send or receive at least 4 extra bytes over the I/O bus in order to get each 0xEB command to the mouse, and the I/O bus is very slow. On the other hand, a typical streaming mode probably sends hundreds more mouse packet bytes over the I/O bus than you need, every second -- so if there is a disadvantage, it is probably small.

One of the biggest problems with streaming mode is "alignment" -- the packets were never defined to have an obvious boundary. This means that it is very easy to lose track of which mouse byte is supposed to be the first byte of the next packet. This problem is completely avoided if you specifically request single packets (instead of using streaming mode) because every packet begins with an ACK (0xFA), which is easily recognizable.

PC Serial Mouse

For info on running a serial mouse on an RS232 port, see this document (http://freedos-32.sourceforge.net/showdoc.php?page=sermouse) .

Mac Mouse Interface

(stub - to be written later)

See Also

Threads

- SANiK's mouse code (http://www.osdev.org/phpBB2/viewtopic.php?t=10247)
- Mouse driver code in assembly (http://www.osdev.org/phpBB2/viewtopic.php?t=24277)
- PS2 mouse IRQ (http://www.osdev.org/phpBB2/viewtopic.php?t=8323)
- PS2 mouse links (http://www.osdev.org/phpBB2/viewtopic.php?t=6942)

External Links

- Adam Chapweske on PS/2 Mouse and Keyboard protocols (http://www.computer-engineering.org)
- Linux PS2 mouse article with non-Microsoft mouse info (http://www.win.tue.nl/~aeb/linux/kbd/scancodes-13.html)
- Tutorial with sample C code (http://houbysoft.com/download/ps2mouse.html)
- IBM's Keyboard and Auxiliary Device (mouse) Controller documentation (http://www.mcamafia.de/pdf/ibm_hitrc07.pdf)

Retrieved from "http://wiki.osdev.org/index.php?title=Mouse_Input&oldid=17336"

Category: Human Interface Device