25 Mouse Programming

"Show respect for all people."

As everyone knows, mouse is one of the inputting devices. In this chapter, I explain interrupts for mouse programming and a few concepts regarding mouse programming. In the graphics programming, we can see more examples. To work with mouse, we must have mouse driver file mouse.com.

25.1 Mouse Interrupts

int 33h is the mouse interrupt. It has so many functions. Certain functions will be available only to certain drivers. A complete interrupt specification is available on Ralf Brown's Interrupt List.

25.2 Useful Mouse functions

25.2.1 Mouselib.h

```
#ifndef __MOUSELIB_H

#define LFTCLICK  (1)

int InitMouse( void );
void ShowMousePtr( void );
void MoveMousePtr( int x, int y );
void RestrictMousePtr( int x1, int y1, int x2, int y2 );
void HideMousePtr( void );
void GetMousePos( int *mbutton, int *x, int *y );

#endif
```

25.2.2 Mouselib.c

```
#include  "mouselib.h"

#pragma inline

/*-----------------------------------------------
      InitMouse - Initializes Mouse.
            Returns 0 for success.        */
```

```
int InitMouse( void )
{
   asm {
        MOV AX, 0;
        INT 33h;
       }
   return;
}  /*--InitMouse( )---*/

/*-----------------------------------------------
      ShowMousePtr - Shows Mouse Pointer. */

void ShowMousePtr( void )
{
   asm {
        MOV AX, 1h;
        INT 33h;
       }
}  /*--ShowMousePtr( )----*/

/*-----------------------------------------------
      HideMousePtr - Hide Mouse Pointer.  */

void HideMousePtr( void )
{
    asm {
         MOV AX, 2h;
         INT 33h;
       }
}  /*--HideMousePtr( )-----*/

/*-----------------------------------------------
      MoveMousePtr - Move Mouse Pointer
            to (x, y).              */

void MoveMousePtr( int x, int y )
{
    asm {
         MOV AX, 4h;
         MOV CX, x;
         MOV DX, y;
         INT 33h;
       }
}  /*--MoveMousePtr( )-----*/
```

```
/*-----------------------------------------------
      RestrictMousePtr - Restrict Mouse Pointer
            to the specified coordinates  */

void RestrictMousePtr( int x1, int y1, int x2, int y2 )
{
   asm {
        MOV AX, 7h;
        MOV CX, x1;
        MOV DX, x2;
        INT 33h;
        MOV AX, 8h;
        MOV CX, y1;
        MOV DX, y2;
        INT 33h;
       }
}  /*--RestrictMousePtr( )--------*/

/*-----------------------------------------------
      GetMousePos - Gets Mouse position &
            mouse button value.           */

void GetMousePos( int *mbutton, int *mx, int *my )
{
   asm {
        MOV AX, 3h;
        INT 33h;

        MOV AX, BX;
        MOV BX, mbutton;
        MOV WORD PTR [BX], AX;

        MOV BX, mx;
        MOV WORD PTR [BX], CX;

        MOV BX, my;
        MOV WORD PTR [BX], DX;
       }
}  /*--GetMousePos( )------*/
```

25.2.3 Mouselib.lib

When you compile the above Mouselib.c file to a library file for Small memory model, you will get Mouselib.lib file. You can use the library – Mouselib.lib in your projects..

25.3 Mouse Function 0Ch

Function 0Ch that is available with int 33h is very much useful. And almost all game programmers and graphics programmers use it. The beauty of this function is that it allows us to install our own handler, so that whenever the int 33h is generated, our own handler will be automatically called. In other words, instead of `setvect( )`, we have to use function 0Ch for installing our own handler.

Installing our own mouse handler to get mouse input is referred as *Event Mode*. Game programmers prefer Event Mode and they use circular queue to store the events as inputs. The following codes by **Alexander J. Russell** illustrate the concept.

First of all, we have to initiate the normal int 33h mouse driver to install a "stub" program that calls our real mouse handler. The "stub" is written in ASM.

```
;*************************************************************
;*                                                           *
;* Assembly language hook for CMOUSE library event handler   *
;*                                                           *
;* Assemble with /Ml switch                                  *
;*                                                           *
;*************************************************************
; real code for real men
; adjust for proper memory model
.MODEL SMALL,C

.CODE
      PUBLIC mouse_event_func,mouse_int

mouse_event_func DD ?


mouse_int PROC FAR
          PUSHF
          CALL CS:[mouse_event_func]
          RET
mouse_int ENDP

END
;-----------------------------------------------
```

The above assembler function `mouse_int()` is called by the int33h driver. `mouse_int()` in turn calls whatever function `mouse_event_func()` points to. `mouse_event_func()` is a pointer to a function and it is not itself a function.

Following is the C code to use the mouse.

```
#define ESC 27

short mouse_x, mouse_y;
short mouse_present;
short mouse_hidden=0;
short button_stat=0;
unsigned short flags;

extern void far *far mouse_event_func;
void mouse_int( void );

typedef struct
  {
     unsigned int flags, x, y, button_flag;
  } mouse_info_t;

#define MAX_MOUSE_EVENTS 10
#define MOUSE_MOVE   1
#define MOUSE_L_DN   2
#define MOUSE_L_UP   4
#define MOUSE_R_DN   8
#define MOUSE_R_UP   16

#define EVENT_MASK   31   /* the logical OR of the 5 above vars */

mouse_info_t mouse_info[MAX_MOUSE_EVENTS];      /* Circular Queue */
int head=0;
int tail=0;

/*-----------------------------------------------
      mouse_handler - the low level interrupt
                  handler calls this     */

void far interrupt mouse_handler(void)
{
   /* save info returned by mouse device driver */
   asm {
         mov   flags,   ax
         mov   mouse_x, cx
         mov   mouse_y, dx
         mov   button_stat, bx
       }

      // place the mouse information in a circular queue
```

```
      mouse_info[tail].x = mouse_x;
      mouse_info[tail].y = mouse_y;
      mouse_info[tail].button_flag = button_stat;
      mouse_info[tail].flags = flags;

      tail++;
      if ( tail == MAX_MOUSE_EVENTS )
            tail=0;
   if ( tail == head )
    {
      head++;
      if ( head == MAX_MOUSE_EVENTS )
         head=0;
    }

} /*--interrupt mouse_handler( )--------*/

/*-----------------------------------------------
      init_mouse - is there a mouse, install int
                   handlers          */
short init_mouse( void )
{
   unsigned short c_seg, c_off;

   asm{
         xor   ax, ax
         int   033h

         /* note BX holds number of buttons, but we don't care */
         mov   mouse_present, ax
      }

   if ( mouse_present )
    {
      /* install our own handler */
      mouse_event_func = mouse_handler; /* global func pointer */

      /* install mouse_int as mouse handler, which will call
         mouse_handler */

      c_seg = FP_SEG(mouse_int);
      c_off = FP_OFF(mouse_int);
      asm{
            mov   ax, c_seg
            mov   es, ax
            mov   dx, c_off
            mov   ax, 0ch
```

```
            mov   cx, EVENT_MASK
            int 033h
         }

      /* set mouse x, y limits */
      asm{
            mov   ax, 7
            mov   cx, 0
            mov   dx, 359
            int 033h

            mov   ax, 8
            mov   cx, 0
            mov   dx, 239
            int 033h

            /* set initial mouse_x, mouse_y */
            mov   ax, 3
            int 033h

            mov   mouse_x, cx
            mov   mouse_y, dx
         }
     }
   return(mouse_present);
} /*--init_mouse( )---------*/

/*-------------------------------------------------
      deinit_mouse - deinstall our mouse handler
                                      */
void deinit_mouse( void )
{
   if ( mouse_present )
    {
      /* deinstall our mouse handler by making int 33 never call it */
      asm{
            mov   ax, 0ch
            xor   cx, cx     /* mask == 0, handler never called */
            int 033h

            /* reset mouse driver */
            xor   ax, ax
            int   033h
         }
      }
} /*--deinit_mouse( )--------*/
```

Assembler function `mouse_int( )` calls `mouse_event_func( )` whenever the mouse is moved, or a button is pressed or released. `mouse_event_func( )` points to `mouse_handler( )` which queues up the mouse events.

25.4 Request Mode or Event Mode?

Request Mode is the one in which we call mouse interrupts to get mouse information or inputs. Whereas event mode is the one in which we install our own mouse handler to get mouse information. Request mode can be used for ordinary programming. In request mode, there is a chance for missing few inputs—mouse move or mouse click. But in Event mode, we can get all inputs. So professional programmers use Event mode.

Exercise

1. Write all mouse functions using interrupt programming. Then find out each function's use. (Hint: Get into graphics mode for better results)

Suggested Projects

1. Write a mouse driver program.
2. Write mouse functions that doesn't use interrupts or mouse.com. (Hint: You have to use ports)