UEFI Development Exploration 30 – A First Look at the Mouse

原创 luobing4365 Posted on 2019-09-22 12:37:22 Read 1.3k Collection 3 Likes

Category columns: UEFI Development Article Tags: UEFI Programming UEFI Mouse Low-level programming EFI_SIMPLE_POINTER UEFI Image Programming

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

There has always been a main thread in my series of blogs on UEFI development and exploration, which is to develop the UEFI Option ROM on the test board.

Recall that so far, **bus and device access** : PCI, Smbus, serial port, have all been (the serial port implementation is not perfect, but it is not needed on the test board); **interface related:** screen drawing, text display, image (BMP, PCX, JPEG) display, have also been completed; **user interaction:** the keyboard has been completed, and the mouse has not been written yet.

In other words, after the mouse exploration is completed, only the Option ROM architecture remains to be built, and this main line is completed.

The UEFI development exploration series has entered the mid-term.

After completing the main storyline, I will delve into various UEFI topics, such as multithreading, video, package organization, etc. Thanks to the debugging environment built in the previous chapters, many topics that were previously impossible to study are finally possible to analyze.

Back to programming the mouse.

1 UEFI support for mouse

BIOS support for mouse has always been in an awkward situation. Because BIOS settings have always been difficult to understand, it is difficult for non-technical personnel to figure out the terms. As a functional configuration, the ease of user interaction has always been at a low level.

In a previous blog post, I talked about the process of writing a mouse driver under Legacy BIOS. In fact, many motherboards eventually did not support mouse interrupts, which meant that the driver I wrote could not run.

With the development of UEFI, this situation has gradually changed. The BIOS setup interface has become more and more beautiful, and in many cases it is almost the same as operating software under the operating system . In the UEFI Spec, the protocol that supports the mouse is the Simple Pointer Protocol. As shown below:

EFI_SIMPLE_POINTER_PROTOCOL

Summary

Provides services that allow information about a pointer device to be retrieved.

GUID

```
#define EFI_SIMPLE_POINTER_PROTOCOL_GUID \
 {0x31878c87,0xb75,0x11d5,\
  {0x9a,0x4f,0x00,0x90,0x27,0x3f,0xc1,0x4d}}
```

Protocol Interface Structure

```
typedef struct _EFI_SIMPLE_POINTER_PROTOCOL {
 EFI_SIMPLE_POINTER_RESET      Reset;
 EFI_SIMPLE_POINTER_GET_STATE  GetState;
 EFI_EVENT                     WaitForInput;
 EFI_SIMPLE_INPUT_MODE         *Mode;
} EFI_SIMPLE_POINTER_PROTOCOL;
```

Parameters

Reset	Resets the pointer device. See the Reset() function description.
GetState	Retrieves the current state of the pointer device. See the GetState() function description.
WaitForInput	Event to use with EFI_BOOT_SERVICES.WaitForEvent() to wait for input from the pointer device.
Mode	Pointer to EFI_SIMPLE_POINTER_MODE data. The type **EFI_SIMPLE_POINTER_MODE** is defined in "Related Definitions" below.

Related Definitions

```
//*******************************************************
// EFI_SIMPLE_POINTER_MODE
//*******************************************************
typedef struct {
 UINT64       ResolutionX;
 UINT64       ResolutionY;
 UINT64       ResolutionZ;
 BOOLEAN      LeftButton;
 BOOLEAN      RightButton;
} EFI_SIMPLE_POINTER_MODE;
```


Figure 1 Protocol supporting mouse (UEFI Spec 2.8 page 454~455)

Mouse information can be obtained through Mode, but in actual operation, I found that it is not very useful. At present, there are still many UEFI BIOS that do not support this protocol well. Mode shows support, but fails to obtain mouse information.

The operation method of the mouse is similar to that of the keyboard. You can use Event to get whether there is a mouse message. I won't discuss it here. Just look at the code provided in this article and you will understand it at a glance.

The functions are not explained one by one, the documentation is written more clearly.

2 Myths

I used a microcontroller to simulate a mouse for a while, and finally implemented a USB mouse using a development board. It was quite interesting to simulate the movement of the mouse and the operation of the left and right buttons through the buttons on the development board.

The program must provide the report descriptor of the mouse. Among them, the logical displacement of the mouse generally has a maximum and minimum value, which are 127 and -127 respectively. As shown in the figure, it is taken from the USB HID specification:

E.10 Report Descriptor (Mouse)

Item		Value (Hex)
Usage Page (Generic Desktop),		05 01
Usage (Mouse),		09 02
Collection (Application),		A1 01
Usage (Pointer),		09 01
Collection (Physical),		A1 00
Usage Page (Buttons),		05 09
Usage Minimum (01),		19 01
Usage Maximun (03),		29 03
Logical Minimum (0),		15 00
Logical Maximum (1),		25 01
Report Count (3),		95 03
Report Size (1),		75 01
Input (Data, Variable, Absolute),	;3 button bits	81 02
Report Count (1),		95 01
Report Size (5),		75 05
Input (Constant),	;5 bit padding	81 01
Usage Page (Generic Desktop),		05 01
Usage (X),		09 30
Usage (Y),		09 31
Logical Minimum (-127),		15 81
Logical Maximum (127),		25 7F
Report Size (8),		75 08
Report Count (2),		95 02
Input (Data, Variable, Relative),	;2 position bytes (X & Y)	81 06
End Collection,		C0
End Collection		C0

Figure 2 HID spec - Mouse report descriptor example

So, in my impression, the maximum logical displacement of the mouse is 127.

However, during the test, it was found that on some platforms (my Xiaomi 15.6, graphics card Geforce MX110), using the GetState function provided by the above Protocol, the relative displacements RelativeMovementX and RelativeMovmentY obtained were much larger than the above values (see Figure 3).

This caused a blind spot in my cognition. Isn't the relative displacement obtained by GetState the same concept as the logical displacement described in the HID protocol? What is the relationship between them?

I don't have the energy to explore this matter today, so I'll leave it here and fill it in when I have time.

3. Compile and run

The mouse cannot be used normally in the TianoCore simulation environment or when using qemu to start Ovmf. Therefore, you can only compile the program to 64-bit, start the UEFI Shell with a USB flash drive, and test it on the actual platform.

I tested three platforms: SenyPC's HM86 platform, Intel NUC6CAY, and the Xiaomi 15.6 (Mx110 graphics card) I use for work.

The first platform was released around 2014, and it does not support the mouse protocol well. It can find the protocol, but the program cannot capture the mouse information. The second and third platforms work fine, as shown in the screenshots below:

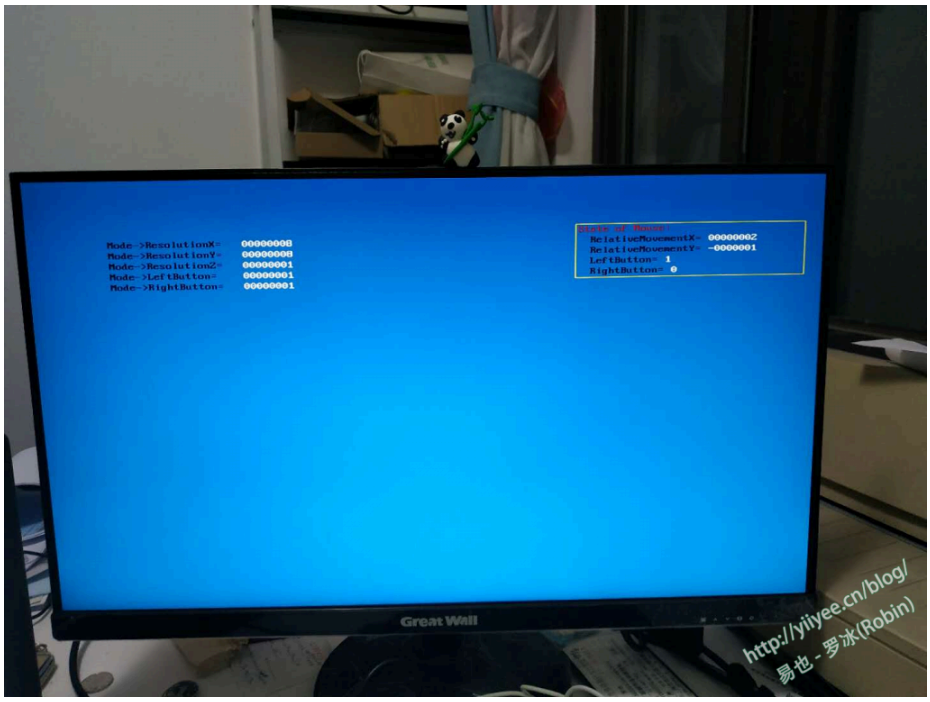

Figure 3 Testing the program on NUC6CAY

Press the left mouse button and you will find that LeftButton becomes true. Other functions including right-clicking and moving the mouse are normal. However, on the Xiaomi platform, the situation described in the previous section, where the relative displacement is very large, occurs.

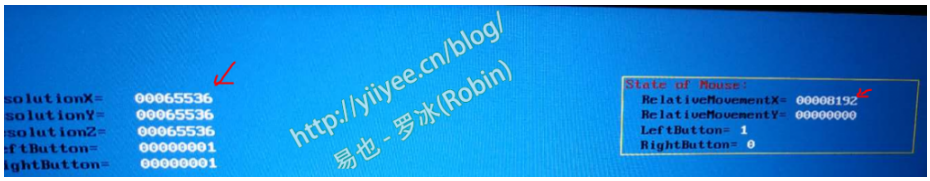

Figure 4 Test program on Xiaomi notebook

Except for the displacement problem, the mouse messages can be obtained well. In other words, the obstacle of supporting the mouse on the GUI interface no longer exists. In the next article, I will try to display the mouse on the graphical interface.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located at: /20 Mouse-GetState