UEFI Development Exploration 31 – Mouse GUI Construction

原创 luobing4365 Posted on 2019-09-22 12:41:04 Read 1.2k Collection 3 Likes copyright

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

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

After conducting various experiments described in the previous blog, the relevant code has basically been built, and we can start to support the mouse of the GUI interface.

UEFI provides interrupt support for developers, and all asynchronous operations can be completed through Event. Simple Pointer Procotol also provides support for this mechanism. According to the function description in the document, build the following function:

Luo2.c | Common.c | Mouse.c | Mouse.h | tools_def.txt

```
1  //code by luobing 2013-5-17 13:11:15
2  #ifndef _MOUSE_H
3  #define _MOUSE_H
4  #include "Common.h"
5
6  EFI_STATUS DisplayMouseMode(void);
7  VOID putMouseArrow(UINTN x,UINTN y);
8  VOID initMouseArrow(VOID);
9  EFI_STATUS GetMouseState(EFI_SIMPLE_POINTER_STATE *State);
10 EFI_STATUS CheckMouseEvent(VOID);      //luobing 2013-5-21 7:58:35
11 #endif
```


Figure 1 Mouse GUI function

The four functions of Line8-Line10 are the main working functions: initialize the mouse image initMouseArrow; check whether there is a mouse event to trigger CheckMouseEvent; get the mouse message GetMouseState; update the mouse state putMouseArrow.

Compared with the mouse driver built under the Legacy BIOS before, this one is much simpler. I can't help but want to comfort myself who was struggling with debugging back then.

1. Program Construction

Some relatively old BIOS do not support the mouse very well, but they can still locate the Simple Pointer Protocol when locating it. This situation has no impact on the code logic, but it just cannot support the mouse on the GUI interface.

Another question is the problem pit dug in the previous article: What is the difference and connection between RelativeMovementX, RelativeMovementY and the logical displacement in the USB HID protocol. My personal guess is that there is no direct connection. These are different expressions of mouse displacement under two different architecture systems.

Under UEFI, the movement of the mouse position should be considered in combination with Mode->ResolutionX and Mode->ResolutionY. If the unit is pixel, the mouse displacement formula should be RelativeMovmentX * MouseRegulator/ResolutionX (the calculation formula for the Y axis is similar). Among them, MouseRegulator is a user-defined adjustment factor used to adjust the sensitivity of mouse movement.

For example, in my program, I set the scaling factor to 8.

```
i = 600, j = 300;
putMouseArrow(i, j);
while (!exit_flag) //左键和右键一起按下，退出
{
  if (!EFI_ERROR(CheckMouseEvent())) //发生了鼠标事件
  {
    GetMouseState(&State);
    exit_flag = (State.LeftButton & State.RightButton);
    //调节因子设置为8
    i += ((State.RelativeMovementX<<3) >> xScale);
    if (i < 0)
      i = 0;
    if (i > SY_SCREEN_WIDTH - 1)
      i = SY_SCREEN_WIDTH - 1;
    j += ((State.RelativeMovementY<<3) >> yScale);
    if (j < 0)
      j = 0;
    if (j > SY_SCREEN_HEIGHT - 1)
      j = SY_SCREEN_HEIGHT - 1;
    putMouseArrow(i, j);
  }
}
```


Figure 2 Mouse display in graphics mode

The program logic is very simple: when the user presses the left and right buttons of the mouse at the same time, exit the while loop. In the loop body, if a mouse event occurs, obtain the mouse information, calculate the displacement value, update the mouse position, and update the mouse image at the same time.

Maybe because I have written a lot of assembly programs, I am more obsessed with the running efficiency of the program. When calculating the displacement, the calculation process is also simplified, mainly through shift calculation. Among them, the right shift xScale and yScale represent the division calculation of ResolutionX and ResolutionY respectively.

During the test, we found that ResolutionX is usually 2^n, so we calculated its power n. With this method, we don't have to worry about annoying type conversions, and the code is much simpler.

2. Program running

As in the previous article, the mouse program cannot run in the simulation environment. Compiled into X64 UEFI app, the running effect on the actual machine is as follows (intel NUC6CAY): (The video is too large to be converted to a GIF image below 2M, I put the video on gitee - 20190703 mouse program demonstration.mp4)

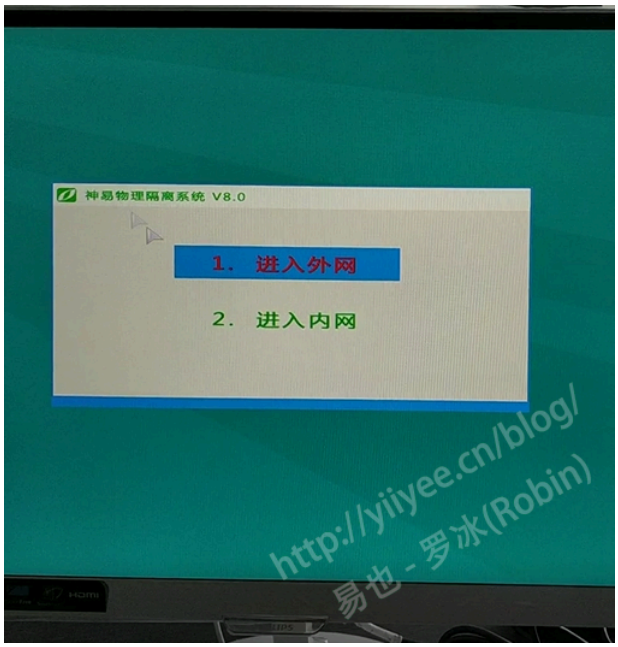

Figure 3 Running the mouse program on NUC6CAY

I tried it on several platforms I could find, especially machines after 2016, and they all supported it very well. So far, the mouse support work is complete.

In order to facilitate demonstration in the simulation environment, a piece of code is added to control the movement of the mouse icon with the keyboard arrow keys. It also works well when it coexists with the mouse control code. The following is the situation when it runs in the simulation environment (the mouse is not supported at this time, and it can only be controlled by the keyboard arrow keys)

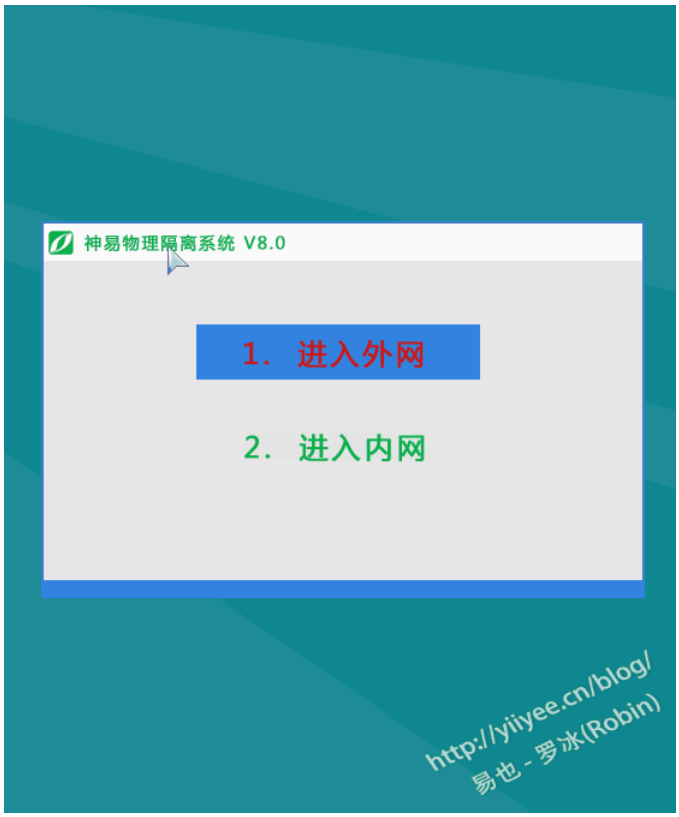

Figure 4 Keyboard control mouse icon

3 One More Thing

In actual commercial products, "appearance" is very important. In order to arouse users' desire to buy, in addition to functional requirements, the appearance of the product accounts for a much larger proportion than we think.

I was particularly obsessed with image effects for a while, and tried to make the underlying interface of the isolation card transparent, and made some designs based on Apple's UI. The leader at the time laughed at me for not doing my job properly, and the program I finished was not used in the product in the end.

In the development of my own open source project Foxdisk, I finally fulfilled my long-cherished wish and implemented some of my favorite special effects, such as image fading in and out, icon transparency, automatic font shadow, lightening and darkening, etc.

Human nature is ultimately uncontrollable.

In fact, the code implementation in this blog makes the mouse icon transparent, as can be seen in the Gif image. In fact, the principle is relatively simple, and after combination, many interesting effects can be achieved.

I have implemented some special effects in the UEFI code, which I will talk about in the next article.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located at: / 21 Mouse-GUI