UEFI Development Exploration 20 – Serial Port Communication

原创 luobing4365 Posted on 2019-09-17 22:48:12 Read 1.8k Collection 1 Likes 1 copyright

Category columns: UEFI Development Article Tags: UEFI Programming UEFI Serial Port Low-level serial port Named pipe communication Low-level programming

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

The serial port may be the most viable interface standard. It has been playing a huge role since 1980. Especially in industrial environments, it can be said to be the king standard.

I developed a network-to-serial converter a few years ago, and I was surprised to find that serial ports are also used in many financial contexts.

Let's get back to the serial port communication under UEFI.

1 Shell command to view the serial port

My development environment is still virtual machine winxp + UDK2010, and I have not had time to build UDK2018. There is a command Sermode in UEFI Shell, which can view and set serial port parameters.

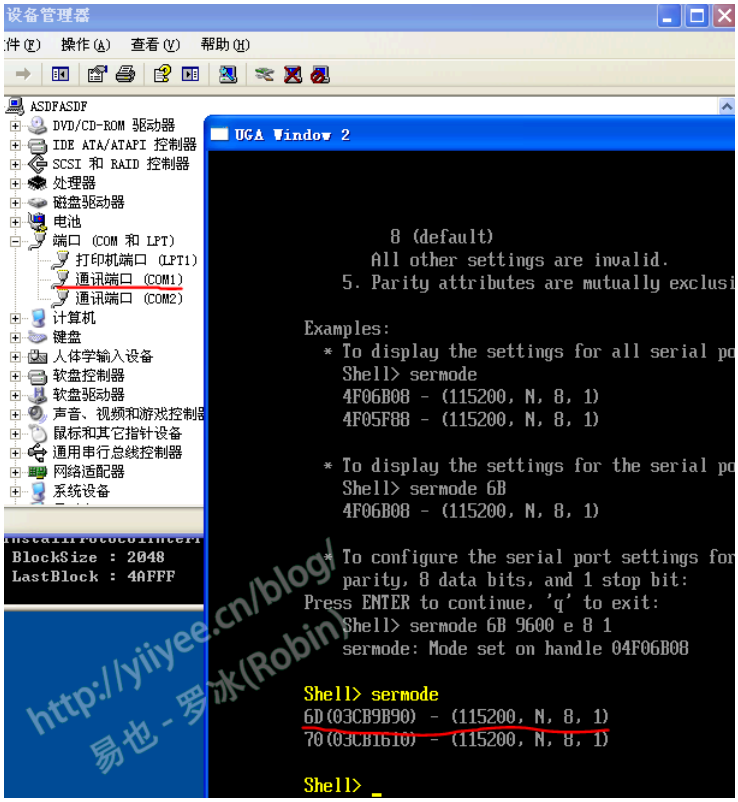

Figure 1 Enumerating serial ports in TianoCore simulation environment

There are two serial ports in the device manager, and two serial ports are also listed in the TianoCore simulation environment. The next question is, how to communicate with the serial ports in the virtual machine?

2 Communicate with the serial port in the virtual machine

I have been learning to use windbg for debugging driver code. It can be used with a virtual machine to debug the execution of the OS, or to debug kernel-mode source code .

Windbg uses named pipes to communicate with virtual machines, which gave me some inspiration. I developed a small tool to read and write named pipes. The commonly used pipe name of the virtual machine is \\.\pipe\com_1, as shown in the figure below.

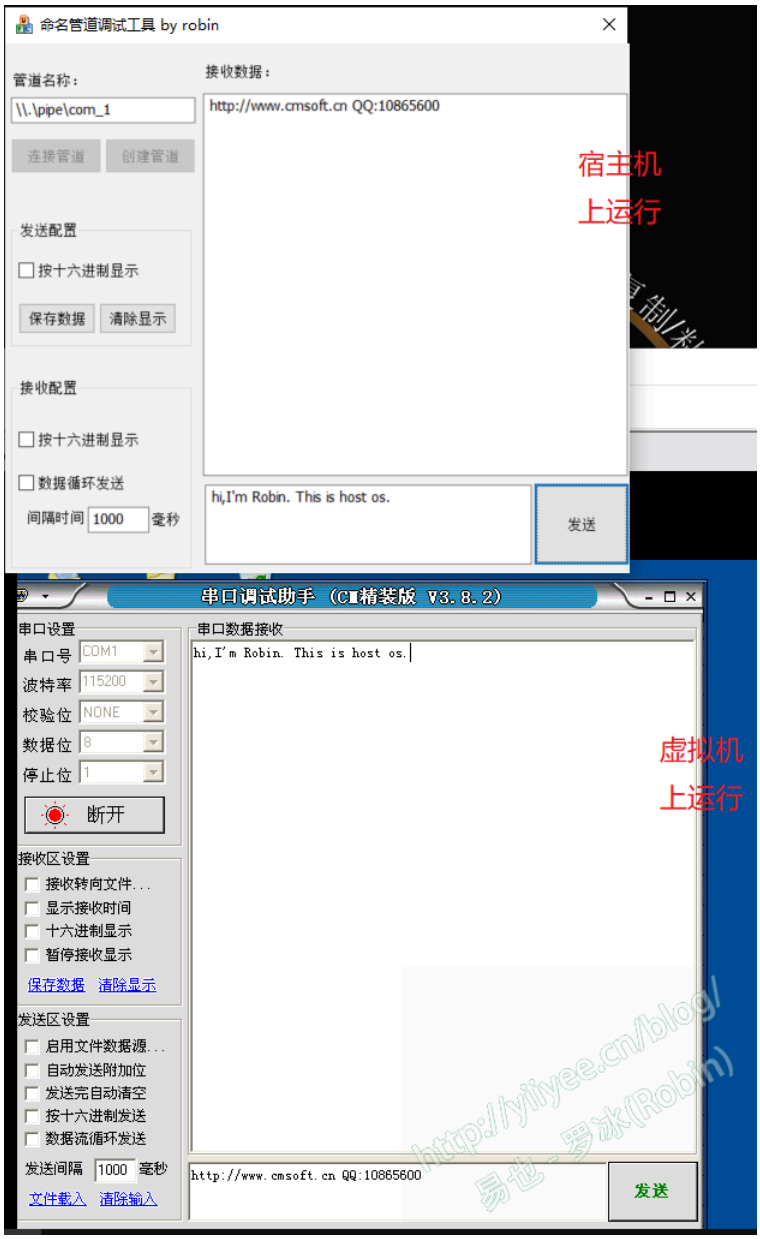

Figure 2 Serial port communication between the host and the virtual machine

This will open up the serial communication channel between the virtual machine and the host machine. Of course, you can also use the serial port of the two computers to communicate directly, but now there are relatively few machines with serial ports and it is difficult to find them.

3 Serial port functions under UEFI

The serial port protocol is relatively simple, and there are a lot of information on the Internet. There is no need to explain its principles one by one. I will mainly focus on the functions of operating the serial port under UEFI.

UEFI Spec 12.8 Serial I/O protocol describes the relevant serial port functions in detail.

Protocol Interface Structure

```
typedef struct {
 UINT32                          Revision;
 EFI_SERIAL_RESET                Reset;
 EFI_SERIAL_SET_ATTRIBUTES       SetAttributes;
 EFI_SERIAL_SET_CONTROL_BITS     SetControl;
 EFI_SERIAL_GET_CONTROL_BITS     GetControl;
 EFI_SERIAL_WRITE                Write;
 EFI_SERIAL_READ                 Read;
 SERIAL_IO_MODE                  *Mode;
 CONST EFI_GUID                  *DeviceTypeGuid; // Revision 1.1
} EFI_SERIAL_IO_PROTOCOL;
```

Parameters

Revision	The revision to which the **EFI_SERIAL_IO_PROTOCOL** adheres. All future revisions must be backwards compatible. If a future version is not back wards compatible, it is not the same GUID.
Reset	Resets the hardware device.
SetAttributes	Sets communication parameters for a serial device. These include the baud rate, receive FIFO depth, transmit/receive time out, parity, data bits, and stop bit attributes.
SetControl	Sets the control bits on a serial device. These include Request to Send and Data Terminal Ready.
GetControl	Reads the status of the control bits on a serial device. These include Clear to Send, Data Set Ready, Ring Indicator, and Carrier Detect.
Write	Sends a buffer of characters to a serial device.
Read	Receives a buffer of characters from a serial device.
Mode	Pointer to **SERIAL_IO_MODE** data. Type **SERIAL_IO_MODE** is defined in "Related Definitions" below.
DeviceTypeGuid	Pointer to a GUID identifying the device connected to the serial port. This field is NULL when the protocol is installed by the serial port driver and may be populated by a platform driver for a serial port with a known device attached. The field will remain NULL if there is no platform serial device identification information available.

Figure 3 Serial IO Protocol (UEFI Spec 2.8 page 466)

According to the Spec, I wrote the read and write functions, as well as the test functions:

```
EFI_STATUS DisplayAllSerialMode(void);
EFI_STATUS DisplaySerialMode(void);
EFI_STATUS SendDataToSerial(UINTN length,VOID *buffer);
EFI_STATUS GetDataFromSerial(UINTN *length,VOID *buffer);
EFI_STATUS SetSerialPortAttrib(UINT64 BaudRate, EFI_PARITY_TYPE Parity,UINT8 DataBits, EFI_STOP_BITS_TYPE StopBits);
```

The first function prints out all the serial ports on the system, similar to the enumeration function of Sermode. The remaining functions only use the first serial port found to read and write to it.

This operation method is a bit strange to me. In the past, when writing serial port programs, whether it was a Windows app or a single-chip microcomputer, I used to open the serial port device first. However, under UEFI, when enumerating Handles, each Handle corresponds to a serial port device.

It is easy to send data to the serial port, just use the Write function. I can't think of any way to read the serial port, UEFI does not provide interrupts, and I can't find what events will be triggered after the serial port data comes.

During the test, the data sent from the Host OS is occasionally printed out. There is still a problem with the code.

4 Compile and run

The results of the operation are shown in the figure.

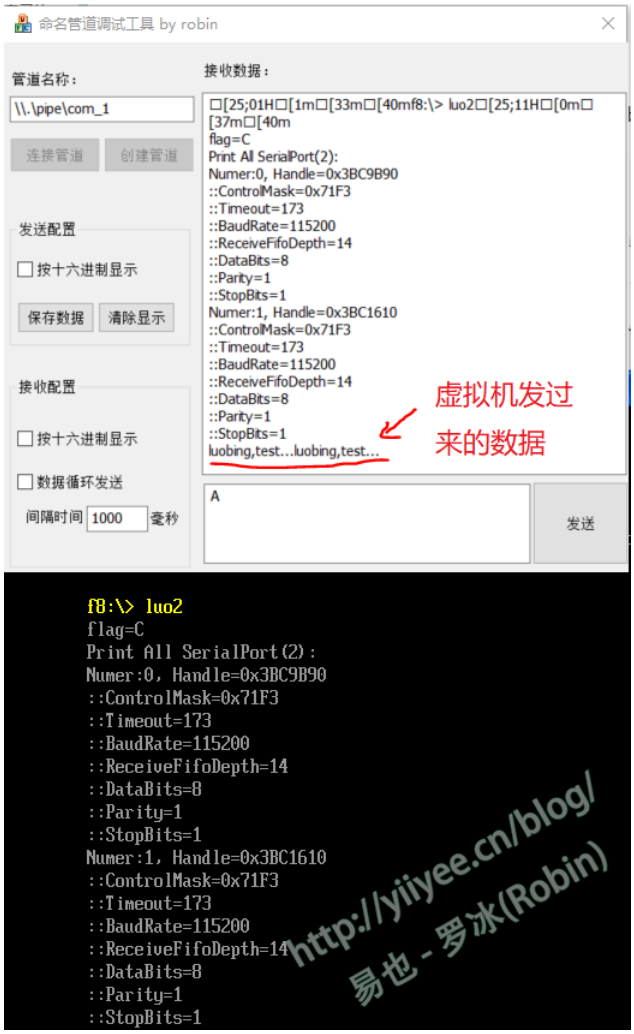

Figure 4 Program running results

The program tests three functions:

1. Print all current serial port device parameters, which are obviously consistent with the Sermode information;
2. Send a test string to the serial port. The string is captured by the debugging tool written;
3. Read the information sent from the serial port. Failed!

The purpose of writing the serial port is to have a printing debugging method to facilitate debugging the program in graphics mode. From this perspective, the current program code has met the requirements.

However, it is always unpleasant to not find a good mechanism to read serial port data.

I hope someone who knows this can give me some pointers^_^

5 One more thing

While writing this program, I discovered a phenomenon: all the data displayed on the screen in the TianoCore simulation environment will be transmitted to the named pipe debugging tool I wrote.

The data sent through the named pipe will also be input to the Shell. The debugging tool I wrote has the Enter key overloaded in the send edit box, so the Enter key can also be sent. Take Sermode as an example:

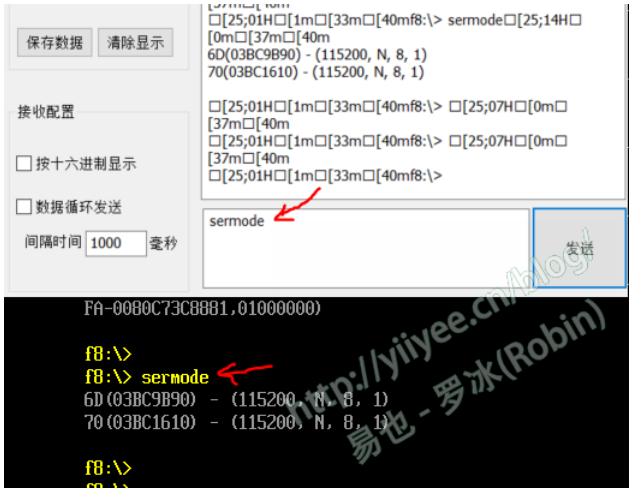

Figure 5 Sending shell commands using the named pipe debugging tool

Write sermode (Enter) through the debugging tool, click the "Send" button, and Shell will directly execute this command.

Another thing that interests me is that the data sent through the named pipe debugging tool can be well received by the TianoCore simulation environment. It should be received through the serial port, which has been proved by the above experiment.

In other words, there is a good mechanism under UEFI to receive serial port data, but I just don't know it.

I think this debugging tool can be transformed into a convenient UEFI development tool. So I put the source code here, in the same folder as the code of this blog. This tool is developed using VS2015 and under Win10.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
The project code is located under: / 13 SerialPort-PipeTool.
/luo2: UEFI serial port debugging code;
/PipeTool.exe: The executable file of the named pipe debugging tool, a 32-bit program, compatible with winxp-win10 series systems;
/pipetool: PipeTool source code, compiled with VS2015.