The multi-serial port driving research of video monitoring system based on the Internet of things

gaojie¹ xujun²

Modern Economics & Management College of Jiangxi University of Finance and Economics
Nanchang, China

Abstract: considering the design of multi-serial port drive of the Video monitoring of the Internet of things system is similar to the design of common character device driver. Serial driver is an essential character of device driver. This paper focuses on the technology of multi-serial port programming which based on the core design of tty.

class number: TP316 document code: A

1 Introduction

The discussion of video monitoring has a long history. The traditional application is in the security field, which is an important means to assist the public security departments and related management units to fight crime and maintain social stability. If the camera as the eyes, intelligent video systems or equipment can be as a human brain, video monitoring is the perception of the Internet link without the "eyes", its main role is to faithfully show video information and records, on the far side is an extension of people visual sense. The intelligent video technology can establish a mapping relationship between image and image description, so that the calculation can understand the contents of video through digital image processing and analysis. Video monitoring refers to the smart video technology, which mainly refers to: "automatic analysis and extraction of key information from video source." With the gradual popularization of the Internet of things, and people to the strong requirement of the intelligent, intelligent video surveillance usher in huge opportunities for development, the Internet of things will have a profound effect on the video monitor, including its concept, technology, form and effect. The intelligence of video monitoring provides a broad prospect for the application of computer vision in the field of public security, which is an inevitable requirement for the application of Internet of things. The hierarchy of tty in the Linux kernel includes tty core, tty line discipline, tty driver. Tty core is responsible for communication with the Linux kernel, tty line discipline tty line discipline work took the form of special formatting data received from a user or hardware, tty core from a user access to data will be sent to a tty device, tty core pass the data to tty line discipline drive, and then the data is passed to the tty driver, tty driver converts data can be sent to the format of the hardware. The process of receiving data in serial port is: from the data received from tty hardware to the tty driver, enter the tty circuit discipline drive, then enter the tty core, where it is obtained by a user. Although most of the time the data transfer between the tty core and tty will go through the transformation of the tty line discipline, the data will also be transmitted directly between the tty driver and the tty core.

2 Theorizing

Serial port driver is essentially a character device driver, but the kernel according to its special encapsulate data transmission process, the file operations (file_operation) packaging to tty core (tty_io. C), provided by the tty core interface to the application calls, tty core also provides interfaces to line discipline and drivers use; Therefore, when writing a serial driver, you only need to invoke the interface provided by the tty core and pass in the driver’s handler (tty_operations).

When driving to the tty core, the tty core will assign the device number to the driver and register the file operation function (file_operations). And each file operations function will be called in the tty core drive incoming processing function, can achieve the function of application requirements or driver, tty core is to realize a transfer process, but the tty core encapsulates many serial ports driver implementation of general function, greatly simplify the writing of a serial port driver.

The main points of serial port driver are divided into the following steps:
2.1 Registration, release device drivers

Int tty_register_driver(struct tty_driver *driver); Be right before you sign up tty_driver for the assignment. Specify device number, serial number, operation function and other information in the tty_driver.

Int tty_unregister_driver(struct tty_driver *driver); Release device driver.

2.2 Serial port operation function

Set the serial port operation function provided by the driver, which is provided to the tty core at the time of registration, and the serial port operation function is set in the tty_driver.

2.3 Interrupt handling function

The interrupt handling function is the core of serial communication, which is realized by interruption. The serial port has two 64B FIFO (TX and RX). Each is used to send and receive data.

A serial port to send data process as shown in figure 1.3.1, generally are based on asynchronous serial port communication protocol, so data in bytes transmission, when the driver will be a byte data is written to a serial port, will first through THR deposited in the RX FIFO registers, and then through the (TSR) will be sent data, every data delivery time is 16 clock cycles. When the data in the RX FIFO is lower than the TRIGGER value or null, the interrupt is triggered and the bit[0] of the LSR register and the bit[1] of ISR are set. The interrupt handler sends data to the serial port based on these values. The TRIGGER value of RX FIFO is set by the bit[6,7] of the FCR register.

The data in the RX FIFO is sent to the serial port, and the send data process is shown in figure 1.3.2. The serial port receiving data process is also in bytes, the serial port by RSR register receiving data, then data will be stored in TX FIFO, the driver also in bytes data from TX FIFO. When the data in TX FIFO is greater than a TRIGGER value, the interrupt is triggered, and the bit[5] of ISR is set to 1, and the interrupt handler reads the data from the serial port TX FIFO based on these values. The TRIGGER value of TX FIFO is set by the bit[4,5] of the FCR register.

When the automatic RTS hardware flow control is enabled (EFR bit[6]=1 MCR bit[1]=1), when the data of the TX FIFO is greater than 56, the serial port will automatically notify the sender to stop sending the data; When the TX FIFO data is less than 8, the serial port will automatically notify the sender to continue sending the data. When using RTS/CTS (request to send/clear) flow control, the RTS and CTS lines at both ends of the communication should be connected.

3 Design ideas

The multi-serial port driver is based on the tty core, mainly designing the following aspects:

3.1 With kernel communication

Here with the kernel of the communication is actually done by tty core, through registered drivers should tell the kernel with the tty core processing which drive and should use which handler, register functions as follows:

int tty_register_driver(struct tty_driver *driver);

3.2 Communicate with the application

Character device drivers and application communication are implemented through file operations, file_operations, and file operations are implemented in the tty core. Visible, although the application is a function call tty core file operations, but in essence or drive to achieve, therefore, drivers can be according to the need to implement and application in the communication function, I realized the open, in more than a serial port driver release, write, and the ioctl function.

3.3 Hardware communication

The driver and hardware communication is the operation register, which has detailed instructions in the datasheet of serial port chip. The multi-serial port chip is ST16C654, which introduces the commonly used registers in several drivers.

- RHR: Receive the data register;
- THR: Send the data register;
- IER: Interrupts enable the energy register, where bit[0-3] corresponds to the interrupt in 4;
- ISR: Interrupt status register, where bit[1,2] represents read and write interrupt;
- FCR: FIFO control register, which allows you to set the FIFO and TRIGGER levels here;
- LCR: Line control register, set line procedures, including: data bits, check bits, stop bits, etc;
- MCR: The MODEM controls the register, where bit[0,1] is used to control the output of the hardware flow;
- LSR: line status register, where bit[0] marks RHR for data.
Bit [5] to mark whether THR is empty;
DLL、DLM:Baud rate divisor register to set baud rate.

4 Data structure design

In this case, the data structure is not in the previous section of the data structure, here refers to a serial port driving internal data organization methods, these data structures throughout the serial driver, reasonable way of designing the data structure and organization can effectively improve the efficiency of the driver, the following respectively introduce the kernel and multiple serial port driver data structure design and organization, and comparison and analysis.

The data structure and organization design should pay attention to the following aspects:

a. The data structure can represent all the information of the serial port and the properties of the serial port in the process;
b. At open, you should be able to determine which serial port is based on the data structure, because there are many serial ports on a board;
c. The interrupt handler also needs to be able to use the data structure to determine which serial port is interrupted.

4.1 Kernel data structure design

The kernel serial driver is implemented mainly through the following two arrays.

struct serial_state rs_table[64];

This is a serial port information structure, basic structure, including baud rate, base address, interrupt number and other information, the definition has been on the initialization, initialization is done by macro SERIAL_PORT_DFNS, this macro to initialize the base address of each serial port and interrupt number, the initial value in the array is the ttySn values by default, so you can according to the main board of the serial port information changes the value of the the macro implementation drive each serial port on the main board.

struct async_struct *IRQ_ports[225];

This is an array pointer, with an array size of 225, which corresponds to each interrupt number, and each element of the array points to the serial information structure linked list that shares the interrupt number.Async_struct is a structure containing information opening a serial port all the body, when opening a serial port, will dynamically create a structure async_struct serial port information, then according to the basic information of the open the equipment serial_state initialized to this structure, finally the chain into the corresponding IRQ the head of the list.

When open the device, device number and driven by opening the device device number to calculate the starting time is which a serial port, to get basic information serial_state serial port;In the interrupt handler, you can obtain all the serial port information using the interrupt number directly, and then process all the serial ports.

This is an array, similar to the 2.4 kernel IRQ_ports, which contains the structure of a list_head type variable to point to the chain header of all the serial ports of the same interrupt number.

4.2 Multi-data structure design

Kernel data structure of the serial port driver design is a kind of general design, the purpose is that can be applied to different kinds of main board, and the data structure of multiple serial port driver is completely on the mainboard design of series of 104 serial port, is according to the characteristics of multiple design structure.

As each more motherboard all serial port are some common properties, so to design a motherboard information structure, this structure each motherboard, motherboard each serial port information can be set by this structure, the structure is defined as follows:

struct evocdrv_brdconf /* Information on each board */
int board_type;/* Board type 1-1048COM 2-COMMON*/

Besides motherboard information structure, also defines a size 32 serial port information structure array, used to hold information of each serial port, serial port information during initialization arrays based on its main board information set;Due to the particularity of the mainboard, such as: all the serial port on the motherboard is Shared interrupt number, serial port base address is continuous and so on, so you can think of these common property is set in the motherboard information structure.The serial information array is designed in this way, by default of 8 serial ports of each motherboard. If the number of serial port of the main board is less than 8, the corresponding digit of the array is null.

Opening a serial port, according to the device number and driven equipment device number starting value, calculate how many serial ports, open the can be directly from the serial port information according to the array serial port information;In the interrupt handler, there is no pointer array as the kernel of the serial port driver interrupt, but due to a serial port interrupt number on every board is the same, so only need to be interrupted in the event of a judgment which is produced by the serial port on the board, and then to judge one by one the serial port on the plate and process;Number in the request is called, will board the first structure as a parameter to a serial port information, then according to the serial port in the interrupt handler,8,16,24 a 0 elements of an array of information to determine which belongs to the motherboards.

5 Conclusion

It can be seen that the design of data structures in the kernel is a universal design, regardless of which type of motherboard and which type of chip can use the kernel driver.But it is because of the commonality, make the default value of the kernel of the serial port information is hard to do a proper Settings, because different motherboard serial port information tend to be different,
only through serserial command to modify manually. Multi-serial port driver design is the kernel of two structure combined by a serial port information, and achieve a motherboard information structure, the structure of all serial port on a board that contains the Shared information, for the design of the interrupt processing was introduced into the first serial every board through the information structure, there is no kernel is convenient, remains to be improved.

Acknowledgments

Jiangxi Provincial Education Office Science and technology project (GJJ171481) “Research on key technology of campus video monitoring system based on Internet of things”

About the author: gaojie (1983.8, Han nationality, lecturer, Network engineer, master), Engaged in teaching and development. The main research area is the embedded direction, which is to cut and customize the x86 series and arm series mainboard in the Linux OS system and drive the development.

References

1. Concertation and Achievements Report for the Transport Sector of the Telematics Applications Programme. Andrew Ryan, Fraser Sommerville. Proceedings for the 4th World Congress on Intelligent Transport System . 2007
2. MC35i Hardware Interface Description V01.03. . 2004
3. SJA1000 datasheet. . 2000
4. NMEA 01 83 ASCII interface specification. The National Marine Electronics Association. .
5. TIM—Lx GPS Module Datasheet. Thomas Christen. www. u-blox. com . 2004
6. GPS-based Message Broadcast for Adaptive Communications. M. S1 In et al. IEEE VTC Fall . 2002
7. “A simple, wireless powering scheme for MEMS devices.”. Jie Wu, Victor Quinn, Gary H Bernstein. Proceedings of SPIE the International Society for Optical Engineering . 2001
8. Precise point positioning for the efficient and robust analysis of GPS data from large networks. J. F. Zumberge, M. B. Heflin, D. C. Jefferson, M. M. Watkins, F. H. Webb. Journal of Geophysical Research . 2007
9. JP13 GPS Receiver Hardware Description Preliminary. www. Falcom. com .
10. The Point-to-Point Protocol (PPP). Simpson W. RFC 1661 . 1994