Development Of Embedded Ethernet Drivers For Arm9

[1,]T.Satyanarayna [2,] S.Latha(Associate Proffesor)

[1,2,]Dept of Electronics & Communication Engineering, Aurora Technological & Research Institute,
Jawaharlal Nehru Technological University.

Abstract

with the widely application of ARM technique, building the embedded operating system based on ARM processor has been a hot point of research. In this paper, the design of network device driver in Linux operating system based on ARM920T processor is implemented on the S3C2410- S development platform made in Beijing universal pioneering technology. Focused on discussing implementation principle of embedded Linux network drivers and detailed to analysis of the frame structure of the program code.

Keywords-ARM9 processor; embedded linux; network device driver; CS8900A

1. Introduction

Now the people more and more like open-source software. As a powerful and stable open-source operating system, Linux is acclaimed by the thousands of computer expert and amateur. In the embedded field, Linux can be cured in dozens of megabytes of memory chips or SCM after small cutting. So that it can be used in a specific context of embedded Linux. Strong network support functions of Linux achieved support for multiple protocols including TCP / IP, and it meets the demand for embedded systems application networking for the 21st century. Therefore, when developing and debugging embedded systems, network interface almost become indispensable module.

2. INTRODUCTION OF LINUX NETWORK DEVICE DRIVER

Linux network device driver is an important part of Linux network application. The whole Linux Network driver follows the common interface. For each network interface, it uses a device data structure. Generally, the network device is a physical device, such as Ethernet card.Network driver must solve two problems: first, not all network device driver based on Linux kernel have control equipment; second, Ethernet device in the system is always called / dev/eth0, dev/eth1 etc., regardless of the underlying device driver. When initialization routines of each network device are called, the driver will return a status, whichindicating whether it is orientation to an instance of the driven controller. If the driver does not find any device, then the entries pointed to the device lists by the 'dev_base' will be deleted. If the driver can find a device, then the rest of the device data structure is filled by this device information and the address of support function in network device driver.

3. Architecture Of Linux Network Device Driver

Shown in Fig. 1, the architecture of Linux network driver can be divided into four levels. Linux kernel source code provided the network device interface and the code above level. Therefore the main work which transplanting specific network hardware drivers is to complete the corresponding code of the device driver function layer. According to the specific bottom hardware features, Structure variable of network device interface 'struct net_device' type is defined and corresponding operation function and interrupt handling program are implemented.

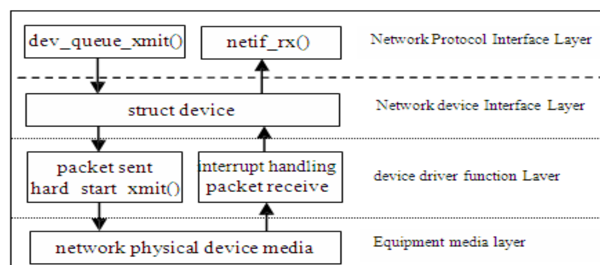

Figure 1.Architecture of Linux network driver

4. Ethernet Controller Chip Cs8900a Cs8900a

is a 16-bit Ethernet controller produced byCIRRUS LOGIC, embedded on-chip RAM, 10VASE-T transceiver filter and direct ISA bus interface. The salient feature of the chip is flexible to use, and it can dynamically adjust according to needs for its physical layer interface, data transfer mode and work mode, and it can adapt to different application environment through setting internal register.

CS8900A can be operated in memory mode and I / O

mode. When CS8900A is configured to Memory Mode operation, its internal registers and frame buffer are mappedto a serial 4KB host memory block, the host can directly access CS8900A's internal registers and frame buffer through the block.

5. Design And The Implementation Principle Of Network Driver

Linux network system can complete data transfer between all levels through the socket buffer sk_buff, data structure sk_buff is each protocol data processing object. sk_buff is the media of exchange data between driver and network. When the driver sends data to the network, data source and data length must be obtained. Data must be saved in sk_buff also after the driver received data from thenetwork. Thus upper layer protocol can process it.For the actual development of Ethernet driver, corresponding template program in the kernel source tree can be consulted, focused on understanding the implementation principle of network driver and program structural framework. Then the code is rewritten to develop specific hardware, and achieve the appropriate operation function. Through transplanting and preparing the embedded CS8900A network card driver on embedded development board (S3C2410 processor) to show the implementation principles of network driver.

A. Initialization Function

Initialization of network equipment is completed mainly by initialization function which is referred by init function pointer in device data structure. After the kernel load the network driver module, initialization process will be called. First it is necessary to detect whether network physical device exist, which is completed by detecting the physical device hardware characteristic. Then the resource equipment needed is configured, such as interrupts. Next the device'device' data structure is constructed. The relevant variable in the device is initialized by detected data, and finally thedevice is registered to the Linux kernel and applies memory space.In this instance, the initialization function is "_init cs8900a_s3c2410（void）".
In the network device driver, the device 'device' data structure is dev_cs89x0, which is defined as follows.
＃ifdef MODULE static struct net_device dev_cs89x0={
'''',
0, 0, 0, 0,0, 00, 0, 0, NULL,NULL};The function 'cs89x0_probe' detect the existence of the network physical device, but device initialization is completed by two functions 'cs89x0_probe' and'cs89x0_probe1' together. In 'cs89x0_probe1', the function pointers of 'device' data structure are filled.After completion of filling pointer, to register by'register_netdev' (struct net_device * dev) function. Two function

register_netdev' and 'unregister_netdev' are defined in file 'net_init.c '.Since there is 'init' function, there should also be'cleanup' function, because they are essential function of each driver. The 'cleanup' function run when module is unloaded, which complete mainly the work of resource release. Such as cancel device registration, free memory, release the port, etc. All in all, it is some action contrary to init.The function of Cancellation of network device registration is 'unregister_netdevice' defined in file / net / core / dev.c, this function is called in 'unregister_netdev'.

B. Open Function

When system response 'ifconfig' command, a network interface will be opened (closed). The 'ifconfig' command given addressto interface by callingioctl' (SIOCSIFADDR). Response of SIOCSIFADDR is accomplished by the kernel, and device-independent. Then, the 'ifconfig' command set IFF_UP bit of dev-> flag to open the device by calling 'ioctl' (SIOCSIFFLAGS). The device's open method is called through the above called. In the network device driver, Open function is called when network device is activated, that device status becomes from down to up. So a lot of initialization work can be done here. In open function, operation on the register uses two
functions: 'readreg' and 'writereg'. 'readreg' function is used to read the register contents, 'writereg' function is used to write registers, the code is as follows:

```
inline int readreg(struct net_ device *dev,int portno){
outw(portno,dev->base_addr+ADD_PORT);
return inw(dev->base_addr+DATA_PORT);
}inline void writereg(struct net_ device
*dev,int portno,int value){ outw(portno,dev->base_addr+ADD_PORT); outw(value,dev-
>base_addr+DATA_PORT);
}
```

C. Close Function

Close function (net_close) releases resources to reduce system burden. It is called when the device status becomes from up to down. In addition, if the driver is loaded as a module, the macro MOD_DEC_USE_COUNT need to call in close also, to reduce the frequency of equipment cited in order to uninstall the driver.

D. Send Function

Sending and receiving of data packets are two key processes achieving Linux network driver, good or bad of the two processes affected directly the driver's overall running quality. First of all, when the network device driver is loaded, device is initialized by init function pointer in the device domain calling network device initialization function. If the operation is successful, device is opened by open function pointer in the device domain calling network device open function. Next, hardware packet header information is set up by building hardware packet header function pointer hard_header in the device domain. Finally, the packet sent is completed through protocol interface layer function dev_queue_xmit calling function pointer hard_start_xmit in the device domain. If sent successfully, sk_buff is released in hard_start_xmit method, and return 0 (send success). If the device can not be process temporarily, such as the hardwareis busy, and then return 1. At this time if the dev-> tbusy is set to non-zero, the system regards that the hardware is busy, it will not re-send until dev-> tbusy is set to 0. tbusy's setting zero task is completed by interrupt generally. Hardware interrupt at the end of sent, at this time tbusy can be set to 0, then call mark_bh () to notify system to re-sent. In the case of sending unsuccessful, dev-> tbusy can not be set to non- zero, this system will try to resent continuously. If hard_start_xmit is not sent successful, then sk_buff cannot be release. The data which is sent in Sk_buff already contains the frame head of hardware requirement. Hardware frame head need not be fill in send method, data can be sent directly to the hardware. sk_buff is locked, and it is assured that other programs will not access it.

E. Receive Function

Receive function is different from receive data, network interface does not provide receive function pointer similar to net_receive_packet, because that network device receive data is achieved through interrupts. Upon receiving the information, an interrupt is generated, the driver will apply a sk_buff (skb) in interrupt handler, and the data read from hardware is placed to applied buffer. Next, some information is filled in sk_buff. skb-> dev = dev judges protocol type ofreceived frame, and fills in the skb-> protocol (multi- protocol support). Pointer skb-> mac.raw point to the hardware data and then discard the hardware frame head (skb_pull). Also skb-> pkt_type is set, indicating the type of data link layer. If the data packet is obtained, then net_rx() is implemented, net_rx () is a subroutines of data reception, it is called by interrupt service routine. Last the data is sent to the protocol layer by calling netif_rx (). netif_rx () put data into the processing queue and then return, the real processing will be implemented after interrupted return. Thus interrupt time is reduced. After netif_rx () is called, the driver cannot be saved in data buffer skb. In the protocol layer, flows control of receiving data packets is divided into two levels: first, netif_rx () function is limited the number of data frames from the physical layer to protocol layer. Secondly, each socket has a queue, which limits the number of data frames from the protocol layer to the socket layer. In transmission, the dev-> tx_queue_len
parameter in driver limits the length of the queue.

F. Interrupt Hhandler

In the open function, the interrupt is applied, the interrupt handler is net_interrupt. The preparation of writing this function is to understand the interrupt process of network control chip. For the CS8900A chip, this process can explain the flow chart using fig. 2.

First kernel need to read ISQ (Interrupt Status Queue)
value, ISQ event has the following 5 types:

```
#define ISQ_RECEIVER_EVENT        0x04
#define ISQ_TRANSMITTER_EVENT     0x08
#define ISQ_BUFFER_EVENT          0x0c
#define ISQ_RX_MISS_EVENT         0x10
#define ISQ_TX_COL_EVENT          0x12
```

After receiving a data packet (RxEvent), to deliver
function net_rx() to process. Interrupt handler parameter is set as follows:
Static void net_interrupt(int irq， void *dev_id， structpt_regs *regs

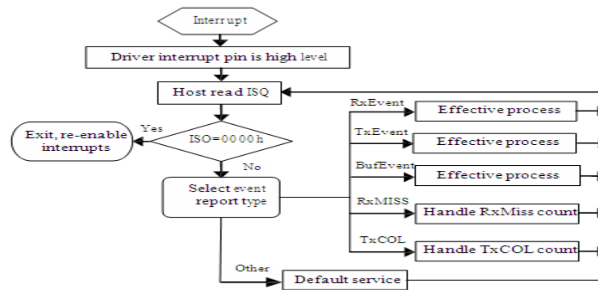

Figure 2.Interrupt handling flowchart

6. DRIVER TEST

Driver is only provided interface between kernel and user. To use the device, not only to install the driver, but also must write an application to use it.
After the network driver is designed, the kernel module can be compiled, and the custom kernel module as part of the system source code is compiled a new system.

A. Configuring The Kernel

In Linux2.6 kernel, want to compile the kernel module, first it is necessary to configure and construct properly kernel source tree in / usr / src, that is, to extract the need kernel source code to / usr / src /, and to use the command 'make menuconfig'or 'make gconfig' configure the kernel under the main directory of kernel source (Here is / usr/src/linux-2.6.18.3), then using the 'make all' to compile the kernel completely.

B. Take CS8900A NIC driver for example, introduce how to compile network device driver into the kernel.

z To create a new directory cs8900a under the directory 'drivers' of system source code tree;
z To copy cs8900a.c and cs8900a.h to drivers/cs890a directory;
z To compile 'Makefile' file under drivers/cs890a directory;

#Makefile for CS8900A network Driver
obj-$(CONFIG_DRIVER_CS8900A) +=cs8900a.o
z To compile 'Kconfig' file under drivers/cs890a directory;

#Just for CS8900A network device
menu "CS8900A network device support"
config DRIVER_CS8900A tristate "CS8900A support" This is a network driver module for CS8900A.
endmenu
z To add a line in front of'endmenu' statement of
'Kconfig' file In the 'driver' directory source
"drivers/cs8900a/Kconfig"
In main directory of kernel source code tree, "CS8900A
network device support [y / n /?]" can be found in the option
'Device Drivers' through command 'make menuconfig' or
'make gconfig' , if select "y", the kernel will provide support for network driver.
To recompile the kernel can be obtain kernel of
supported CS8900A card, then download the kernel to the development board. By configuring network parameters, the behavior of the network card driver can be test.

7. CONCLUSION

In recent years, Internet has a rapid development. Remote monitoring and remote maintenance become very easy after embedded system access internet, so the network of embedded systems is very important. Embedded system achieved internet access; its prerequisite is that system software has TCP/IP protocol support. Fortunately, Linux kernel provides support for multiple protocols including TCP/IP. This paper takes network chip CS8900A for example, and introduces the key process of implementing network driver.

8. Acknowledgment

This research was supported by the Open Project Program of Key Laboratory of Intelligent Manufacture of Hunan Province (Xiangtan University), China (No.2009IM03).

References

[1] Q.Sun, Developing Detail Explain of Embedded Linux Application, Beijing, Posts & Telecom Press, July 2006.

[2] T.Z.Sun and W.J.Yuan, embedded design and Linux driver development guide, Beijing, Publishing House of Electronics Industry, October 2009.

[3] M.liu, Embedded system interface design and Linux driver development, Beijing, Beihang University Press, May 2006.

[4] L.G.Zhou, Example of ARM Embedded Linux System Building and Driver Development, Beijing, BeihangUniversity Press, 2006.

[5] C.Qian, R.H.Xu and Q.R.Wang, "Device Driver Development Based on Linux Operating System", Control & Automation,2004,(09).

[6] Q.N.Cao, B.Zhao and K.Y.Meng, "Design and realization of embedded linux network communication system based on ARM9 platform ", Journal of Northwest University(Natural Science Edition), 2009,(1).

[7] J.Zhao,X.Q.Ding, "Development and Implementation Principle of Network Driver Based on Embedded Linux", Microcomputer Information, 2008,(17).

[8] F.J.Li and W.D.Jin, "Research and Implementation of Network Driver in Embedded Linux", Modern Electronic Technique, 2005(16)

[9] W.Su, "Design of Linux Network Device Driver" Financial Computer of Huanan, 2005(06).

[10] D.Cao and K.Wang, "Research of Network Device Driver based on Linux", Computer Knowledge and Technology, 2005(21).

[11] Q.Wu, SH.H.Zhou and ZH.X.Ma, "Development of Linux Network Driver Based on USB Device", Microcomputer Information, 2007,(02). V12-448