Multimedia Collection and Transmission Based on ARM11 Embedded Wireless Terminal Device

Bai Changqing*
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications
Beijing, China
bcq@bupt.edu.cn

Hu Anyang
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications
Beijing, China
zzuhay@163.com

Abstract—This paper presents an embedded terminal design based on ARM11 to implement multimedia information sharing and relay transmission between wireless terminal devices. Multimedia collection and wireless transmission to each other are easy to be realized based on an embedded board combined with a CMOS camera module and a SDIO wifi module. After the procedure of installation of cross complier on PC, and transplantation of bootloader, linux kernel and file system on board, through programming application based on V4L2, the camera can be controlled to capture a picture, which is compressed into jpeg format by calling libjpeg, another application is applied to wirelessly send pictures to each other through TCP socket network programming between two embedded devices. This design can widely be used in fields like modern security, monitor system for safety production, cloud camera and so on.

Keywords-ARM11; multimedia collection and wireless transmission; V4L2; libjpeg; socket

I. INTRODUCTION

With the rapid development of mobile internet, the relationship between wireless terminal devices and people is becoming closer and closer, pictures taking by phone, pictures sharing through Weibo or WeChat is becoming more and more popular among the young generation. This paper is based on ARM11 development board, with combined with a CMOS camera which can be used to capture pictures and a wifi module which is used for wireless transmission, hardware environment can be easily constructed. Otherwise, C language programming applications based on V4L2, a programming frame to control the camera to capture a picture, and socket network programming, by which two terminal devices can send and receive pictures between each other, are the key to make this design work efficiently. This design involves basic procedure of embedded development, pictures collecting and processing and socket network programming.

OK6410, one kind of Samsung's ARM11 development board, whose processor is S3C6410 inside, has the feature of high performance and low power. Using OK6410 as a basis platform, by transplanting UBoot1.1.6, embedded linux kernel linux-3.0.1 and a file system formatted into yaffs2, the fundamental running environment is constructed. Then power off the board, link camera OV9650 with the CMOS interface to the board's CAM cassette and the wifi module WM-G-MR-09 with SDIO interface to its SD cassette, an integrated hardware which can run applications on it to collect and send or receive pictures is complete. However, applications are still needed to make these functions come true. Application development is easier since drivers of CMOS camera and SDIO wifi module have already been integrated into the version of linux kernel 2.4 and above[1]. All developers need to focus on are their certain application requirements.

II. DEVELOPMENT PROCESS OF EMBEDDED SYSTEM

Development process of embedded system includes the following steps[2]:

- Build development environment.
- Transplant bootloader.
- Transplant embedded kernel of operating system.
- Transplant a file system.
- Develop applications.

For development machines (mostly PCs), a certain operating system and a cross complier that can run on it are essential. For development board, a bootloader, an embedded kernel of operating system and a file system are the basis environment of software to load an application then run it.

Fig .1 shows the development procedure of an application. First, program the C source codes under a develop machine, which has already installed the cross complier. Second, compile the source codes then an obj file is got. Third, link the obj file with cross function lib, a binary program is generated. At last, copy this binary program into the board (in this design, we do this through SD card) to debug. If the program cannot work as expected, modify the source codes then do the above procedure again until it can work as wanted.

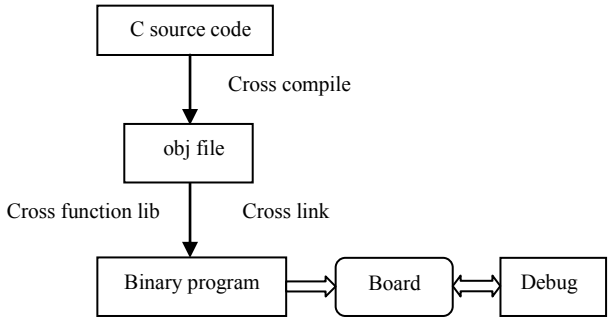

Figure 1. Development process of an application

A. Establish the development environment on PC

This section is about establishing the development environment on PC. There are two parts to establish. One is Linux operating system installation, the other is cross complier installation. Since the Linux operating systems and their installing procedure are various but easy, we do not introduce the installation of them here now. However, a virtual Ubuntu10.10 has been installed with the help of VMware, a virtual workstation software running on the PC, by which we are able to program an application, and then compile it with a cross compiler of embedded system.

Cross compile, which refers to compiling the C source codes in the development machines, namely a PC, to get a binary program so that it is able to run on an embedded linux system, is essential. In this design, the version of cross complier is arm-linux-gcc4.3.2. Open a command terminal of Ubuntu, type "arm-liunx-gcc -V" to check cross compiler's version. If "gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)" is displayed then, it has been installed correctly.

B. Transplant Bootloader

Bootloader, namely the set-up procedure, is the first program that runs after the board is powered on[3]. It's used to guide the embedded operating system, and run before the kernel of operating system. After this procedure, initialization of the board devices, establishment of memory mapping and other preparatory work are finished, then rights of control are handed over to the operating system. There are two common types of Bootloader, one is U-Boot, another is Vivi. Since U-boot is an open source software, which means it's free and has powerful functions, widely used in embedded boards, we choose a compiled already mirror image, u-boot.bin, as OK6410's Bootloader to transplant.

C. Transplant kernel of embeded operating system

Kernel, the most import part for an operating system, is core of operating system. It's responsible for kinds of resource management[4]. Configuration of kernel should match the hardware that the board can provide, a cross compiler working in PC is a must software to compile kernel source code to an image file that can run on certain development boards. Operating system of this design is embedded linux, whose version is linux-3.0.1. Since version of embedded linux-2.4 and above has been integrated with drive programs of CMOS camera and SDIO wifi module, there is no need to program these drive programs any more. We choose the linux image, zImage provided by Forlinx for OK6410, as the kernel of OK6410 to transplant.

D. Transplant file system

File system is a method to store and organize data of computer to make accessing and searching easier. There are two kinds of file system used in embedded linux system, one type is random access memory, like ramdisk, ramfs/tmpfs and so on, another type is read only memory (mostly is Flash), like jffs2,yaffs, cramfs and so on[5].

OK6410 board supports the storage of 1G Nandflash, so file system formatted by yaffs2 can be applied to it. Yaffs2, the first open source embedded file system designed specially for type of Nandflash, is able to support devices with mass storage. We choose a file system that has been compiled already into yaffs2 format, rootfs.yaffs2, as OK6410's file system to transplant.

Get the u-boot.bin, zImage and rootfs.yaffs2 ready, copy them to the SD card, and set development board start-up from SD card, then power on the board, auto-installation will be finished in half an hour. Now you've got a new running environment in OK6410. However, before reboot the board again, remember to set it start-up from nandflash back, or next time, bootloader, linux kernel and file system will be installed again.

The following work is linking camera and wifi module to the board's CAM and SD interface respectively, and program applications to control the two parts.

III. REALIZATION OF APPLICATIONS

This design involves two stages of pictures collecting and wireless transmission, which means at least two applications need to be completed. The application of pictures collecting is based on Video4Linux2, which is a application frame programmed by C language. Two processes are used in pictures collecting application, the main process is used to receive exit order passed from son process, and it is also responsible for real time monitoring. The son process controls the camere to capture a picture (based on v4l2 frame) and reserve it to be a jpge file (by calling functions of libjpeg) according to capture order typed from the command line, or send a singal to the parent process to exit from this application according to the typed exit command as welll.

Pictures wireless transmission application is based on network programming of Clietn/Server mode, TCP protocol is used to ensure the correction of data flow instead of UDP. Besides, In addition to simplex, I/O multiplexing is added to both server side and client side, namely the sender can receive pictures from receiver, while the receiver can also send pictures to senders.

A. Realization of picture collecting

Pictures collecting of CMOS camera under embedded linux can be achieved with the help of Video4Linux2 (or V4L2). V4L2 is the second version of V4L2, which is a set of application programming interfaces of vedio and picture developing. Through V4L2, it is easy to program an application to realize picture collecting. To disply the vedio on LCD of development board, framebuffer is needed. Framebuffer is a certain device in Linux, it is used to display the vedio get form camera, mostly lies under the folder of /dev. For the OK6410 board, /dev/fb0 is the framebuffer file to use.

As said before, drive program for CMOS camera OV9650 is no need to program since the linux kernel used has already installed. The main steps of pictues collecting through OV9650 are as followd:

1) *Control the camera to capture video data.*
2) *Convert the captured video data to bit flow of RGB format.*
3) *Display the bit flow of RGB format on board's LCD in real time.*
4) *Compress the bit flow to a jpeg file to get a certain*

picture.

Cooperating with V4L2, the main function of display video and capture a picture includes the following steps:

1) Call open() function to open the CMOS camera and framebuffer device:

```
#define V4L2_FILE "/dev/video0"  // camera
#define FB_FILE "/dev/fb0"   // framebuffer
int v4l2_fd = open(V4L2_FILE, O_RDWR);
int fb_fd = open(FB_FILE, O_RDWR);
```

2) Call mmap() function to mapping framebuffer to certain buffer field in memory, then the operation to I/O of the buffer will be the same as operation to framebuffer:

```
screensize = 480 * 272 * 16 / 8;
fb_buf=(char*)mmap(0,screensize,
PROT_READ|PROT_WRITE,MAP_SHARED, fb_fd, 0);
```

3) Parameters configuration of vedio mode and format:

```
// type of of data flow, need to be
// V4L2_BUF_TYPE_VIDEO_CAPTURE
fmt.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
// width, height and depth, multipe of 16
fmt.fmt.pix.width = 320;
fmt.fmt.pix.height = 240;
fmt.fmt.pix.depth = 16;
// storage type of video data
fmt.fmt.pix.pixelformat=V4L2_PIX_FMT_RGB565;
 // set the above parameters
ioctl(v4l2_fd, VIDIOC_S_FMT, &fmt);
buf = malloc(320*240*3);
// beginning of video recording
ioctl(v4l2_fd, VIDIOC_STREAMON, &type);
```

4) The main process is used to display real time vedio, and receive exit order from son process to exit from the application:

```
while (1) {
    if (!c_flag){
        n=read(v4l2_fd, buf, 320*240*2);   // read vedio
data to buffer
        // display vedio on LCD
        show_rgb565_img(fb_buf, (__u16 *)buf);
    } else {
        printf("c_flag=[%d]\n", c_flag);
        printf("Stop while loop!\n");
        break;
    }
    // receive exit order to exit from application
    signal(SIGUSR1, change_flag);
}
```

5) The son process calls function to capture a certain picture by compressing data flow to a jpeg file, or send exit order to parent process according to different commands typed in the command line:

```
while(1){
    while (temp != 'c' && temp != 's'){
        printf("Input 's' to stop or 'c' to capture a picure:");
        scanf("%c", &temp); getchar();
    }
    if (temp == 'c') {
    //function of compress the data flow to a jpeg file
    v4l2_save_rgb_to_jpeg(v4l2_fd,320,240);
    printf("Input 's' to stop or 'c' to capture a picure:");
    scanf("%c", &temp);getchar();
    } else if (temp == 's') {
    // send exit order to parent process
    kill(getppid(), SIGUSR1);
    break;
    }
}
```

6) The main process exits from the application after received exit order passed from the son process. Before exiting, program hill free used memory and close opened file descriptors.

```
munmap(fb_buf, screensize);  // free memory
close(v4l2_fd);   // close camera
close(fb_fd);   // close framebuffer
```

B. Pictures compressing

Output of the CMOS carema is RGB data bit flow, to display them as a picture on a terminal, the data flow need to be compressed to in a picture format like BMP, JPEG, PNG and so on[6]. In this design, libjpeg, a lib functions of jpeg realized by C language, is applied to compress the RGB data flow to a jpeg picture. However, before the employment of libjpeg, it should have been installed already in Ubuntu. For users of Ubuntu, libjpeg will be installed after the root user of Ubuntu types ~~–a~~pt-get isntall libjpeg" in the command line then click the enter .

The function used in main function to compress the data is v4l2_save_rgb_to_jpg(), the file descriptor of carema is passed into it. The following is procedure of compressing[7]:

1) *Apply an object of compressing , assign an error processor to this object, the initialize it.*
2) *Open a new file to save as a picture, notice that this file must be open by binary mode.*
3) *Set parameters of picture compressing, at least width, height, color channel and color space are set correctly.*
4) *Call funtions of libjpeg to start compressing.*
5) *At the end of compressing, call functions of libjpeg to stop and free occupied resources.*

At last, with the help of cross compiler, compile this application source code (for example v4l2_get_pic.c) to get a binary executeable program (for example v4l2_capture). Remember to link libjpeg into the source code with the following command with a root user:

```
# arm-linux-gcc v4l2_get_pic.c  -o v4l2_capture –ljpeg
```

C. Realization of pictures transmission

After reserve the vedio image to a jpge file, it can be wireleely sent to a remote terminal device to display. Multimedia information including characters, pictures, audio and video . Pictures transmission between two devices can be deemed as messages transmission between two different processes, which can be achieved by socket network programming. Both TCP and UDP can work in socket communication, however, to ensure data's reliability, we choose TCP to establish an ordered, reliable and object-oriented connection between two devices[8].

Socket communication is based on Client/Server mode, which means a device is took for client side, while another is took for server side. Correspondingly, two different applications are needed to make the success of connection come true. The theory of socket communication based on TCP is as followed:

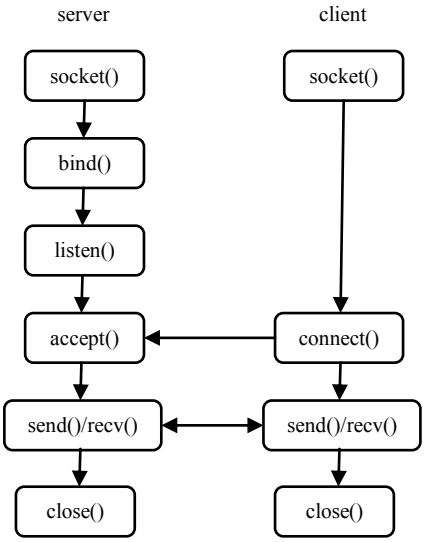

Figure 2. Socket communication of TCP

In Fig .2, for the server side, socket() function is used to new a socket descriptor through TCP, socket is the object for two different devices to communicate with each other, access to socket is access to scoket descriptor. Bind() function is used to bind certain IP address and port to listen to the newed socket. Listen() function's duty is announcing to the client side that connection can be established now. Accept() function gets the socket descriptor that connects to the server. Send() and rece() functions are responsible for sending and receiving of TCP data flow. At last, after data transmission is finished, close() functon closes all opened socket descriptor. So far, for the server side , work is done.

However, the client side also need socket(), send(), recv() and close() functions to do the same work as they do in server side. What's different with the server side is another function, connect() is applied to actively establish the connection for data transmission between the server side and the client side[9].

The main work server side does includes the following steps:

1) Call scoket() function to new a socket descriptor and assign it to a certain communication domain and socket type:

```
sockfd=socket(AF_INET, SOCK_STREAM, 0);
```

AF_INET stands for protocol of Ipv4 is used in socket communication, which including wired and wireless types, SOCK_STREAM stands for TCP is assigned to establish an ordered, reliable and object-oriented connection instead of UDP.

2) Call bind() function to bind the newed socket to certain IP address and certain port:

```
struct sockaddr_in my_addr; //declare socket address
bzero(&my_addr, sizeof(my_addr));//clean the address bit
bind(sockfd, (struct sockaddr *) &my_addr,
sizeof(struct sockaddr);
```

3) Call listen() funciton to announce that the server is ready to be connected:

```
// lisnum stands for the max connection num
listen(sockfd, lisnum) == -1) ;
```

4) Call accept() function to connect with the client side, retrun value is the socket descriptor connected:

```
int new_fd = accept(sockfd, (struct sockaddr *) &their_addr, &len);
```

5) Call send() and recv() functions to send and receive data:

```
FILE * fd_read = fopen("./muduo.jpg", "r");  // open the picture to be sent
int res = fread(buf, 1, MAXBUF, fd_read);  // read bytes of the picture to buffer
fclose(fd_read);  // close file descriptor
// send the buffer to the scoket descriptor that connected to the server side
int len = send(new_fd, buf, res, 0);
  .....
  bzero(buf, MAXBUF + 1);  // clean buffer
  len = recv(new_fd,buf, MAXBUF, MSG_WAITALL); // read data flow that received to buffer
  FILE * fd_write = fopen("./recv.jpg", "w");  // new a file to reserve the picture
  res = fwrite(buf, 1, len, fd_write);  // write buffer to the newed file
  fclose(fd_write);
```

6) Call close() function to close all opened socket descriptor, then connction is over:

```
close(new_fd);
close(sockfd);
```

The main work client side does includes the following steps:

1) Same as server side, client side also needs to call socket() function ot new a scoket descriptor：

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);  // Ipv4, connect by TCP
```

2) Initialize the IP address and port information that to be connected to, call connect() function to actively establish connection between client side and server side:

```
struct sockaddr_in dest; // declare destination IP address to connect to
bzero(&dest, sizeof(dest));  // clean address bit
dest.sin_family = AF_INET;  // address type
// pass IP address form command line
inet_aton(argv[1], (struct in_addr*) & dest.sin_addr. s_addr);
// pass prot number from command line, which should be the same with that binded in server side
dest.sin_port = htons(atoi(argv[2]));
connect(sockfd, (struct sockaddr *) &dest, sizeof(dest));
```

3) Call send() and recv() functions to send and receive data:

```
// reveice data to buffer
recv(sockfd,buffer,MAXBUF,MSG_WAITALL);
FILE * write_image = fopen("./out.jpeg", "w");  // new a file to reserve picture data
fwrite(buffer, 1, len, write_image);  // write buffer to the picture file
fclose(write_image);  // close file
.....
FILE * fd_read = fopen("./duoduo.jpg", "r");  // open the picture to be sent
fread(buffer, 1, MAXBUF, fd_read);  // read data to buffer
```

fclose(fd_read); // close file

send(sockfd, buffer, res, 0); //send buffer to scoket descriptor

4) Call close() function to close opened socket descripotr, then connection is over:

close(sockfd);

However, except the above programs of server side and client side, the mechanism of select() is still essential in the the two program above to achieve I/O multiplexing so that both server side and client side all can obtain sending and receiving funtions[10].

IV. CONCLUSION

This paper introduces a general solution of pictures colletion and sharing. By concatenating CMOS camera to OK6410 board's CAM interface, and SDIO wifi moudle to its SD interface, along with OK6410 itself, the basic hardware is constructed. After transplanting bootloader, linux kernel and file system to the board, the basic software environment is built. Since drivers of camera and wifi module have been integrated in the kernel that transplanted into the board, there is no need to develop drives for them any more, which helps to make a product quickly, developers can focus on their own requirements and how to realize them. Experiments have proved that this design can work efficiently and conveniently. Pictures can be captured and shared between two different wireless terminal devices successfully. However, two different modules, camera and wifi, occupy two cassettes of borad, which means it is impossible to extend other modules with the two occupied cassettes any more. Consider to integrate the two modules to be one unit and make them work respectively under the manipulation of software, so that one cassette is able to be saved for other use. Also access control can be took into account to make only the user who has the qualification can manipualte the operation of board.

ACKNOWLEDGMENT

This project was supported by China Meteorological Administration Project of Comprehensive Climate Observation Metrological Verification System (210280).

REFERENCES

[1] K uang Shunming, He Xiaojian. Design and application of CMOS device driver based on S3C2440. Journal of Electronic Measurement & Instruments, 2011,1,pp.110-114

[2] L i Shanping, Liu Wenfeng, Wang Huanlong. Linux and embedded system. Beijing: TSINGHUA UNIVERSITY PRESSS, 2006,pp.37-39

[3] Sun Tianze, Yuan Wenju, Zhang Haifeng. Design of embedded and development guide of Linux driver.Beijing: PUBLISHING HOUSE OF ELECTRONIC INDUSTRY, 2005,pp.95-98

[4] Sun Yanpeng, Peng Peng, Zhang Yuan. Linux transplantation based on the process S3C2440. Journal of Electronic Measurement & Instruments, 2009,2,pp.306-309

[5] Wei Wu,Chen Zongyu. Design and implementation of embedded remote video surveillance system. Journal of Electronic Design Engineering,2010,18(3),pp.62-64

[6] Song Lili. Image Capture and Pretreatment Device Design Based on Embedded System. Journal of Harbin University of Science and Technology,2010,15(3), pp.23-26

[7] Zheng Jiehang. The Design of Network Video Monitoring Based on ARM.Ms D Thesis. Wuhan:Wuhan University of Technology,2010

[8] W.Richard Stevens, Stephen A.Rago. Advanced Programming in the UNIX Environment. You Jingyuan, Zhang Yaying, Qi Zhengwei. Beijing: POSTS & TELECOM PRESS, 2012,pp.437-440

[9] Cai Minqiang, The Design Method of Network Chat System Based on Socket and Cloud Computing.Computer Science & Service, 2012, pp.610-613. Doi: 10.1109/CSSS.2012.157

[10] Neil Matthew, Richard Stones. Beginning Linux Programming. Chen Jian, Song jianjian. Beijing: POSTS & TELECOM, 2010,pp.555-558