The design of touch screen driver based on Linux input subsystem and S3C6410 platform

[a]Liangfeng Fu*, [b]Linbo Xie, [c]Zhigang Zhou
The internet of things Engineering Academy
Jiangnan University
Wuxi, china
[a]1753038435@qq.com, [b]xie_linbo@jiangnan.edu.cn, [c]914264582@qq.com

***Abstract*—A kind of resistive touch screen driver was developed based on S3C6410 platform and input subsystem in kernel of Linux 2.6 and higher. And to verify the driver's function, this paper also did a test for it in detail. The test results show that the touch screen driver can properly capture the information of touching coordinates and up/down state. Besides, it can also obtain the real-time coordinates of sliding touch state. This touch screen driver can be easily applied in the embedded GUI systems such as android, Qt, miniGUI and etc.**

Keywords-input subsystem, S3C6410, driver

I. INTRODUCTION

The process of developing an input device driver has its traditional method, that method always contains the definition and realization of the devices' operational function pointers in the structure of *file_operations*, such as *open, read, write, ioctl, llseek* and etc [1]. However, the realization of the *file_operations* often occupied a lot of development time, and it's also an error-prone area. In order to avoid doing this work, this paper designed a touch screen driver based on Linux input subsystem and the S3C6410 platform, and a detailed test process was given to verify its function.

II. AN ANALYSIS OF THE INPUT SUBSYSTEM IN LINUX2.6 KERNEL

A. Input subsystem's analysis

The 2.6 edition of the Linux kernel introduced an input subsystem framework. It's on the same level of other subsystems like file system and network system. The introduction of the input subsystem makes it more efficient to develop drive software of mouse, keyboard or touching screen in Linux. Further study of the subsystems in the Linux kernel shows that the system packed frequently used *function sets* for specific hardware devices. Linux input subsystem is constructed by input driver layer, input core layer and input event handle layer [2]. And the structure within the Linux system is illustrated in Fig.1.

The input driver layer control the specific hardware by configuration of registers and interrupt process. The input motion of input devices, such as click, press and touch, is abstracted into an event, which will be reported to the input core layer. And this part shall be developers' work.

Input core layer connect two layers of input driver and

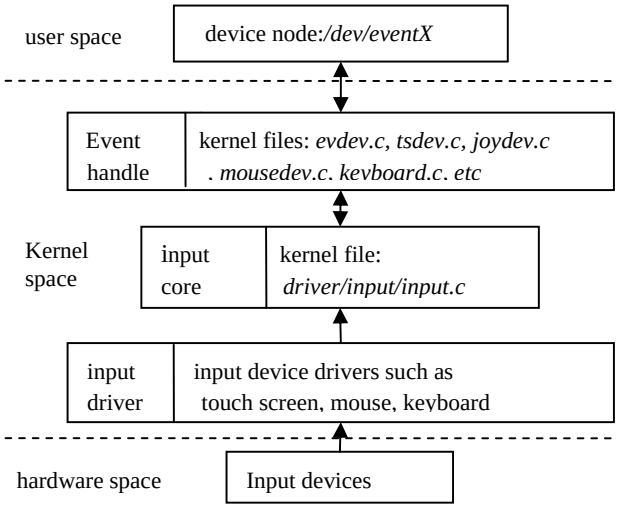


Figure.1. Structure of Linux input subsystems

event handle. When an input event occurred, the core layer receive input messages from the input layer and notice the event handle layer that an input event happened, and provide event handle a call interface. And this part is realized by the kernel.

Event handle layer get the input messages by calling the input layer's interface, and then pack the messages into a specific structure which would be inserted into the buffer of *client_list* in *evdev.c* so that the APs(application programs) can easily get it. Besides, event handle also defined unified device file node for each input devices' drive software. And this part is also realized by the kernel [3] [4].

Concerning that input core layer and event handle layer defined the unified file operation interfaces for AP, the drives of the input subsystems shall only report the input message to the input core according to the interface provided by the input core layer. So there's no need to care about the file operation interface.

B. Driver development mode under the input subsystem

The device drivers under Linux system have an inseparable relationship with the kernel. The Equipment Management System in the kernel defined a special mode to seek and manage the device drivers, this mode should be abided by driver developers. In the mode, some structures and functions that describe the equipment have their own names and parameter, which shall not be modified by

developers. Usually, the knowledge of the modes plays a very important role in driver development.

The developing mode based on Linux input system is simply concluded as follows [5]:

- Input device's descriptive structure
  input_dev //This structure defines an input device
- Input device drivers' register and unregister
  *int input_register_device(struct input_dev *dev)*
  *int input_unregister_device(struct input_dev *dev)*
- Input device drivers' event supporting
  set_bit(event type,xxx_dev.evbit)
  The drivers use *set_bit()* to inform input subsystem what events they supports. The structure *input_dev* has a member named *evbit* which represent the supported events. Frequently used classes of event types include EV_KEY (press the keys), EV_REL (relative location), EV_ABS (absolute location) and etc. The key event, mouse event and touch screen event are sorted into EV_KEY, EV_REL and EV_ABS respectively.
- Input device drivers' event reporting
  *void input_report_key(struct input_dev *dev, unsigned int code, int value)*
  *void input_report_rel(struct input_dev *dev, unsigned int code, int value)*
  *void input_report_abs(struct input_dev *dev, unsigned int code, int value)*
  input_sync(input_dev)
  These function prototypes above are used for reporting keyboard/key, mouse and touch screen event respectively. In which the 'code' means the code of the input event. If the event type is EV_KEY, then it represents the pressed/released button's code on the keyboard, this is specified in the head file of */include/linux/input.h*. And the 'value' means the value of the input event. If the event type is EV_KEY, the value of '1' refers to a press action and '0' refers to a release action. *input_sync* used for the synchronization of event, which will inform the event receiver that a complete report has been sent over.

III. RESISTIVE TOUCH SCREEN'S WORKING PRINCIPLE AND ITS CONNECTION WITH PROCESSOR S3C6410

A. The working principles of resistive touch screen

The body of the resistive touching screen is a thin multi-layer composite film that fits the monitor well. The first layer is the basic layer made by glass or organic glass. The second layer is the insulate layer and the third layer is the polyhydric resin layer which is coated with a conductive layer. When one's finger touch the screen, the two insulated layer come together under pressure. And one of them is put through the uniformed voltage field of the coordinate Y, whose voltage is 5V, which makes the voltage of determinative layer change from 0V. Once the CPU's controller detected the change, it would get the coordinate of Y by compare the voltage with 5V through A/D transform. The coordinate of X would be got at the same way.

The operation modes of touch screen controller of S3C6410 are as follows [6]:

- Normal Conversion Mode
  The operation of this mode is identical with AIN0~AIN3's. It can be initialized by setting the ADCCON (ADC Control Register) and ADCTSC (ADC touch screen control register). All of the switches and pull-up resister should be turned off (reset value 0x58 makes switches turn-off). The converted data can be read out from ADCDAT0 (ADC conversion data 0 register).
- Separate X/Y position conversion Mode
  This mode consists of two states; one is X-position measurement state and the other is Y-position measurement state. The touch screen can use either of the conversion state. A single X/Y coordinates conversion is describe like this: The X coordinate state write the converted X coordinate data into register ADCDAT0, and the Y coordinate state write the converted Y coordinate data into register ADCDAT1. An INT_ADC interrupt is generated each time the conversion is done.
- Automatic mode of X/Y coordinates transform
  In this mode, the controller converts the coordinates automatically and writes them into ADCDAT0 and ADCDAT1 respectively. An INT_ADC interrupt will be triggered once the conversion is done, and then the ADC conversion will be closed up automatically at the same time. For the next conversion, it should be turn on manually.
- Waiting for interrupt mode
  Touch screen controller generates an interrupt signal (INT_TC) when the stylus pen is down or up. And this interrupt signal will be cleared automatically.

B. The connections between S3C6410 and touch screen

A four-line resistive touch screen is utilized for the design of its driver in this article. Fig.2 shows the connections between the four-line resistive touch screen and the S3C6410 processor.

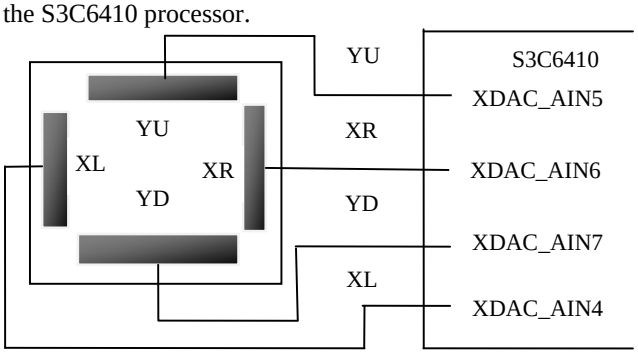


Figure.2. The circuit connection between touch screen and the S3C6410 processor

IV. THE REALIZATION OF TOUCH SCREEN DRIVER BY USING LINUX INPUT SUBSYSTEM

Combining the operation mode of S3C6410's touch screen controller, the principle of Linux input subsystem, and the general rules to write a kernel module in Linux (driver is a kernel module in Linux), we can easily imagine that the major work of designing a touch screen driver based on Linux input subsystem include the realization of module load/unload functions, the INT_TC/INT_ADC interrupt handler, and the touch-event's report function. Main work of each function is summarized as follows (program codes are omitted as the space limitations) [7] [8].

A. The function for module loading

The main tasks of this function are as follows: applying and initializing an input device, setting the event types of the input device, registering the INT_TC and INT_ADC interrupt, configuring registers to set s3c6410 work in the interrupt mode of wait for touching, registering an input-device driver.

B. The function of the INT_TC interrupt handler

The INT_TC interrupt will be triggered when touch screen is touched or released, this interrupt handler function can judge the touch screen's state(touched or released) by reading the value of touch screen controller's data register on S3C6410 ,this will determine whether to open the ADC converter or not.

C. The function of the INT_ADC interrupt handler

The INT_ADC interrupt will be triggered at the end of every ADC conversion. this interrupt handler function will firstly convert the original data produced by a touch motion for eight times, and then it will start a timer to report the average coordinates to the input core by calling the data report function when the timer expires. At last, it will set the touch screen controller to the interrupt mode of wait for releasing.

D. The the function for data's reporting

This function's major work is to report the touch event to the input core, after the report, it will do some necessary replacement to wait for the next touch, and then it will restart the ADC converter to get the coordinates of sliding state.

E. The function for module's unloading

This function will be executed with the driver module unloading from the kernel. Its major work is to release the resources occupied by the driver, such as unregistering this input device, releasing the INT_TC and INT_ADC interrupts, releasing the IO memory and etc.

At last, the following code should be added to the end of the drive source file to let the module load/upload function be the inlet/outlet of this kernel module.

module_init(s3c6410ts_init);
module_exit(s3c6410ts_remove);

So far, a full touch screen driver is almost completed

F. The touch-event's process flow of touch screen driver

The author's design of processing flow which based on S3C6410 and Linux 2.6.36 is shown in Fig.3.

In Fig.3, the dotted lines indicate there will have a function-call triggered by interrupt or timer. The dotted line

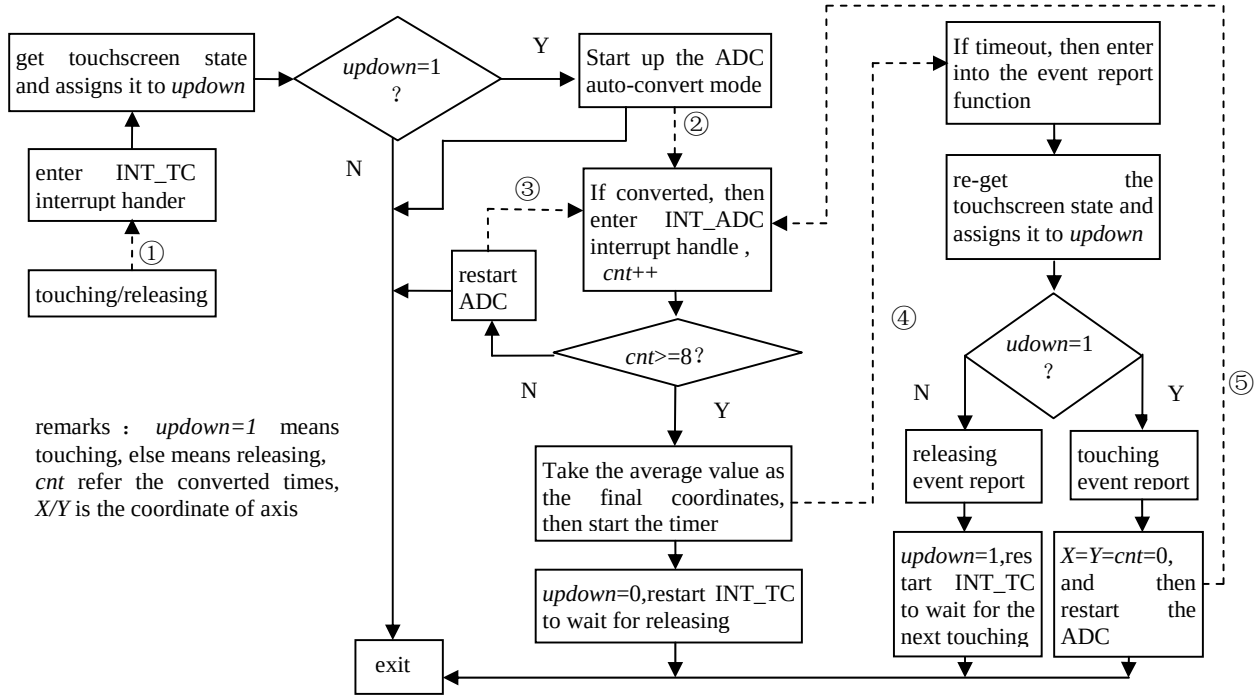


Figure.3. Touch screen driver's event process flow

"①"means touching or releasing will call the function for INT_TC interrupt handling, the dotted line "②③⑤" means the ending of ADC conversion will call the function for INT_ADC interrupt handling, and the dotted line"④"means the timer's timeout will call the function for event reporting.

In order to get more accurate coordinates, this design take the average data of eight times conversion as the final coordinate data.

Considering the demand of getting the real-time coordinate when the touch-pen is sliding (touched but without releasing) on the screen, the author uses a timer to achieve this function, this started timer will report the coordinate data to the input core at a regular interval (10ms), as can be seen from the Fig.3 that after the touch event was reported, the ADC conversion is started again, the time-length of the timer determines the accuracy of the real-time coordinates on the sliding state.

V. THE TOUCH SCREEN DRIVER'S TEST

After the driver is designed, it should be supported by the kernel at first, and then a test program should be written to verify whether it can work normally or not.

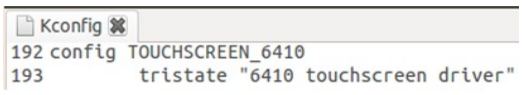
A. Compile the touch screen driver into the Linux kernel

There are two ways to let the kernel support the new driver: to compile it into the kernel or to compile it as a loadable module. This paper chooses the first one.

1) Modification of the Linux kernel's configuration file and compilation file. Coping the touch screen driver (6410_ts.c) to the dir(directory, similarly hereinafter) of /driver/input/touch-screen in linux2.6.36.In the dir, there are some existed touch screen drivers based on other architectures, and another two files named *Kconfig* and *Makefile* which used to decided whether to compile a driver into the kernel or not. In order to let the touch screen driver (6410_ts.c) be supported by Linux kernel, the *Kconfig* and *Makefile* should be modified.

- to modify the *Makefile*
  Run the command *#gedit Makefile* to open *Makefile* in the linux terminal, and then add a new build option to it. The result is shown at line 56 in Fig.4(a).
- to modify the *Kconfig*
  Run the command *#gedit Kconfig* to open *Kconfig*. Then add two lines (shown at line 192 and 193 in Fig.4(b)) to it so as to generate a new configure option in linux kernel's configure menu.

(a)



(b)

Figure.4. The added contents in Makefile and Kconfig

2) Configure the linux kernel. Run command *#make menuconfig ARCH=arm* to enter kernel' configuration menu. Then choose the "device drivers->input drivers support->touch screen" to enter touch screen driver's configuration menu, and selected "6410 touch screen driver".

3) Compiling and running the linux kernel. Run command *#make zImage ARCH=arm CROSS_COMPILE=arm-linux-* to cross-compile the kernel. This will generate a file named "zImage", and then download it to S3C6410 hardware platform through *tftp* server. After the kernel is started, An item named "event1" can be found in the dir of file list which depict the input-devices in kernel's *sysfs* virtual file system [9], show as Fig.5, this is generated by Linux input subsystem for touch screen driver. Besides, a device node named "event1" also can be found under the dir of "/dev".

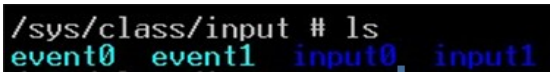


Figure.5. Generated file list of input-device in sysfs

B. The design of the test program

Test approach: firstly, geting test dada by calling touch screen driver's device node and APIs which supported by input subsystem's event handle. And then, printing these data on the console to make an analysis. Test program's major codes are as follows.

```
int main(void)
{
  struct input_event ev_ts;
  ts_fd=open("/dev/event1",O_RDWR); while(1){
    count=read(ts_fd,&ev_ts,sizeof(struct input_event));
    for(i=0;i<(int)count/sizeof(struct input_event);i++)
      if(EV_KEY==ev_ts.type)
        printf("type:%d,code:%d,value:%d\n",ev_ts.type,
              ev_ts.code,ev_ts.value);
      if(EV_ABS==ev_ts.type)
        printf("type:%d,code:%d,value:%d\n",ev_ts.type,
              ev_ts.code,ev_ts.value);
      if(EV_SYN==ev_ts.type)
        printf("sys event\n\n\n");
  }close(ts_fd);
}
```

In this program, "while(1)" here is for waiting the touch screen being touched. Once touched, the input subsystem will quickly save the coordinates and status information reported by touch screen driver to a memory buffer in a format of input event structure. "read()" (which is an API provided by input subsystem) is for getting these information, and the "for" loop is for resolving this information and printing them on the super console.

C. Testing

After the system's software and hardware environment were built. We copy the test program (ts_app.c) to dir of /home/fuliangfeng/touch-test. And then run command

#arm-linux-gcc ts_app.c –o ts_app2 to cross compile the test program. The result is shown in Fig.6.

Figure.6. Compile result of test program

After that, copying the executable file (ts_app2.exe) to the root dir of NFS file system, this file will be shared by S3C6410 hardware platform [10]. At last, run command *#./ts_app2* to execute the test program in the super console. When the screen is touched, results will be printed on the console, as shown in Fig.7.

Figure.7. Printing information of once touch

As we can see from Fig.7, the touch screen driver did three-times data report (sys event is the ending mark of each report), the first report occurred quickly when touching, the second report occurred 10ms later after the first one, and the third report occurred when releasing the touch pen. ‘type=3’ means the driver reported an absolute coordinate event (EV_ABS), ‘type=1’ refers it reported a key event (EV_KEY, press the touch screen can also generate a key event). ‘code=0’ refers to X-axis coordinate, ‘code=1’ refers to Y-axis coordinate, ‘code=330’ refers the key value is 330, ‘code=24’ refers the pressure is 24. ‘value=1’ means touching state, ‘value=0’ means releasing state, for other value, it refers to the value of absolute coordinate.

The test results show that the touch screen driver not only can capture coordinates and touch-release states, but also can obtain real-time coordinates under the long-time touching state.

VI. CONCLUSION

On the basis of this paper’s work, a touch screen driver based on other CPU arch can be easily developed just by changing some different register’s configuration and interrupt handler. Developer no longer needs to care about the definition and realization of device operation functions which occupied a lot of work in driver developing. Besides, this paper’s driver development includes the analysis of input subsystem, the detailed elaboration of driver development, and the procedure of how to compile a driver into the Linux kernel as well as how to call the driver in the test program. This will be conductive for a developer to understand the input subsystem and the mechanism of data interaction between APP and driver. Finally, the popular GUI systems such as android, Qt and miniGUI all support the linux2.6 and higher, so this touch screen driver can be applied in these embedded GUI systems easily.

VII. REFERENCES

[1] Weigong Chang, Zhonglin Ding, “The investigation of Touch Screen Panel driving program in embedded Linux system,” transl. J. Mirco-computer and Information. China, vol.23, 2007, pp.103-105.

[2] Bard Hards, “USB input subsystem, part I,” unpublished.

[3] Chuang Wang. “analysising the linux input subsystem,” Transl. J. Computer Development and Applications. China, vol.25, 2012, pp.70-72.

[4] Brad Hards, “Using the input subsystem,” unpublished.

[5] Shaopin Liu, “The researching of driver based on Linux input subsystem”, transl. J. Electronic Design Engineering. china, vol.20, No.17, 2012, pp.29-31.

[6] Samsung Electronics Co. Ltd, “S3C6410 datasheet,” unpublished.

[7] Jun Li, The detailed guide for developing Linux device driver, Beijing: People’s Post and Telecommunications, 2008, pp.36-43.

[8] Jonathan Corbet, Alessandro Rubini, Gerg Kroab-Hartman. Linux Device Drivers, 2rd ed., USA: O’ Reilly Media, Inc, 2006, pp.3-45.

[9] Yuanlou Gao, Xixin Zhao “The Implementation of Buttons Driver Using the Linux Input Subsystem Based on S3C6410 Platform,” in IEEE International Conference on Industrial Informatics, Beijing, 2012, pp. 281-286.

[10] Feiling embedded technology co., Ltd, Linux developing guide of ok6410. Baoding: Feiling, 2011.