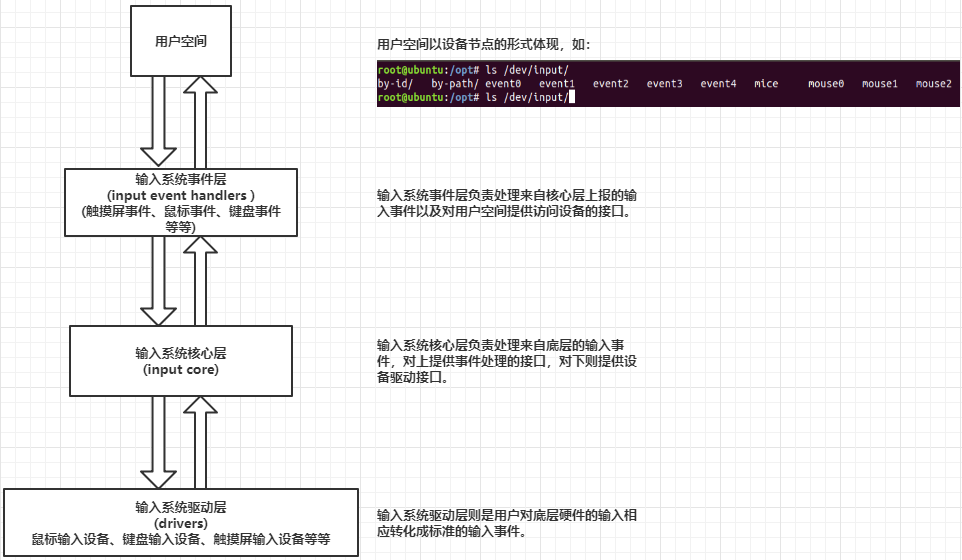
## 输入系统

### 3.1、什么是输入系统？

在了解输入系统之前，先来了解什么是输入设备？常见的输入设备有键盘、鼠标、遥控杆、书写板、触摸屏等等,用户通过这些输入设备与Linux系统进行数据交换，Linux系统为了统一管控和处理这些设备，于是就实现了一套固定的与硬件无关的输入系统框架，供用户空间程序使用，这就是输入系统。

### 3.2、输入系统应用框架描述

在Linux输入系统中，主要分三层进行管理，分别是input core(输入系统核心层)、drivers(输入系统驱动层)以及event handlers(输入系统事件层)，如下图所示，这就是Linux输入系统的基本框架:



举个非常简单的例子，比如用户按下键盘里的其中一个按键，它遵循流程是这样的：

按键按下-->输入系统驱动层-->输入系统核心层-->输入系统事件层--->用户空间

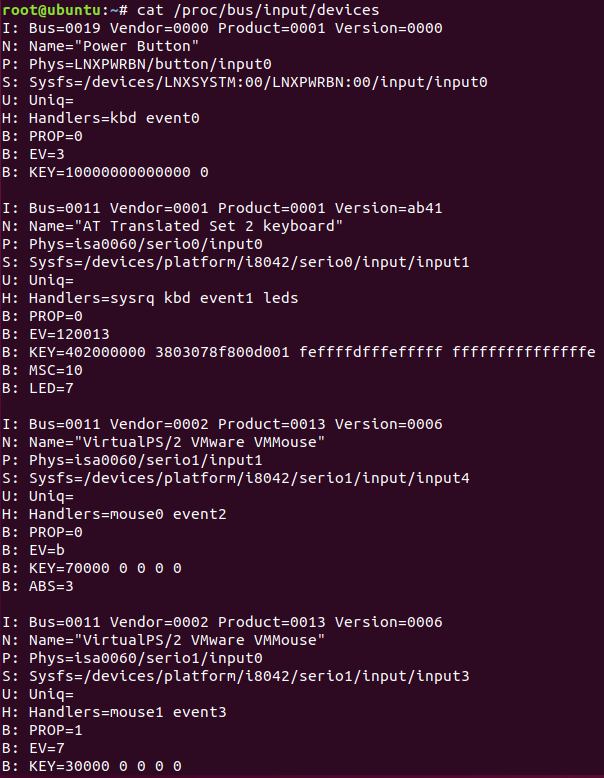
对于应用程序软件编程的角度，我们只需要关注用户空间是怎么得到按键按下以后获取的是什么事件就可以了，例如我想知道我当前按下的按是短按还是长按？或者我想知道当前我按下键盘的是空格键还是回车键等等。

### 3.3、输入系统事件的读取与分析

用户空间的设备节点那么多，怎么知道当前是哪个设备上报的呢？例如想知道键盘是由哪个设备节点上报的，就可以通过以下这条指令来获取：

cat /proc/bus/input/devices

这条指令的含义就是获取与event对应的相关设备信息，在ubuntu系统上，我们输入这个指令可以看到以下结果：



那么这里的I、N、P、S、U、H、B对应的每一行是什么含义呢？

**I:id of the device(设备ID)**

该参数由结构体struct input\_id来进行描述

41 struct input\_id {

42 //总线类型

43 \_\_u16 bustype;

44 //与厂商相关ID

45 \_\_u16 vendor;

46 //与产品相关ID

47 \_\_u16 product;

48 //版本ID

49 \_\_u16 version;

50 };

**N:name of the device**

设备名称

**P:physical path to the device in the system hierarchy**

系统层次结构中设备的物理路径。

**S:sysfs path**

位于sys文件系统的路径

**U:unique identification code for the device(if device has it)**

设备的唯一标识码

**H:list of input handles associated with the device.**

与设备关联的输入句柄列表。

**B:bitmaps(位图)**

PROP:device properties and quirks.

EV:types of events supported by the device.

KEY:keys/buttons this device has.

MSC:miscellaneous events supported by the device.

LED:leds present on the device.

PROP:设备属性和怪癖。

EV:设备支持的事件类型。

KEY:此设备具有的键/按钮。

MSC:设备支持的其他事件。

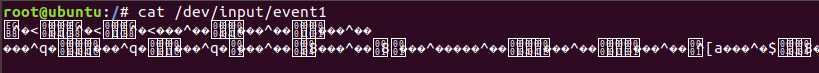
LED:设备上的指示灯。

通过了解以上参数的含义，结合以下指令

cat /proc/bus/input/devices

显示出来的信息很容易可以知道event1即是键盘上报的事件设备节点，通过读取这个event1即可获得当前用户按下的按键具体是哪个事件。

**使用cat命令来测试键盘事件**

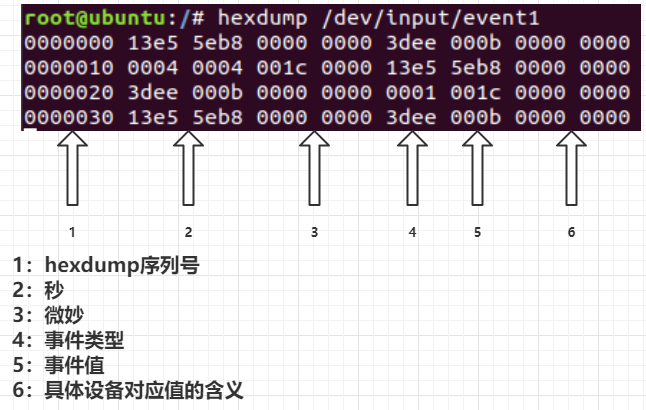


当我们在终端输入

cat /dev/input/event1

这条指令并按回车键后可以看到一堆乱码数据，这些数据我们看不懂，但是我们可以知道如果按下了按键，终端有反馈消息，这时候就知道这个事件就是我们当前操作的这个设备上报的事件，那么如何能让这些数据看得懂呢？这时候可以使用hexdump命令来读取键盘事件。

**使用hexdump命令来测试键盘事件**



这些数值是通过input\_event结构体来上报的，它位于/usr/include/linux/input.h这个头文件,input\_event结构体描述如下：

24 struct input\_event {

25 //事件发生的事件

26 struct timeval time;

27 //事件类型

28 \_\_u16 type;

29 //事件值

30 \_\_u16 code;

31 //该事件上报的数值

32 \_\_s32 value;

33 };

而input\_event结构体中的time即是：

1 struct timeval

2 {

3 \_\_time\_t tv\_sec; /\* Seconds. \*/

4 \_\_suseconds\_t tv\_usec; /\*Microseconds. \*/

5 };

其中tv\_sec为Epoch到创建struct timeval时的秒数，tv\_usec为微秒数，即秒后面的零头，Epoch的意思是指定为1970年一月一日凌晨零点零分零秒，格林威治时间。

回到input\_event结构体，事件类型type主要有以下三种，分别是：相对事件、绝对事件、键盘事件

例如：鼠标就是一个相对事件，有些情况下也有可能是绝对事件，当移动鼠标的时候，type类型也就是底层上报给用户的事件类型，那么code表示的就是相对于鼠标当前的位置的X或者Y的坐标，value则表示相对于当前的位置偏移了多少。

**事件类型(type)**

文件头文件路径：

/usr/include/linux/input-event-codes.h

当然Linux内核版本较低的有可能在以下路径的这个头文件：

/usr/include/linux/input.h

34 /\*

35 \* Event types

36 \*/

37

38 #define EV\_SYN 0x00 //同步事件

39 #define EV\_KEY 0x01 //按键事件

40 #define EV\_REL 0x02 //相对事件

41 #define EV\_ABS 0x03 //绝对事件

42 #define EV\_MSC 0x04

43 #define EV\_SW 0x05

44 #define EV\_LED 0x11

45 #define EV\_SND 0x12

46 #define EV\_REP 0x14

47 #define EV\_FF 0x15

48 #define EV\_PWR 0x16

49 #define EV\_FF\_STATUS 0x17

50 #define EV\_MAX 0x1f

51 #define EV\_CNT (EV\_MAX+1)

**事件值(code)**

由于事件值种类繁多，这里就不一一列举出来，这里举例键盘的部分事件值:

文件头文件路径：

/usr/include/linux/input-event-codes.h

当然Linux内核版本较低的有可能在以下路径的这个头文件：

/usr/include/linux/input.h

64 /\*

65 \* Keys and buttons

66 \*

67 \* Most of the keys/buttons are modeled after USB HUT 1.12

68 \* (see http://www.usb.org/developers/hidpage).

69 \* Abbreviations in the comments:

70 \* AC - Application Control

71 \* AL - Application Launch Button

72 \* SC - System Control

73 \*/

74

75 #define KEY\_RESERVED 0

76 #define KEY\_ESC 1

77 #define KEY\_1 2

78 #define KEY\_2 3

79 #define KEY\_3 4

80 #define KEY\_4 5

81 #define KEY\_5 6

82 #define KEY\_6 7

83 #define KEY\_7 8

84 #define KEY\_8 9

85 #define KEY\_9 10

86 #define KEY\_0 11

87 #define KEY\_MINUS 12

88 #define KEY\_EQUAL 13

89 #define KEY\_BACKSPACE 14

90 #define KEY\_TAB 15

91 #define KEY\_Q 16

92 #define KEY\_W 17

...

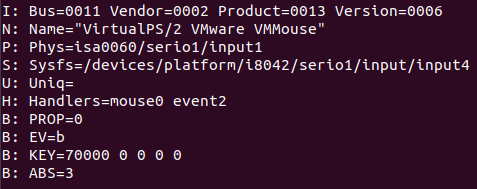
当然还有鼠标事件值、摇杆事件值、触摸屏事件值等等。

**该事件上报的数值(value)**

这部分上面已经举了鼠标的案例进行了介绍，接下来我们就通过应用程序来获取事件，后面章节将会通过鼠标、键盘以及触摸屏三个案例，进一步的了解输入系统的应用编程。

### 3.4、输入系统应用编程实战一：通用USB鼠标事件读取

根据前面章节的讲解，如果我们需要获取USB鼠标的事件，首先我们要先通过cat /proc/bus/input/devices这个指令查询与USB鼠标事件对应的相关设备信息，通过实际测试得知，event2为USB鼠标上报的事件节点。



接下来，通过hexdump命令测试一下鼠标事件的输出:



具体上报的数值是什么含义可以结合3.3章节进行分析，这里就不再进行阐述，本节的目的是编写一个获取通用USB鼠标的事件的应用程序，要获取一个事件，我们需要了解以下几个部分。

**1 设备上报事件类型(type)**

通过3.3章节，我们知道找到对应的事件类型的定义：

文件头文件路径：

/usr/include/linux/input-event-codes.h

当然Linux内核版本较低的有可能在以下路径的这个头文件：

/usr/include/linux/input.h

34 /\*

35 \* Event types

36 \*/

37

38 #define EV\_SYN 0x00 //同步事件

39 #define EV\_KEY 0x01 //按键事件

40 #define EV\_REL 0x02 //相对事件

41 #define EV\_ABS 0x03 //绝对事件

42 #define EV\_MSC 0x04

43 #define EV\_SW 0x05

44 #define EV\_LED 0x11

45 #define EV\_SND 0x12

46 #define EV\_REP 0x14

47 #define EV\_FF 0x15

48 #define EV\_PWR 0x16

49 #define EV\_FF\_STATUS 0x17

50 #define EV\_MAX 0x1f

51 #define EV\_CNT (EV\_MAX+1)

**2 设备上报的事件值(code)**

由于本节我们写的是通用USB鼠标的应用程序，所以我们找到鼠标相关的code,如下：

文件头文件路径：

/usr/include/linux/input-event-codes.h

当然Linux内核版本较低的有可能在以下路径的这个头文件：

/usr/include/linux/input.h

696 /\*

697 \* Relative axes

698 \*/

699

700 #define REL\_X 0x00 //相对X坐标

701 #define REL\_Y 0x01 //相对Y坐标

702 #define REL\_Z 0x02

703 #define REL\_RX 0x03

704 #define REL\_RY 0x04

705 #define REL\_RZ 0x05

706 #define REL\_HWHEEL 0x06

707 #define REL\_DIAL 0x07

708 #define REL\_WHEEL 0x08

709 #define REL\_MISC 0x09

710 #define REL\_MAX 0x0f

711 #define REL\_CNT (REL\_MAX+1)

在这里，我们暂时只会用来REL\_X和REL\_Y这两个参数。

那么所谓的value,就是选择具体的事件类型(type)和具体的事件值(code)以后所反应出来的值，鼠标就是相对于当前X或者相对于当前Y的值，接下来，我们来看一下如何来读取鼠标事件。

在编写input应用程序之前，在程序中需要包含以下头文件：

#include <linux/input.h>

程序编写步骤：

1 定义一个结构体变量input\_event用于描述input事件

struct input\_event event\_mouse ;

2 打开input设备的事件节点，这里我们获取的通用USB鼠标是event2

open("/dev/input/event2",O\_RDONLY);

3 读取事件

read(fd ,&event\_mouse ,sizeof(event\_mouse));

4 根据上报的事件进行处理

//判断鼠标上报的类型，可能为绝对事件，也有可能是相对事件

if(EV\_ABS == event\_mouse.type || EV\_REL == event\_mouse.type)

{

//code表示相对位移X或者Y，当判断是X时，打印X的相对位移value

//当判断是Y时，打印Y的相对位移value

if(event\_mouse.code == REL\_X)

{

printf("event\_mouse.code\_X:%d\n", event\_mouse.code);

printf("event\_mouse.value\_X:%d\n", event\_mouse.value);

}

else if(event\_mouse.code == REL\_Y)

{

printf("event\_mouse.code\_Y:%d\n", event\_mouse.code);

printf("event\_mouse.value\_Y:%d\n", event\_mouse.value);

}

}

5 关闭文件描述符

close(fd);

不难发现，获取一个输入系统事件，也是标准的文件操作，这体现了Linux一切皆文件的思想。

完整的程序案例如下：

01 #include <stdio.h>

02 #include <unistd.h>

03 #include <stdlib.h>

04 #include <fcntl.h>

05 #include <linux/input.h>

06

07 int main(void)

08 {

09 //1、定义一个结构体变量用来描述input事件

10 struct input\_event event\_mouse ;

11 //2、打开input设备的事件节点 我的通用USB鼠标事件的节点是event2

12 int fd = -1 ;

13 fd = open("/dev/input/event2", O\_RDONLY);

14 if(-1 == fd)

15 {

16 printf("open mouse event fair!\n");

17 return -1 ;

18 }

19 while(1)

20 {

21 //3、读事件

22 read(fd, &event\_mouse, sizeof(event\_mouse));

23 if(EV\_ABS == event\_mouse.type || EV\_REL == event\_mouse.type)

24 {

25 //code表示相对位移X或者Y，当判断是X时，打印X的相对位移value

26 //当判断是Y时，打印Y的相对位移value

27 if(event\_mouse.code == REL\_X)

28 {

29 printf("event\_mouse.code\_X:%d\n", event\_mouse.code);

30 printf("event\_mouse.value\_X:%d\n", event\_mouse.value);

31 }

32 else if(event\_mouse.code == REL\_Y)

33 {

34 printf("event\_mouse.code\_Y:%d\n", event\_mouse.code);

35 printf("event\_mouse.value\_Y:%d\n", event\_mouse.value);

36 }

37 }

38 }

39 close(fd);

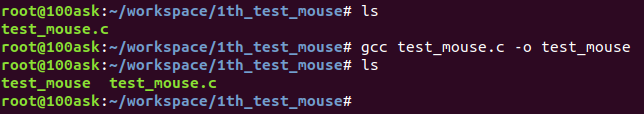
40 return 0 ;

41 }

代码编写完毕后，然后执行

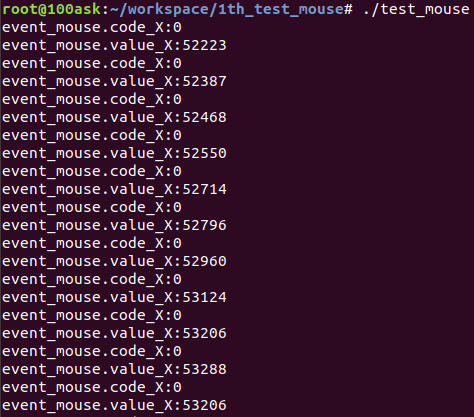
gcc test\_mouse.c -o test\_mouse

编译程序：



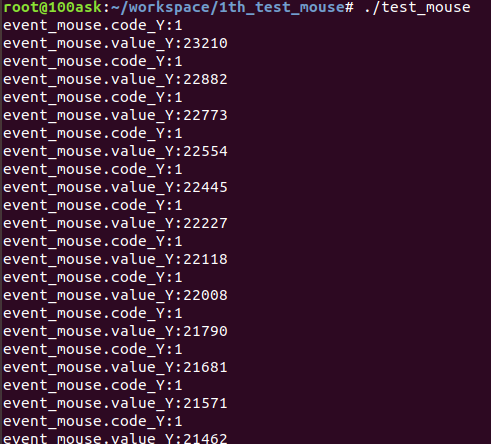
编译成功后会生成test\_mouse，接下来执行test\_mouse这个程序。

当鼠标左右移动的时候上报的事件：



这时候可以看到，只有相对于X的事件值在发生，这时候打印的value是X方向相对于原点坐标的偏移值。

当鼠标上下移动的时候上报的事件：



这时候可以看到，只有相对于Y的事件值在发生，这时候打印的value是Y方向相对于原点坐标的偏移值。

### 3.5、输入系统应用编程实战二：通用键盘事件读取

如何获取键盘事件在3.3章节已经有了相应的介绍，这里就不再写出来，本节实现的是通用键盘事件的获取，结合3.4章节获取鼠标事件的方式，这里通用键盘事件的节点为event1,通过结合3.3章节与3.4章节，编写步骤如下：

在编写input应用程序之前，在程序中需要包含以下头文件：

#include <linux/input.h>

程序编写步骤：

1 定义一个结构体变量input\_event用于描述input事件

struct input\_event event\_keyboard ;

2 打开input设备的事件节点，我的通用键盘事件的节点是event1

open("/dev/input/event1",O\_RDONLY);

3 读取事件

read(fd ,&event\_keyboard ,sizeof(event\_keyboard));

4 根据上报的事件进行处理

//判断键盘事件上报的类型

if(EV\_KEY == event\_keyboard.type)

{

if(1 == event\_keyboard.value)

printf("事件类型:%d 事件值:%d 按下\n", event\_keyboard.type, event\_keyboard.code);

else if(0 == event\_keyboard.value)

printf("事件类型:%d 事件值:%d 释放\n", event\_keyboard.type, event\_keyboard.code);

}

5 关闭文件描述符

close(fd);

完整程序案例实现如下：

01 #include <stdio.h>

02 #include <unistd.h>

03 #include <stdlib.h>

04 #include <fcntl.h>

05 #include <linux/input.h>

06

07 int main(void)

08 {

09 //1、定义一个结构体变量用来描述input事件

10 struct input\_event event\_keyboard ;

11 //2、打开input设备的事件节点 我的通用键盘事件的节点是event1

12 int fd = -1 ;

13 fd = open("/dev/input/event1", O\_RDONLY);

14 if(-1 == fd)

15 {

16 printf("open mouse event fair!\n");

17 return -1 ;

18 }

19 while(1)

20 {

21 //3、读事件

22 read(fd, &event\_keyboard, sizeof(event\_keyboard));

23 if(EV\_KEY == event\_keyboard.type)

24 {

25 if(1 == event\_keyboard.value)

26 printf("事件类型:%d 事件值:%d 按下\n",event\_keyboard.type,event\_keyboard.code);

27 else if(0 == event\_keyboard.value)

28 printf("事件类型:%d 事件值:%d 释放\n",event\_keyboard.type,event\_keyboard.code);

29 }

30 }

31 close(fd);

32 return 0 ;

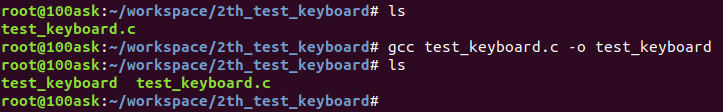
33 }

不难发现，通用USB键盘程序编写步骤与通用USB鼠标程序编写步骤几乎一样，区别只是读取的事件类型以及后面处理的数据value不同。

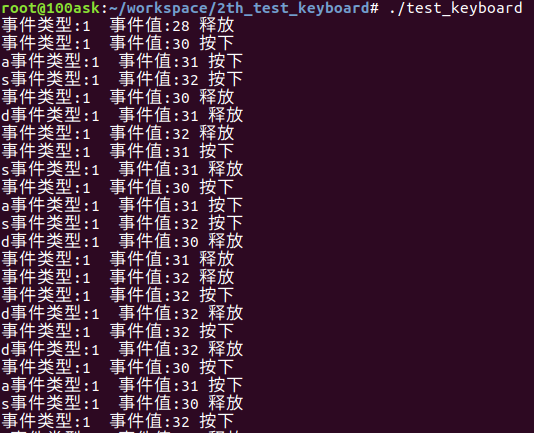
代码编写完毕后，然后执行

gcc test\_keyboard.c -o test\_keyboard

编译程序：



编译成功后会生成test\_keyboard，接下来执行test\_keyboard这个程序。



当按下按键时候，可以观察到按键的按下和释放的过程，这其实就是同一个事件下的两个不同的状态。

### 3.6、输入系统应用编程实战三：百问网imx6ul开发板触摸屏事件读取

在前面，我们已经熟悉了鼠标、键盘的基本操作，但发现一个规律，那就是编程方法类似，唯一不同的地方就是获取的事件类型以及事件值不同，那么触摸屏在input系统中是一类什么事件呢？

一般情况下，触摸屏在input系统中属于绝对事件，也就是触摸的坐标点X和Y会在屏幕的分辨率范围内上报一个绝对的坐标。

绝对事件对应的值为：EV\_ABS

相应X和Y分量的值分别为：

ABS\_MT\_POSITION\_X、ABS\_MT\_POSITION\_Y

通过结合前面的章节内容，很容易编写如下程序：

01 #include <stdio.h>

02 #include <unistd.h>

03 #include <fcntl.h>

04 #include <stdlib.h>

05 #include <linux/input.h>

06

07 int main(int argc, char \*\*argv)

08 {

09 int tp\_fd = -1 ;

10 int tp\_ret = -1 ;

11 int touch\_x,touch\_y ;

12 struct input\_event imx6ull\_ts ;

13 //1、打开触摸屏事件节点

14 tp\_fd = open("/dev/input/event1",O\_RDONLY);

15 if(tp\_fd < 0)

16 {

17 printf("open /dev/input/event1 fail!\n");

18 return -1 ;

19 }

20 while(1)

21 {

22 //2、获取触摸屏相应的事件，并打印出当前触摸的坐标

23 read(tp\_fd ,&imx6ull\_ts ,sizeof(imx6ull\_ts));

24 switch(imx6ull\_ts.type)

25 {

26 case EV\_ABS:

27 if(imx6ull\_ts.code == ABS\_MT\_POSITION\_X)

28 touch\_x = imx6ull\_ts.value ;

29 if(imx6ull\_ts.code == ABS\_MT\_POSITION\_Y)

30 touch\_y = imx6ull\_ts.value ;

31 break ;

32 defalut:

33 break ;

34 }

35 printf("touch\_x:%d touch\_y:%d\n",touch\_x,touch\_y);

36 usleep(100);

37 }

38 close(tp\_fd);

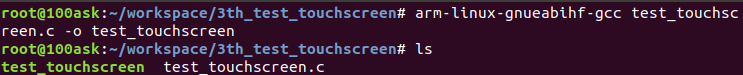
39 return 0;

40 }

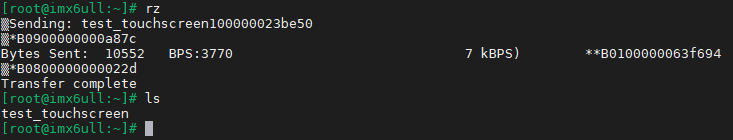
代码编写完毕后，然后执行

gcc test\_touchscreen.c -o test\_touchscreen

交叉编译程序：(注意这里是要在开发板运行，不是在PC端)

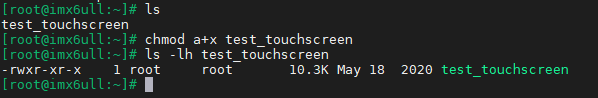


接下来启动开发板，然后串口终端输出rz命令，等待接收PC端的文件，这里我们将test\_touchscreen这个文件传输到开发板。



具体操作步骤可参考第11章：PC和开发板之间传输文件

接下来给test\_touchscreen添加可执行权限:



执行test\_touchscreen，然后用手触摸屏，可以看到有相应的坐标值打印：

