多线程

一.多线程的概念

- 什么是多线程
  线程是从进程的内存资源中切割一部分内存，提供给线程使用，线程也能够实现多任务并发执行。

  Windows系统，linux系统，还有STM32它们都是支持多任务的。

  早期的操作系统中并没有线程的概念，进程是能拥有资源和独立运行的最小单位，也是程序执行的最小单位。
  任务调度采用的是时间片轮转的抢占式调度方式，而进程是任务调度的最小单位，每个进程有各自独立的一块内存，
  使得各个进程之间内存地址相互隔离（独立）。

  后来，随着计算机的发展，对CPU的要求越来越高，进程之间的切换开销较大，
  已经无法满足越来越复杂的程序的要求了。

  为了提高系统的性能，于是就发明了线程，许多操作系统规范里引用了轻量级进程的概念，
  也就是线程。而且线程是CPU系统调度的最小单位。

二.线程与进程的区别

定义上的区别：

进程：具有一个独立功能的程序，拥有独立的内存空间。

线程：它是比进程更小的能独立运行的基本单位，线程本身是不拥有系统资源的，
但是它可以和属于同一个进程的其他线程共享进程所拥有的全部资源。

调度区别：

进程是拥有资源的基本单位。

线程是调度和分派的基本单位。

共享地址空间：

进程：拥有各自独立的地址空间、资源，所以共享复杂，需要用IPC（进程间通信），但是同步简单。

线程：共享所属进程的资源，因此共享简单，但是同步复杂，需要用加锁等措施。

占用内存和cpu：

进程：占用内存多，切换复杂，cpu利用率低；

线程：占用内存少，切换简单，cpu利用率高。

互相影响：

进程之间不会互相影响；

一个线程挂掉可能会导致整个进程挂掉。

三.基本相关API

- **创建一条新线程**

```
NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *thread,
                          const pthread_attr_t *attr,
```

```
                          void *(*start_routine) (void *),
                         void *arg);

        Compile and link with -pthread.      //编译程序时,需要连接线程库.

//参数解析
    参数1 pthread_t *thread :  新线程TID.                    保存线程的id号.(正整数)
    参数2 const pthread_attr_t *attr  : 线程属性.            一般设置为NULL,表示使用系统默认的属性.(标准属性)
    参数3 void *(*start_routine) (void *)  :  线程例程.     (函数指针) 返回值为void* 参数为 void* 与 signal函数用法一致.
                                                            创建线程需要完成的任务,靠这个函数来实现.称之为线程任务函数.
    参数4 void *arg :  传递给线程的参数.

//返回值
    成 功 : 返回0;
    失 败 : 返回errno;

#注意
    在使用多线程函数的时,需要链接我们的线程库 -pthread.
    也就是在gcc example.c -o main -pthread.
    如果没有链接,在编译时则会报错,必须要链接我们的线程库.
```

```
基本使用示例:
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

void *fun(void * arg) //线程任务函数    接口为固定的除了函数名 与 参数名可以改变,其他的不允许改变
{
   while(1)
```

```
    {
            printf("11111111\n");
            sleep(1);
    }
}
int main(int argc, char const *argv[])  //执行main函数的线程 被称为主线程
{

    pthread_t thread_id;
    //1.创建一个子线程
    pthread_create(&thread_id,    //线程的ID号
                   NULL,          //表示系统标准属性,不额外设置线程属性
                   fun,           //线程将要执行的任务函数名
                   NULL);         //不传递额外参数给线程
    while(1)
    {
            printf("2222222222\n");
            sleep(1);
    }

    return 0;
}

```

- **退出当前线程**

```
NAME
       pthread_exit - terminate calling thread

```

```
SYNOPSIS
       #include <pthread.h>

       void pthread_exit(void *retval);

参数解析:
    void *retval : 保存线程的退出值.

    配合着 pthread_join()使用.


//获取线程的ID
SYNOPSIS
       #include <pthread.h>

       pthread_t pthread_self(void);
返回值:
       成功 : 线程ID号
       这个函数是永远不会失败的.
```

- **发送线程取消请求**

```
线程的取消
SYNOPSIS
       #include <pthread.h>

       int pthread_cancel(pthread_t thread);
参数解析 :
        pthread_t thread : 需要取消的线程ID号.

```

```
返回值:
        成功 : 0;
        失败 : 错误码.
```

- **等待进程退出**

```
NAME
       pthread_join - join with a terminated thread

SYNOPSIS
       #include <pthread.h>
函数原型 :
       int pthread_join(pthread_t thread, void **retval);  //阻塞等待子线程退出

       该函数会使调用者阻塞等待，直到所指定的线程退出为止。
       该返回时系统将回收退出线程的资源，调用线程可以获得退出线程的返回值。

       Compile and link with -pthread.

参数解析 :
       pthread_t thread : 线程的ID号,(指定等待某个线程退出);
       void **retval    : 储存线程退出值的指针.(保留线程退出时所返回的数据);

返回值:
        成功 : 返回0;
        失败 : 返回errno值;
```

使用示例:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <strings.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <pthread.h>

void *fun(void * arg) //线程任务函数    接口为固定的除了函数名 与 参数名可以改变,其他的不允许改变
{

    int a = *(int *)arg;
    printf("%d \n", a);
     //不需要返回值写法
     //pthread_exit(NULL);
     //需要返回值写法
    pthread_exit("abcd");
}

int main(int argc, char const *argv[])  //执行main函数的线程 被称为主线程
```

```c
{

    int a = 100;

    pthread_t thread_id;
    //1.创建一个子线程
    pthread_create(&thread_id,    //线程的ID号
                   NULL,          //表示系统标准属性,不额外设置线程属性
                   fun,           //线程将要执行的任务函数名
                   &a);           //不传递额外参数给线程

     //只等待,不需要返回值
     //pthread_join(thread_id,NULL);

    //等待以上线程退出,并获取退出值
    void *ret;
    pthread_join(thread_id, &ret);
    printf("%s\n", (char *)ret);
    return 0;
}
```

触摸屏应用

一.输入子系统

- Linux 的硬件设备：所有的输入设备组合在一起，称为输入子系统。

  输入设备：常用的输入键盘，触摸屏，摇杆，手柄...

  输入子系统就是用来管理系统中的输入设备的.

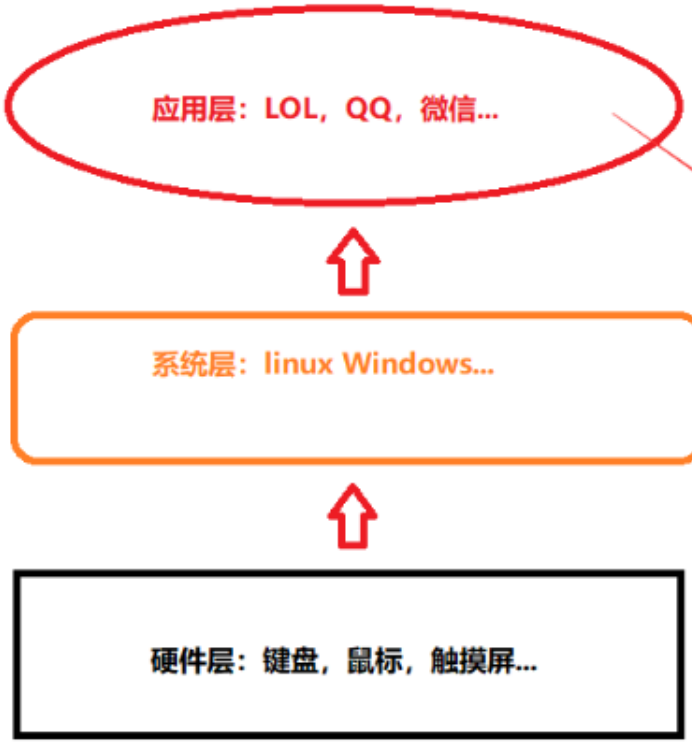
  例如：

  点击鼠标，相当于输入一个指令，触发一个事件。

  那么我们点击触摸屏或者是滑动的动作，上滑，下滑，那么我们系统会去分析你的手势动作，
  根据你的手势动作去做相对应的事情,这个事情,是你可以自己编写的。

  - 输入子系统路径

  可以通过我们linux系统中的 /usr/include/linux 中进行查看



应用层操作会触发我们的函数接口，
例如：点击屏幕

用系统提供的函数API把系统层与
应用层联系起来

系统层的API接口被调用，最终指令
会到达硬件层，去使用系统里的驱
动文件

以驱动文件把硬件层与系统层联系起来

例如：

1.我们发生了触摸动作，这时，驱动文件就会发生改变

2.我们可以通过文件IO来读取驱动文件里的信息

3.把这个信息丢到输入子系统里进行分析

通过读取到的文件改变信息，子系统进行了分析
就能知道当前发生的是什么动作

二..输入子系统模型

其实输入子系统模型就是一个结构体，我们分析一下这个结构体；

```
struct input_event
{
    struct timeval time;            //输入事件发生的时间
    __u16 type;                     //输入事件的类型        可理解为：操作了哪个硬件（是鼠标，还是键盘）            相对事件
    __u16 code;                     //输事件的子事件        可理解为具体事件 ：（以鼠标为例）滑动滚轮 左击 右击    绝对事件
    __s32 value;                    //输入事件发生后的结果
                                            //1.例如操作鼠标：滑动的距离
                                            //2.滑动触摸屏 ：  显示坐标值
};
```

- 流程思路：

  1.输入设备发生动作了（发生人为的操作）

  2.对应的驱动文件就会发生改变

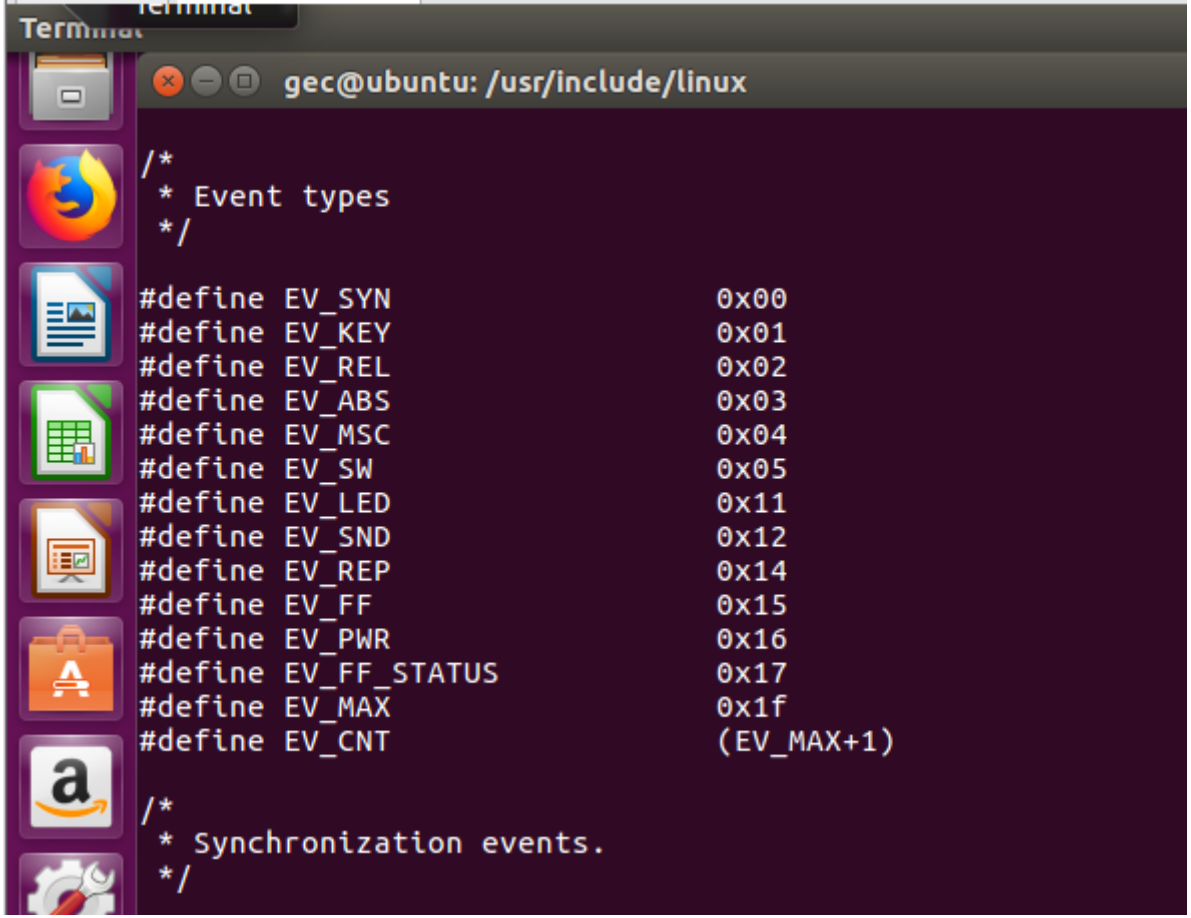
  3.通过文件IO的函数，读取驱动文件里的信息

  4.把这些文件信息丢到输入子系统中，分析当前发生的是什么事件,产生什么样的结果.

- 事件类型判断

  先进入Ubuntu /usr/include/linux 路径下 使用cat input.h查看

  使用cat input-event-codes.h 可查看具体事件信息




```

#define EV_SYN              0x00   //持续输入事件
#define EV_KEY              0x01   //按键操作/点击  键盘 触摸屏
#define EV_REL              0x02   //鼠标事件
#define EV_ABS              0x03   //触摸屏事件
```

```
/*
 * Absolute axes
 */

#define ABS_X                   0x00
#define ABS_Y                   0x01
#define ABS_Z                   0x02
#define ABS_RX                  0x03
#define ABS_RY                  0x04
#define ABS_RZ                  0x05
#define ABS_THROTTLE            0x06
#define ABS_RUDDER              0x07
#define ABS_WHEEL               0x08
#define ABS_GAS                 0x09
#define ABS_BRAKE               0x0a
#define ABS_HAT0X               0x10
#define ABS_HAT0Y               0x11
#define ABS_HAT1X               0x12
#define ABS_HAT1Y               0x13
#define ABS_HAT2X               0x14
#define ABS_HAT2Y               0x15
```

```
#define ABS_X               0x00    //x轴事件
#define ABS_Y               0x01    //y轴事件
#define ABS_PRESSURE        0x18    //压力事件    手指头按下还是抬起
#define BTN_TOUCH           0x14a   //点击事件    手指头按下还是抬起
```

- 获取触摸屏坐标的代码编写步骤

  注意 包含 输入子系统模型的头文件 #include<linux/input.h>

  1.打开触摸屏驱动文件 "/dev/input/event0"

  2.等待触摸操作，读取触摸操作信息（数据），

  3.触摸动作发生，放到输入子系统模型里

4.分析输入子系统存放的数据 就可以获取触摸屏坐标信息

5.关闭打开的文件

CRT显示中文 ： 鼠标右击端口号 点击 选项 --》 点 外观---》右侧找到 字符编码 ---》选择 UTF_8

```c
/*注意   包含 输入子系统模型的头文件   #include<linux/input.h>
  1.打开触摸屏驱动文件
    /dev/input/event0
  2.读取触摸操作数据放到输入子系统模型里
  3.分析输入子系统模型  获取触摸坐标信息
  4.关闭打开的文件 */

 #include <sys/types.h>
 #include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/input.h>
#include <stdio.h>

//获取触摸屏坐标
int get_xy()
{
   int x,y;
   //1.打开触摸屏驱动   （/dev/input/event0）
   int ts_fd = open("/dev/input/event0",O_RDWR);
   if(ts_fd<0)
   {
            printf("open ts fail\n");
            return -1;
```

```c
    }
    printf("open ts ok\n");
    //2.新建输入子系统类型的结构体
    struct input_event ts;
    // 3.循环从驱动中读取触摸数据到输入子系统里面
    //（检测是否有触摸操作发生，如果没有，代码默认是阻塞状态）
    while(1)
    {
            read(ts_fd,&ts,sizeof(ts));
            //4.通过分析输入子系统获取触摸坐标值
            if(ts.type == EV_ABS)
            {
                 if(ts.code==ABS_X)
                 {
                     x = ts.value;
                 }
                 if(ts.code==ABS_Y)
                 {
                     y = ts.value;
                 }
            }
            if(ts.type == EV_KEY && ts.code==BTN_TOUCH && ts.value == 0)
            {
                    //坐标轴效验
                    //tx = x*800/1024;
                    //ty = y*480/614;

                    printf("x %d y %d \n",x,y);
                    break;
            }
```

```
55      }
56  }
```

练习

可以持续的获取触摸屏坐标，判断现在点击的位置 在左边 还是在右边

作业

封装函数 ，实现获取触摸坐标

做按钮 启动系统

做按钮 退出系统

==========密码登陆界面==========

1.基本要求

实现密码登陆

密码最低4位

实现密码输入和删除

输入错误密码 有相应的提示重新输入

2.拓展功能

加入账号注册

同时输入正确账号和密码才能进入系统

输入账号要有数字显示