进程同步：竞争关系 协作关系

互斥量(互斥锁) ：加锁和解锁 线程在进入临界区之前，加锁操作。线程在退出临界区之后，解锁操作。如果锁是加锁状态的，则执行加锁操作将被阻塞（这把锁只能要求一个线程对其加锁） 解决竞争关系。任一时刻只能有一个线程对临界区的访问。

信号量---线程库中的信号量 条件变量 读写锁

1. 互斥锁
2. 一般将锁定义到全局

互斥锁操作：



例子：

主线程负责接收用户输入，函数线程负责将用户输入打印到终端界面

分析：

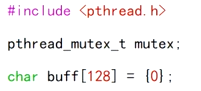
1. 数据传递：如何将主线程的数据传到函数线程————》全局空间
2. 主线程和函数线程需要同步。

使用临界资源前先进行加锁。使用完成后进行解锁。

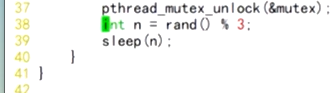
Mutex.c

首先去创建线程。定义一个全局空间char buff用于数据传递。定义一把锁。进程启动的时候首先完成锁的初始化操作。初始化的锁是解锁状态的。

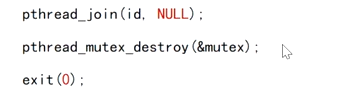
While循环，主线程接受用户输入。函数线程循环去往出去显示，打印完成过后再对buff里面的数据进行清空。



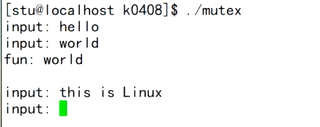








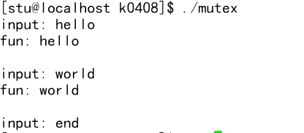
测试：



原因：程序执行是并发执行的。如果不睡眠，主线程获取用户输入，速度会非常快，lock和unlock是连着的。

改进把睡眠时间加长

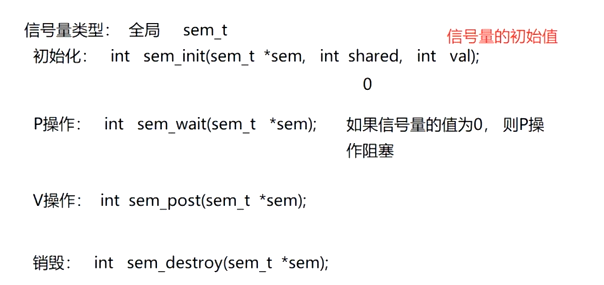




1. 信号量

特殊的一个计数器。

信号量类型：全局sem\_t（因为线程对于全局数据都是共享的）



例题：主线程负责接收用户输入，函数线程负责将用户输入打印到终端界面

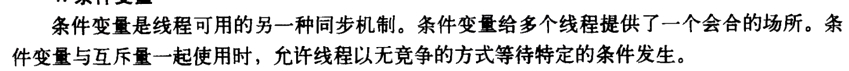
同步：使用两个信号量来完成操作，一个信号量的初值是1，另一个信号量的初值为0.

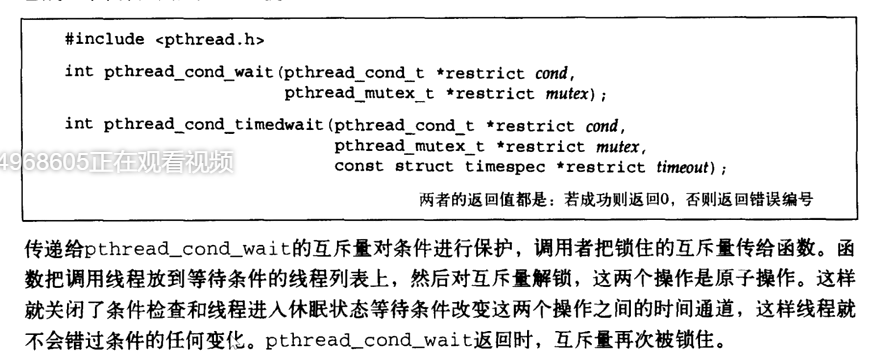
Sem1是主线程对函数线程的，sem2是函数线程对主线程的。



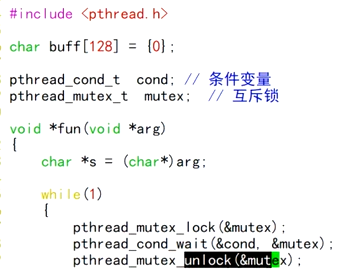
1. 条件变量《unix323》

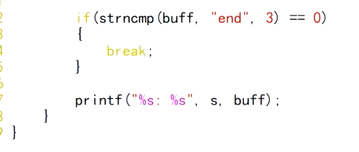
提供了一种线程间的同志机制：当某个共享数据达到某个值的时候，唤醒等待这个共享数据的线程。必须需要互斥锁来进行保护，这个互斥锁是什么东西。



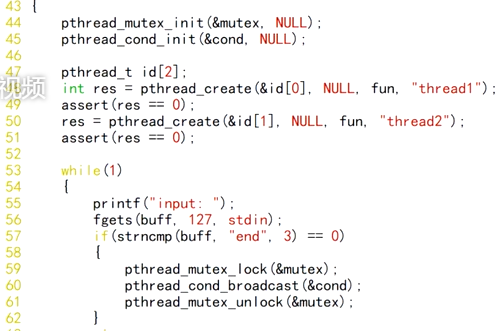


Cond.c

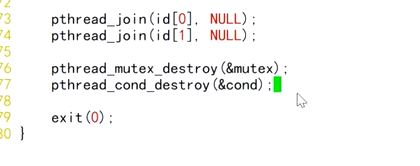




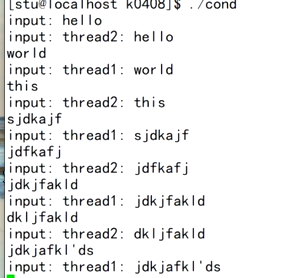
主函数







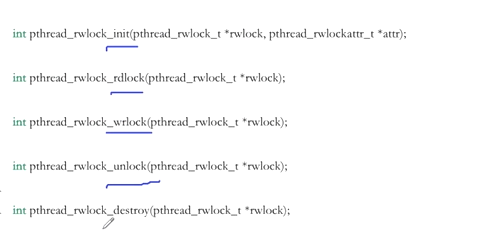
测试：



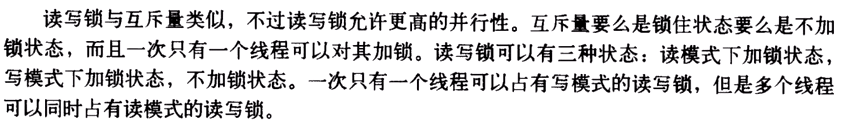
1. 读写锁

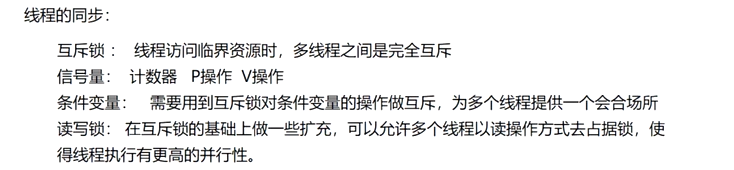
是互斥锁的进阶，更高的并行性

有初始化、读加锁、写加锁、解锁、销毁



读加锁，允许多个线程同时进行读加锁。不允许出现有读有写的情况。所有读线程都是执行读操作，不会改变数据。





读者-写者问题 哲学家就餐问题 消费者生产者问题

锁的类型：互斥锁、读写锁、自旋锁……. 死锁产生的4个必要条件，如何避免死锁

锁的实现：忙等待(中断的关闭与启用) 非忙等待（test-set）

什么是中断，什么是中断处理程序，中断处理程序的上半部分和下半部分