实验5 同步互斥

仔细阅读实验文档lab7同步互斥，完成以下练习（不做实验文档中的题目）。扩展练习选做，有能力者完成。

练习1: 了解信号量和管程的实现机制

1. 同步互斥的底层支持是如何实现的？

由于有定时器、屏蔽/使能中断、等待队列wait\_queue支持test\_and\_set\_bit等原子操作机器指令(在本次实验中没有用到)的存在,使得我们在实现进程等待、同步互斥上得到了极大的简化。定时器提供了基于时间时间的调度机制，基于时间片单位，操作系统向上提供了基于时间点的时间，并实现了基于时间长度的睡眠等待和唤醒机制。由于ucore只实现了队单处理器的支持，所以通过开、关中断的方式可以实行中断操作。等待事件的进程在转入休眠状态后插入到等待队列中。当事件发生之后,内核遍历相应等待队列,唤醒休眠的用户进程或内核线程,并设置其状态为就绪状态并将该进程从等待队列中清除。

1. 对比原理课上学到的信号量和p，v操作，说明Ucore中信号量机制的实现。

在ucore中，一个进程会由于无法满足信号量设置的某条件而在某一位置停止,直到它接收到一个特定的信号(表明条件满足了)。为了发信号,需要使用一个称作信号量的特殊变量。为通过信号量s传送信号,信号量的V操作采用进程可执行原语semSignal(s);为通过信号量s接收信号,信号量的P操作采用进程可执行原语semWait(s);如果相应的信号仍然没有发送,则进程被阻塞或睡眠,直到发送完为止。

1. Ucore中的信号量是基于信号量和条件变量实现的，请说明其中的数据结构和函数方法的设计。

信号量的数据结构定义如下:

typedef struct {

int value; //信号量的当前值

wait\_queue\_t wait\_queue; //信号量对应的等待队列

} semaphore\_t;

semaphore\_t是最基本的记录型信号量结构,包含了用于计数的整数值value,和一个进程等待队列wait\_queue,一个等待的进程会挂在此等待队列上。

在ucore中最重要的信号量操作是P操作函数down(semaphore\_t \*sem)和V操作函数up(semaphore\_t \*sem)。但这两个函数的具体实现是\_\_down(semaphore\_t \*sem, uint32\_twait\_state) 函数和\_\_up(semaphore\_t \*sem, uint32\_t wait\_state)函数。

\_\_down(semaphore\_t \*sem, uint32\_t wait\_state, timer\_t \*timer):具体实现信号量的P操作,首先关掉中断,然后判断当前信号量的value是否大于0。如果是>0,则表明可以获得信号量,故让value减一,并打开中断返回即可;如果不是>0,则表明无法获得信号量,故需要将当前的进程加入到等待队列中,并打开中断,然后运行调度器选择另外一个进程执行。如果被V操作唤醒,则把自身关联的wait从等待队列中删除(此过程需要先关中断,完成后开中断)。

\_\_up(semaphore\_t \*sem, uint32\_t wait\_state):具体实现信号量的V操作,首先关中断,如果信号量对应的wait queue中没有进程在等待,直接把信号量的value加一,然后开中断返回;如果有进程在等待且进程等待的原因是semophore设置的,则调用wakeup\_wait函数将waitqueue中等待的第一个wait删除,且把此wait关联的进程唤醒,最后开中断返回。

练习2: 了解基于信号量和管程的哲学家就餐问题

1.说明ucore中基于信号量的哲学家就餐问题的实现机制。

每位哲学家一共有三种状态，分别为思考、饥饿、进餐，当哲学家从思考中醒来则进入饥饿状态。他会试图获取餐叉，当年获取到两把叉子，则进入进餐状态，使用一个数组来跟踪每位哲学家的状态，一位哲学家只有当两个邻居都没有进餐时才允许进入进餐状态。哲学家i的两个邻居分别为left（i）和right（i），，使用一个信号量数组，每个信号量对应一位哲学家，这样就在所需叉子被占用时，想进餐的哲学家就被堵塞。

2.说明ucore中基于管程的哲学家就餐问题的实现机制。

哲学家每个人都有pickup和putdown用来在哲学家之间互相传递信息，pickup进入管程而putdown退出管程，当一个哲学家的两边都是putdown的时候将被唤醒。

3.扩展练习：了解java中同步互斥的实现机制，说明其与操作系统原理课的管程之间的关系，并用其实现写者优先的读者写者问题。

 Java中可以使用public synchronized void fun()来实现同步互斥。fun()就是一个互斥的方法，此时synchronized关键字锁定的是当前方法所属的对象。假设有p1和p2是同一个类的两个对象，当不同的线程调用访问p1的fun方法时，会产生互斥同步效果；但是不同线程调用p1的fun方法和p2的fun方法，两者无互斥同步效果。 还可以用synchronized同步程序块来实现同步互斥。和管程相似的是在p1和p2之间会有互相传递信息的fan（）函数。

程序不会…………