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？

硬件支持： 1.禁用中断，指的是在线程执行临界区代码过程中不可中断，即不会把CPU的时间片交出去，直到执行完成。这种方式在多处理器的系统中不能保证互斥，且即使在单核系统中能实现互斥，代价也是很大的。

 2.专用的机器指令  
包括比较和交换指令（CompareAndSwap，CAS）和交换指令（exchange）；  
在硬件级别上，对存储单元的访问排斥对相同单元的其他访问。基于这一点，处理器的设计者提出了一些机器指令，用于保证两个动作的原子性，如在一个取指令周期中对一个存储器单元的读和写或者读和测试。在该指令执行的过程中，其他指令访问内存将被阻止。而且这些动作在一个指令周期中完成。

软件支持：

程序设计语言机制

互斥量（基于二元信号量）

管程（monitor）（基于互斥量的结构）

java synchroized的实现基础就是monito

信号量

信号量是多线程协作的一个共享变量，是进程（线程）间传递信号的一个整数值。它有三种原子操作：初始化、递减（P）和递增（V）。

递减操作可以用来阻塞一个线程，递增操作可以用来唤醒一个线程。

管程

管程是编程语言提供的一种抽象数据结构，用于多线程互斥访问共享资源。首先，是互斥访问，即任一时刻只有一个线程在执行管程代码；第二，正在管程内的线程可以放弃对管程的控制权，等待某些条件发生再继续执行。

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。

\_\_up(semaphore\_t \*sem, uint32\_t wait\_state)

  对照信号量的原理性描述和具体实现，可以发现二者在流程上基本一致，只是具体实现采用了关中断的方式保证了对共享资源的互斥访问，通过等待队列让无法获得信号量的进程睡眠等待。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

\_\_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）

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。

（1）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;

（2）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

\_\_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）。

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。

在进行练习1之前需要对先前LAB中编码的内容进行更新，主要更新为在处理时钟中断的时候，将sched\_class\_proc\_tick函数修改为run\_timer\_list函数（由于后者中已经包括了前者），用于支持定时器机制;

在完成先前实验的代码的更新之后，就能够完成基于信号量的哲学家就餐问题了，因此不妨先查看分析内核级信号量的实现：

实现了内核级信号量机制的函数均定义在sem.c中，因此不妨对这些函数进行分析：

sem\_init: 对信号量进行初始化的函数，根据在原理课上学习到的内容，信号量包括了等待队列和一个整型数值变量，该函数只需要将该变量设置为指定的初始值，并且将等待队列初始化即可；

\_\_up: 对应到了原理课中提及到的V操作，表示释放了一个该信号量对应的资源，如果有等待在了这个信号量上的进程，则将其唤醒执行；结合函数的具体实现可以看到其采用了禁用中断的方式来保证操作的原子性，函数中操作的具体流程为：

查询等待队列是否为空，如果是空的话，给整型变量加1；

如果等待队列非空，取出其中的一个进程唤醒；

\_\_down: 同样对应到了原理课中提及的P操作，表示请求一个该信号量对应的资源，同样采用了禁用中断的方式来保证原子性，具体流程为：

查询整型变量来了解是否存在多余的可分配的资源，是的话取出资源（整型变量减1），之后当前进程便可以正常进行；

如果没有可用的资源，整型变量不是正数，当前进程的资源需求得不到满足，因此将其状态改为SLEEPING态，然后将其挂到对应信号量的等待队列中，调用schedule函数来让出CPU，在资源得到满足，重新被唤醒之后，将自身从等待队列上删除掉；

up, down：对\_\_up, \_\_down函数的简单封装；

try\_down: 不进入等待队列的P操作，即时是获取资源失败也不会堵塞当前进程

1. 说明ucore中基于管程的哲学家就餐问题的实现机制。

关于使用条件变量来完成哲学家就餐问题的实现中，总共有两个关键函数，以及使用到了N（哲学家数量）个条件变量，在管程中，还包括了一个限制管程访问的锁还有N个用于描述哲学家状态的变量（总共有EATING, THINKING, HUNGER）三种状态；

首先分析phi\_take\_forks\_condvar函数的实现，该函数表示指定的哲学家尝试获得自己所需要进餐的两把叉子，如果不能获得则阻塞，具体实现流程为：

给管程上锁；

将哲学家的状态修改为HUNGER；

判断当前哲学家是否有足够的资源进行就餐（相邻的哲学家是否正在进餐）；

如果能够进餐，将自己的状态修改成EATING，然后释放锁，离开管程即可；

如果不能进餐，等待在自己对应的条件变量上，等待相邻的哲学家释放资源的时候将自己唤醒；

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。