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？

答：

1.进程获取临界资源之前，要先获取信号量资源；  
若无信号量资源，则该进程阻塞等待，进入等待队列。  
若有信号量资源，则对信号量进行P（-1）操作，再获取临界资源。  
当临界资源+1时，对应的信号量资源则执行V（+1）操作，然后唤醒在等待队列中等待获取临界资源的进程。

2.一个进程获取了该临界资源之后，另一个进程无法再访问该临界资源。

实现互斥，采用一元信号量，即：该信号量的计数器，只能为0或1。

一个进程要获取临界资源时，先获取对应的信号量资源；

当无信号量资源时，则该进程阻塞等待，进入等待队列。

当有信号量资源时，则对该信号量资源进行P（-1）操作，然后获取该临界资源。

当该进程使用完临界资源时，将释放信号量资源（对信号量资源进行V（+1）操作），然后唤醒等待队列中的进程。

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。

答：内核级信号量的实现主要包含信号量数据结构semaphore\_t和实现P操作的函数down以及实现V操作的函数up。

semaphore\_t：信号量数据结构。value是一个计数器，wait\_queue是等待队列。

down：完成了信号量中的P操作。该函数主要调用了\_\_down函数。\_\_down函数中，首先关掉中断，然后判断信号量的value值是否大于0，如果大于0说明资源未被占用，则将value值减一并退出。若value值小于或等于0，则说明资源已经被占用，因此该进程需要等待。将该进程加入到等待队列中，开中断，然后进行调度。如果之后被V操作唤醒，则先关中断，将该进程从等待队列中删除，再开中断。  
up：完成了信号量中的V操作。该函数主要调用了\_\_up函数。在\_\_up中，首先关中断，如果当前等待队列为空则直接将value值加一，否则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。

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。

答：首先定义条件变量的结构体。其中需要一个计数器count来记录等待的进程数和一个等待队列wait\_queue,接下来完成条件变量的wait操作。wait操作之前首先要关中断以保证其原子性。随后判断count是否为0，若为0则表明没有进程在占用该资源，直接使用即可；否则将自身挂起等待别的进程唤醒。条件变量的signal操作同样需要先关中断，然后唤醒等待列表上的第一个进程。

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。

答：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。

答：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；

1.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：

2.给管程上锁；

3.将哲学家的状态修改为HUNGER；

4.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；

而phi\_put\_forks\_condvar函数则是释放当前哲学家占用的叉子，并且唤醒相邻的因为得不到资源而进入等待的哲学家：

1.首先获取管程的锁；

2.将自己的状态修改成THINKING；

3.检查相邻的哲学家是否在自己释放了叉子的占用之后满足了进餐的条件，如果满足，将其从等待中唤醒（使用cond\_signal）；

4.释放锁，离开管程；

由于限制了管程中在访问共享变量的时候处于RUNNABLE的进程只有一个，因此对进程的访问是互斥的；并且由于每个哲学家只可能占有所有需要的资源（叉子）或者干脆不占用资源，因此不会出现部分占有资源的现象，从而避免了死锁的产生；

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。