**实验5 同步互斥**

一、实验要求

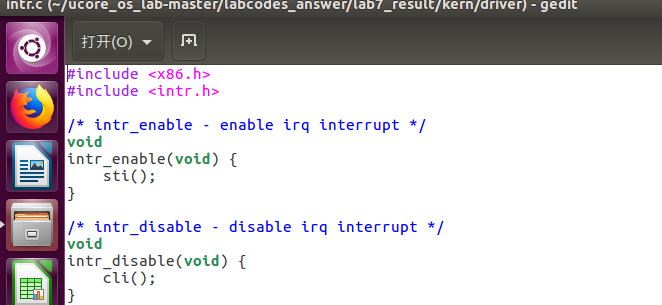
仔细阅读实验文档lab7同步互斥，完成以下练习（不做实验文档中的题目）。扩展练习选做，有能力者完成。

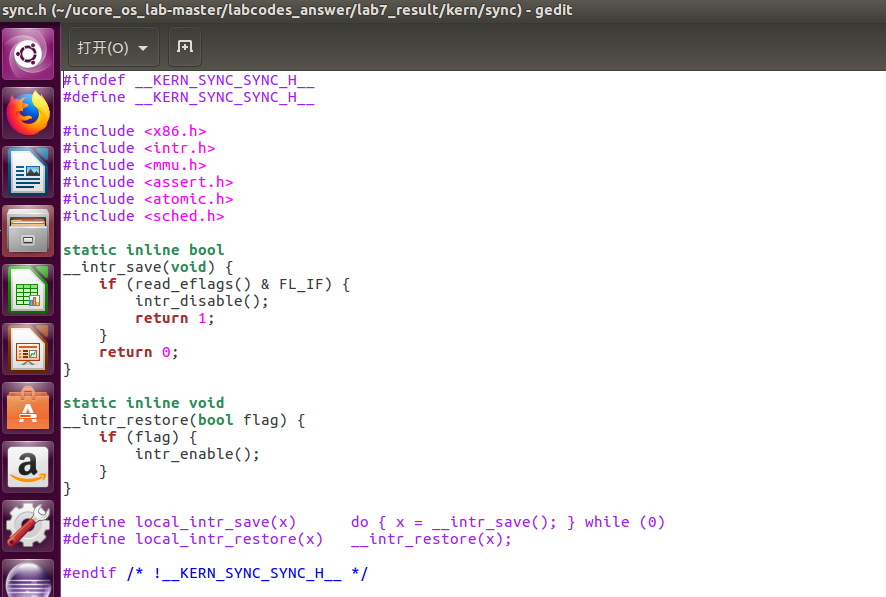
二、实验内容

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. 开关中断：





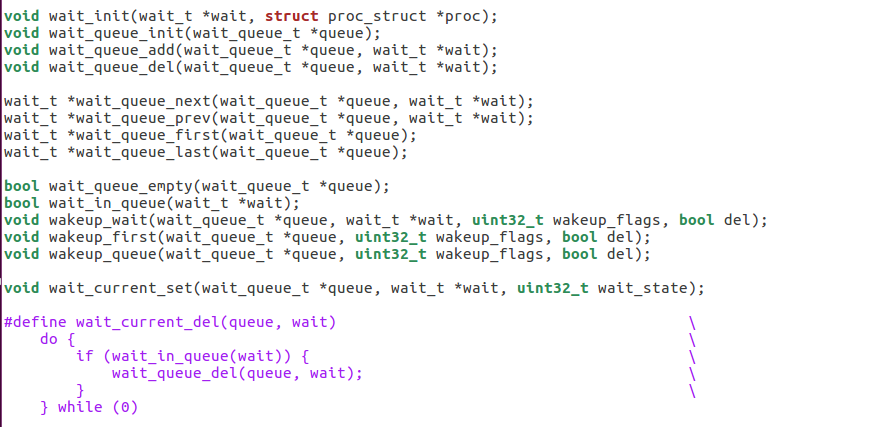
在ucore中提供的底层机制包括中断开关控制和test\_and\_set相关原子操作机器指令。kern/sync.c中实现的开关中断的控制函数local\_intr\_save(x)和local\_intr\_restore(x)，它们是基于kern/driver文件下的intr\_enable()、intr\_disable()函数实现的。具体调用关系为：

关中断：local\_intr\_save --> \_\_intr\_save --> intr\_disable --> cli

开中断：local\_intr\_restore--> \_\_intr\_restore --> intr\_enable --> sti

最终的cli和sti是x86的机器指令，最终实现了关中断和开中断，即设置了eflags寄存器中与中断相关的位。通过关闭中断，可以防止对当前执行的控制流被其他中断事件处理所打断。既然不能中断，那也就意味着在内核运行的当前进程无法被打断或被从新调度，即实现了对临界区的互斥操作。所以在单处理器情况下，可以通过开关中断实现对临界区的互斥保护。开关中断只对单处理器下的互斥操作起作用。

1. 等待队列：



用户进程或内核线程可以转入休眠状态以等待某个特定事件，当该事件发生时这些进程能够被再次唤醒。内核实现这一功能的一个底层支撑机制就是等待队列（wait queue），等待队列和每一个事件联系起来。需要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（runnable state），并将该进程从等待队列中清除。ucore在kern/sync/{ wait.h, wait.c }中实现了wait结构和wait queue结构以及相关函数），这是实现ucore中的信号量机制和条件变量机制的基础，进入wait queue的进程会被设为睡眠状态，直到他们被唤醒。

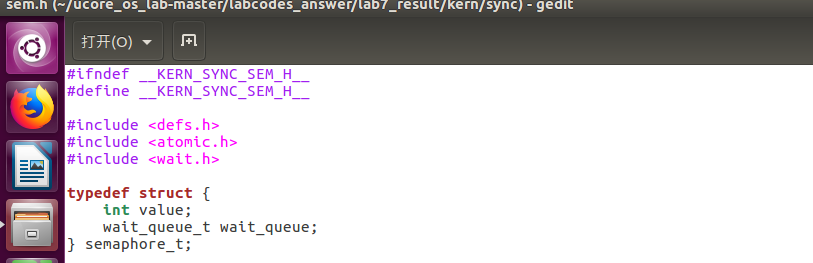
1. 对比原理课上学到的信号量和p，v操作，说明Ucore中信号量机制的实现。

答：当多个进程可以进行互斥或同步合作时，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。

答：

信号量的数据结构为：



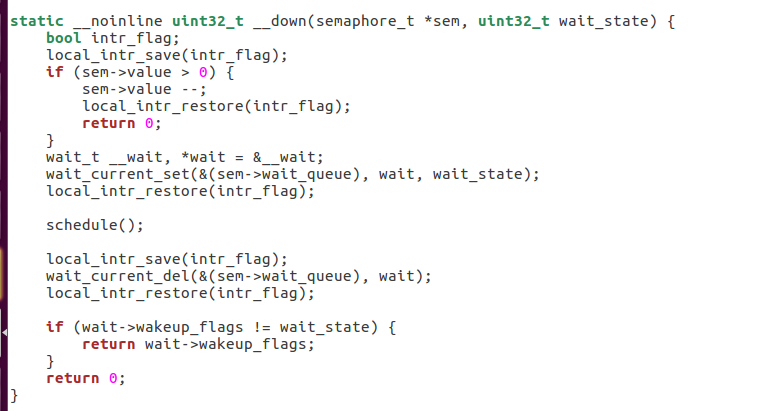
value为当前信号量的值

wait\_queue\_t wait\_queue;为信号量对应的等待序列

semaphore\_t是最基本的记录型信号量（record semaphore)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。

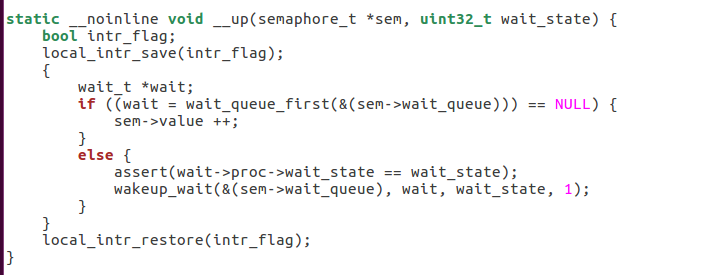
在ucore中最重要的信号量操作是P操作函数down(semaphore\_t \*sem)和V操作函数 up(semaphore\_t \*sem)。但这两个函数的具体实现是\_\_down(semaphore\_t \*sem, uint32\_t wait\_state) 函数和\_\_up(semaphore\_t \*sem, uint32\_t wait\_state)函数，二者的具体实现描述如下：

Down的部分：



\_\_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的部分：

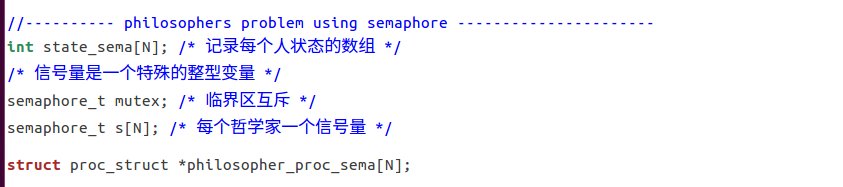


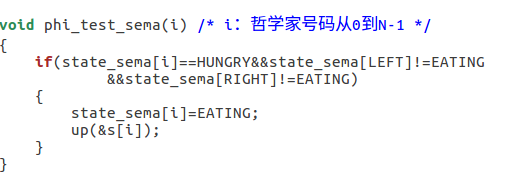
\_\_up(semaphore\_t \*sem, uint32\_t wait\_state)：具体实现信号量的V操作，首先关中断，如果信号量对应的wait queue中没有进程在等待，直接把信号量的value加一，然后开中断返回；如果有进程在等待且进程等待的原因是semophore设置的，则调用wakeup\_wait函数将waitqueue中等待的第一个wait删除，且把此wait关联的进程唤醒，最后开中断返回。

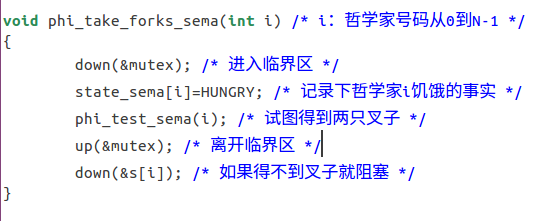
练习2: 了解基于信号量和管程的哲学家就餐问题

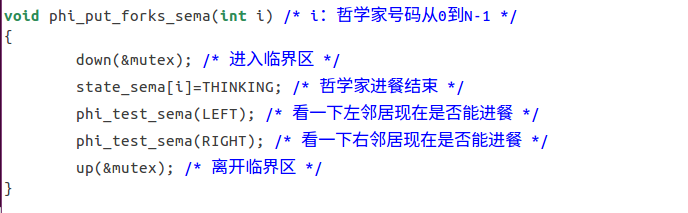
1. 说明ucore中基于信号量的哲学家就餐问题的实现机制。

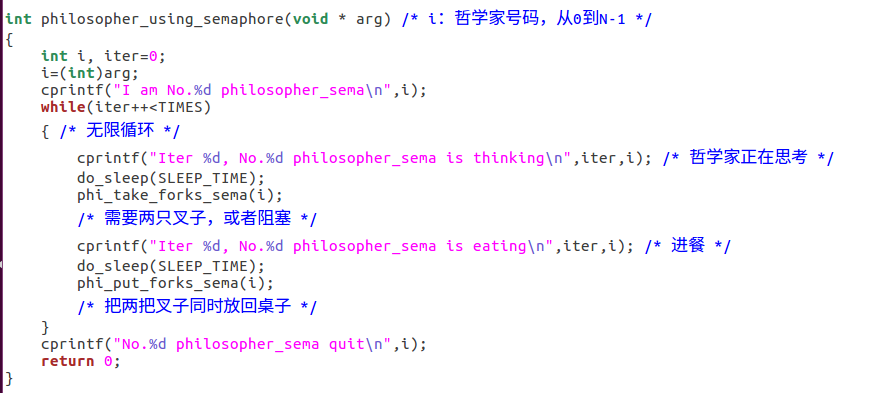
答：











1. 说明ucore中基于管程的哲学家就餐问题的实现机制。

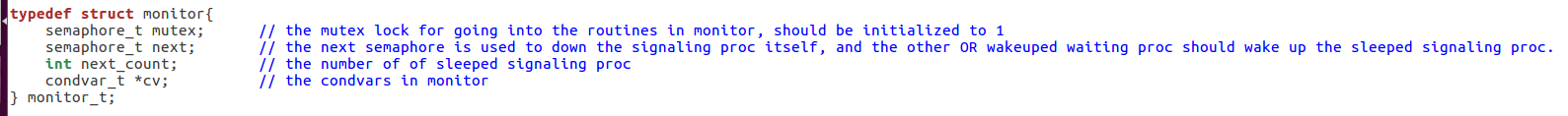
答：

实现流程为：

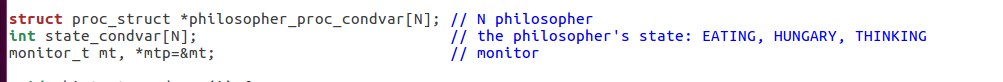
-->给管程上锁；

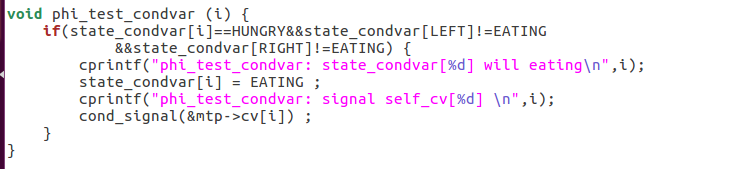
-->将哲学家的状态修改为HUNGER；

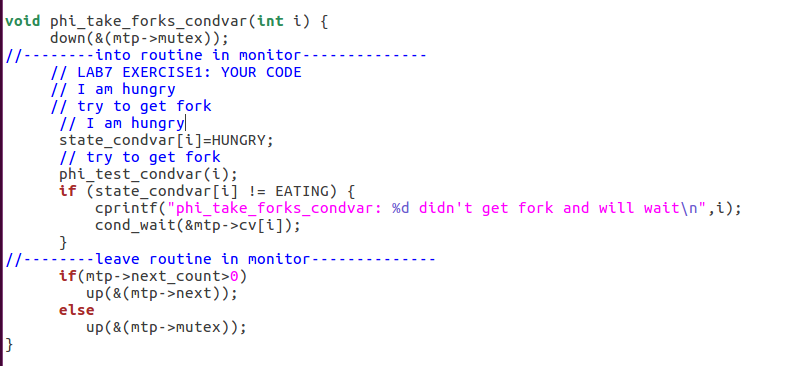
-->判断当前哲学家是否有足够的资源进行就餐（相邻的哲学家是否正在进餐）；-->如果能够进餐，将自己的状态修改成EATING，然后释放锁，离开管程即可；-->如果不能进餐，等待在自己对应的条件变量上，等待相邻的哲学家释放资源的时候将自己唤醒；



管程的数据结构monitor







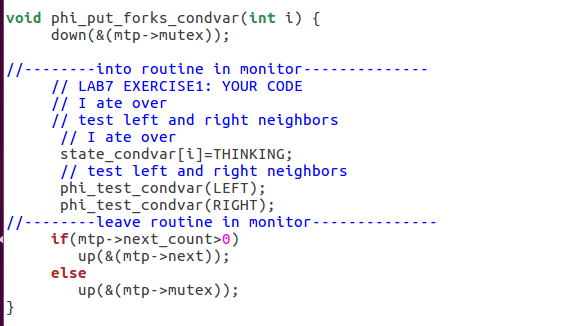
其中：

state\_condvar[i]=HUNGRY; //记录下哲学家i饥饿的事实

cond\_wait(&mtp->cv[i]);//如果得不到叉子就阻塞

if(mtp->next\_count>0)//如果阻塞则唤醒

up(&(mtp->mutex));//离开临界区



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。