同步互斥

**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？**

互斥的实现有两个方面：硬件支持和软件设计

硬件支持：

1. 禁用中断，即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和交换指令在硬件级别上，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。

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**。

信号量是一种同步互斥机制的实现，普遍存在于现在的各种操作系统内核里。相对于spinlock的应用对象，信号量的应用对象是在临界区中运行的时间较长的进程。等待信号量的进程需要睡眠来减少占用CPU的开销。

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*。***

关于条件变量机制的实现主要位于monitor.c文件中的cond\_signal, cond\_wait两个函数中，这两个函数的含义分别表示提醒等待在这个条件变量上的进程恢复执行，以及等待在这个条件变量上，直到有其他进行将其唤醒位置；下文将对这两个函数的具体实现进行分析：

cond\_signal: 将指定条件变量上等待队列中的一个线程进行唤醒，并且将控制权转交给这个进程；具体执行流程为：

判断当前的条件变量的等待队列上是否有正在等待的进程，如果没有则不需要进行任何操作；

如果由正在等待的进程，则将其中的一个唤醒，这里的等待队列是使用了一个信号量来进行实现的，由于信号量中已经包括了对等待队列的操作，因此要进行唤醒只需要对信号量执行up操作即可；

**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。**

将使用信号量来实现条件变量和管程中使用的锁和等待队列即可；

1. **说明ucore中基于管程的哲学家就餐问题的实现机制**

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：

a．给管程上锁；

b将哲学家的状态修改为HUNGER；

c.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）；

d.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；

e.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唤醒；

**代码实现**

down(&(mtp->mutex)); // 获取管程的锁

state\_condvar[i] = HUNGRY; // 将自己设置为饥饿

if (state\_condvar[(i + 4) % 5] != EATING && state\_condvar[(i + 1) % 5] != EATING) { // 判断当前叉子是否足够就餐

state\_condvar[i] = EATING; // 就餐

} else {

cprintf("phi\_take\_forks\_condvar: %d didn’t get fork and will wait\n", i);

cond\_wait(mtp->cv + i); // 等待其他人释放资源

}

if(mtp->next\_count>0) // 释放管程的锁

up(&(mtp->next));

else

up(&(mtp->mutex));

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：

首先获取管程的锁；

将自己的状态修改成THINKING；

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）；

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。