**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？**
2. **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。**
3. **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：

（1）：

禁用中断方法：引入临界区、进入区、退出区等概念，空闲则入：没有进程在临界区时任何进程可以进入；忙则等待：有进程在临界区时，其他进程均不能进入临界区；有限等待：等待进入临界区的进程不能无限等待下去；让权等待：不能进入临界区的进程，应释放CPU

软件方法:Pererson算法：使用锁来控制临界区的访问

（2）：

当多个（>1）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

（3）：

管程的数据结构monitor\_t如下：

typedef struct monitor{ // 二值信号量，只允许一个进程进入管程，初始化为1

semaphore\_t mutex; //用于进程同步操作的信号量

semaphore\_t next; // 睡眠的进程数量

int next\_count; // 条件变量cv

condvar\_t \*cv;

} monitor\_t;

条件变量condvar\_t的数据结构如下：

typedef struct condvar{

semaphore\_t sem; // 在这个条件变量上的睡眠进程的个数

int count; // 此条件变量的宿主管程

monitor\_t \* owner;

} condvar\_t;

**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。**
2. **说明ucore中基于管程的哲学家就餐问题的实现机制。**

答：

（2）：

Semaphore phis[5];

Semaphore mutex(1);

int state[5];

for(phi : phis)

phi.sem = 0;

void check(i){

if(state[i] == HUNGRY && (state[LEFT] != EATING && state[RIGHT] != EATING)){

state[i] = EATING;

phis[i].V();

}

}

void philosopher(){

while(true){

thinking();

state[i] = HUNGRY

mutex.P();

check(i);

mutex.V();

phis[i].P();

eating();

state[i] = THINKING;

mutex.P();

check(LEFT);

check(RIGHT);

mutex.V();

}

}

（2）：

用三种不同状态表示哲学家的活动：进餐、饥饿、思考。（thinking，hungry，eating）

为第i个(i=0….4)哲学家设置一个条件变量self[i]，当哲学家饥饿又不能获得筷子时，用self[i].wait来阻塞自己。

管程设置三个函数：pickup取筷子，putdown放筷子，test测试是否具备进餐条件。

 Monitor DP;

(thinking, hungry, eating) state[5] ;

condition  self[5];

void Entry pickup(int i)  / \*第i号哲学家取筷子\*/

{  state[i] =hungry;    test(i);

if(state[i]=eating) self[i].wait;

}

void Entry putdown(int i)  /\*第i号哲学家放下筷子\*/

{  state[i] = thinking;

  test((i+4)%5);

test((i+1)%5);

}

void test(int i)    /\*测试第i号哲学家是否具备进餐条件\*/

{

if(state[(i+4)%5]<>eating)%%(state[i]=hungry)%%(state[(i+1)%5]<>eating)

{

state[i]=eating;

self[i].signal;

}

｛

For（ i=0;i<5;i++）

State[i]=thinking;

}

Cobegin

Void philosopher(int i)

{

While(true){

Thinking;

DP.pickup(i);

Eating;

DP.putdown(i);

}

}

Coend;