第六章 同步

1. Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

2. Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

3. The Sleeping-Barber Problem. A barbershop consists of a waiting room with n chairs and abarber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and the customers.

1. 因为自旋锁的核心特点是：线程获取锁失败时，会持续循环检查锁是否可用，而进入休眠。

单处理器系统中不合适的原因：单处理器同一时间只能运行一个线程，假设线程A持有自旋锁但暂时无法释放，此时线程B尝试获取该锁，就会进入"自旋"，由于CPU被线程B霸占，持有锁的线程A无法获得CPU时间来释放锁，从而导致死锁

多处理器系统：是有多个CPU核心，在B进入"自旋"时，A可以通过另一个CPU核心来执行操作从而释放锁。

2. 假设信号量S（初始值为1），两个进程P1，P2，假设wait(), signal()不是原子操作

wait()操作分两步执行 ① 检查S的值，判断S≥1 ② S--

演示情况：

一. P1执行wait()，执行① 发现S=1，突然CPU切换到P2进程，

而P1的wait()的② 还没有执行

二. P2执行wait()，执行①，检查到S=1，接着执行②，把S减成0，然后P2进入临界区。

三. CPU切换到P1，P1继续执行②，由于①已经确认过，S减成-1，然后P1也进入临界区。

结果是互斥性出现违反。

如果 signal( ) 也不是原子操作会出现为 signal( ) 分两步：

步骤1：将信号量加1

步骤2：唤醒等待队列

P1 刚执行"加1"操作，就被调度走，P2 执行 wait(S)，由于S=1 进入临界区，之后P1 继续执行"唤醒 思辑"，但此时 P2已经在临界区，P3 执行 wait(S) 可能错误进入。

3.

```
semaphore   customers=0;
semaphore   barber=0;
semaphore   mutex=1;
int     chairs=n;
int   waiting=0;
```

```
customers(){
  while(1){
    P(mutex);
    if(waiting<chairs){
      waiting++;
      V(customers);
      V(mutex);
      P(barber);
    //等待理发;
    } else   V(mutex);
```

```
barber(){
  while(1){
    P(customers);
    P(mutex);
    waiting--;
    V(barber)
    V(mutex)
  //理发;
  }
}
```

}

}

4．有一个系统，定义 P、V 操作如下：

P(s):

s:=s-1;

if s<0 then

将本进程插入相应队列末尾等待；

V(s):

s:=s+1;

if s<=0 then

从相应等待队列队尾唤醒一个进程，将其插入就绪队列；

问题：

(1) 这样定义 P、V 操作是否有问题?

(2) 用这样的 P、V 操作实现 N 个进程竞争使用某一共享变量的互斥机制.

(1) 没有问题

P 操作：尝试"申请"资源（先让 s 减 1，如果 s<0（资源不够），就把当前进程放在等待队列。

V 操作：释放资源（s++）如果有进程因为资源不够而等待，就从等待队列里唤醒一个进程。

这个可以实现进程的互斥。

(2) 设定一个信号量 S ，S=1

任意一个进程

```
while (1) {
    P(s);
    // 开始访问临界区的代码
    V(s);
}
```

① P(S) 当前没有进程在访问共享变量（s初始为1）

P(S)后 S=0；第二个进程再执行 P(S)，S=-1

进入等待队列程，直至 S= -n+1

② 其中一个进程执行完到 V(S)，s++表示资源空闲出来在等待队列可以有一个进程进行访问。S 的值并不是代表当前是否可以访问共享变量，而是表示有多少进程在等待访问共享变量。

5. P、V 操作实现第二类读者写者问题：写者优先。

具体条件如下：

多个读者可以同时进行读；

写者必须互斥（只允许一个写者写，也不能读者写者同时进行）；

写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）。

```
semaphore r_mutex=1;    // 保护 read_count
semaphore w_mutex=1;    // 保护对 write_wait
semaphore resource=1;   // 保证写操作互斥
semaphore readtry=1;    // 读者访问

int read_count=0;
int write_count=0
```

```
reader(){
    P(readtry);
    P(r_mutex);
    readcount++
    if(readcount==1){
        P(resource);
    }
    V(readtry)
    //读操作
    P(r_mutex)
    readcount--;
```

```
writer(){
    P(w_mutex);
    write_count++;
    if(write_count==1){
        P(readtry);
    }
    V(w_mutex);
    P(resource);
    //写操作
    V(resource);
    P(w_mutex);
```

```
if(readcount ==0) {
        V(resource);
}
V(r_mutex);
}
```

```
write_count--;
if(writecount ==0){
    V(readtry);
}
V(w_mutex);
}
```

6. 把学生和监考老师都看作进程，学生有 N 人，教师 1 人。考场门口每次只能进出一个人，进考场原则是先来先进。当 N 个学生都进入考场后，教师才能发卷子。学生交卷后可以离开考场。教师要等收上来全部卷子并封装卷子后才能离开考场。

问题:

(1) 问共需设置几个进程?

(2) 试用 P、V 操作解决上述问题中的同步和互斥关系。

进出

(1) 一共需要 N+1 进程，N个学生进程，1个教师进程

(2)
```
semaphore mutex=1; // 用于门口访问的互斥
student_arrival=0; // 记录学生进入考场的人数
teacher_ready=0; // 用于让学生等待教师发卷子
all_finished=0; // 统计学生交卷数量
```

```
student(){
    P(mutex);
    // 进入考场;
    V(mutex);
    V(student_arrival);
```

(n)

```
teacher(){
    for(i=1;i<=n;i++){
        P(student_arrival);
    }
    for(i=1;i<=n;i++){
```

```
P(teacher_ready)
//考试，交卷
V(all_finished)
P(mutex);
//离开考场
V(mutex);
```

```
V(teacher_ready);//为每个学生发
}                //卷子
//等待所有学生交卷
for(i=1; i<=n; i++){
    P(all_finished);
}
P(mutex);
//离开考场
V(mutex);
```