OS 第三次作业

6.9

考虑如何使用原子硬件实现互斥锁。假设定义互斥锁的结构如下：

```
typedef struct { int available; } lock;
```

当 available 为 0 时，表示锁可用；当 available 为 1 时，表示锁不可用。通过这个 struct，说明如何采用指令 test_and_set() 和 compare_and_swap() 来实现如下函数：

- `void acquire(lock *mutex)`
- `void release(lock *mutex)`

一定要包括任何可能必要的初始化。

初始化：

```
available = 0;
```

解答：

```
void acquire(lock *mutex)
{
    while(test_and_set(&(mutex->available)));
}

void release(lock *mutex)
{
    compare_and_swap(&(mutex->available), 1, 0);
}
```

6.10

6.5 节所述的互斥锁实现存在忙等待的问题。请讨论一下：通过什么样的必要修改，以便等待获取互斥锁的进程应阻塞，并添加到等待队列，知道锁可用位置。

解答：

必要修改：创建一个进程链表；当一个进程尝试获取互斥锁时，若锁可用，则获取锁，若不可用，将该进程添加到进程链表中进入阻塞状态；当一个进程释放锁时，先检查进程链表是否为空，若为空，

则直接释放，若不为空，则唤醒链表末尾的进程，然后释放。

6.11

假设一个系统有多个处理核。针对下面的各个场景，请讨论一下：哪一个是更好的加锁机制，是自旋锁？还是互斥锁？这里要求等待进程在等待可用锁时应睡眠。

- 用锁的时间很短。
- 用锁的时间很长。
- 线程在拥有锁时可能处于睡眠。

解答

- 用锁的时间很短：用自旋锁，等待锁时没有上下文切换，开销小。
- 用锁的时间很长：用互斥锁，若进程自旋会浪费大量CPU开销。
- 线程在拥有锁时可能处于睡眠：互斥锁。

6.19

就能用于实现同一类型的同步问题的解决方案而言，管程和信号量是等价的。请证明。

解答

所谓等价指的是用管程能够实现信号量,也能用信号量实现。
用信号量实现管程：课本已实现。

用管程实现信号量：

```
monitor mutex
{
    condition x;
    int v;
    void P()
    {
        if(v <= 0)
            x.wait();
        v--;
    }
    void V()
    {
        v++;
        x.signal();
    }
}
```

6

进程get把从读卡机读取的数据放进 buffer1；进程 copy 把 buffer1 的信息处理后放到 buffer2；进程 put 从 buffer2 中取数并打印输出。请用 PV 操作协调上述 3 个进程的同步关系。

设置信号量 S1=1, S2=1, empty1=1, empty2=1, full1=0, full2=0。

解答

进程 get：

```
do {
    P(S1);
    P(empty1);
    从读卡机读取数据放入buffer1;
    P(V1);
    V(full1);
} while(true);
```

进程 copy：

```
do {
    P(full1);
    P(S1);
    处理buffer1的信息；
    V(S1);
    V(empty1);
    P(empty2);
    P(S2);
    将信息放入buffer2;
    P(V2);
    V(full2);

} while(true);
```

进程 put：

```
do {
    P(full2);
    P(S2);
    从buffer2取数据打印；
    P(V2);
    V(empty2);
} while(true);
```

7.1

考虑如图 7 - 10 所示的交通死锁。
a. 证明这个例子实际包括死锁发生的 4 个必要条件。
b. 给出一个简单规则，以便避免系统死锁。

解答

a. 死锁的四个必要条件：

1. 互斥：车俩占的位置不能同时有两辆车占有。
2. 请求与保持：一辆车占有自己位置的同时想要到达前面其他车辆的位置。
3. 不可剥夺：一辆车的位置不能被其他车抢夺，否则会撞车。
4. 环路等待：图内的每辆车都在申请前一辆车的位置，并形成了环路，造成了环路等待。

b. 简单规则：加入红绿灯，使得十字路口在红灯时只允许横向通过，绿灯时只允许纵向通过。

7.9

考虑这样的哲学家就餐问题：筷子放在桌子中央，并且每个哲学家可使用任意两根筷子。假定一次只能请求一根筷子。设计一条简单规则，根据现有筷子的分配来确定一个特定请求是否可以满足不会出现死锁。

解答:

简单规则：当桌子上筷子数>1或请求的哲学家手上已有一根筷子时，允许请求。

7.13

假设一个系统具有如下的快照。
采用银行家算法，回答下面的问题：
a. 通过进程可以完成执行的顺序，说明系统处于安全状态。
b. 当进程 P_1 的请求为（1，1，0，0）时，能否立即允许这一请求?
c. 当进程 P_4 的请求为（0，0，2，0）时，能够立即允许这一请求?

解答

a.

执行安全算法，得到执行顺序P0，P3，P1，P2,P4

- $Work = [3,3,2,1]$
- i=0， $Work = [5,3,2,2]$;
- i=3， $Work = [6,6,3,4]$;
- i=1， $Work = [9,7,5,5]$;
- i=2， $Work = [11,8,5,8]$;

- i=4， $Work = [12, 12, 8, 10]$;
- $Finish = [1, 1, 1, 1, 1]$，系统处于安全状态。

b.

- $Request_1 \leq Need_1, Request_1 \leq Available$，符合。
- 修改$Available = [2, 2, 2, 1], Allocation_1 = [3, 1, 0, 1]$。
- 进行安全算法，发现系统处于安全状态，允许请求。

c.

- $Request_4 \leq Need_4, Request_4 \leq Available$，符合。
- 修改$Available = [3, 3, 0, 1], Allocation_4 = [1, 4, 5, 2]$。
- 进行安全算法，发现系统处于不安全状态，不允许请求。