第二章

1、在P、V操作中，若系统中总共有n个进程，当信号量S初始化m时（n>m），则处于等待状态的进程至少有n-m个。（并分析原因）

**原因：**每当有一个进程进入临界区时，信号量S就减1，从m-1直到减到0为止

当S=0时，后面的进程无法再进入临界区，只能等待资源。因此，最多有m个进程并发地访问临界资源，那么处于等待状态的进程至少有n-m个。

2、对于进程的描述，下列哪个选项是错误的？（D）

A 进程可分为用户级进程和系统级进程；

B 每个进程都有自己的进程控制块（PCB）；

C 进程状态的切换需要操作系统内核来控制；

D 对于文件的读写操作，不需要创建进程来完成。

3、举一个例子说明进程如何在三种状态（执行、就绪、阻塞）之间转换。注意：要结合例子，说明这四种转换及其原因。

**例如**：在分时系统中，

若干个进程得到了资源，就排在就绪队列上。OS从中选中了进程1，给它一个时间片，使其运行，即**就绪->执行**；

时间片用完了，进程1从**执行->就绪**，OS从就绪队列中选中进程2，分配一个时间片，让其执行；

进程2在执行过程中，突然需要某种资源但是没得到，就把自己阻塞，即**执行->阻塞**，让出处理机，让其他进程运行；

当进程2得到资源后，就被唤醒，即从**阻塞->就绪**，重新等待处理机调度。

4、用一个整型信号量写出一个不会死锁的哲学家进餐问题的算法。（方法3选1）

方法1：最多有4个人同时拿左边的筷子；

Var chopsticks：aray[0,…,4] of semaphore :=[1,1,1,1,1];

Var room : semaphore := 4;

Repeat

Wait(room);

Wait (chopstick [i] );

Wait (chopstick[ (i+1) mod 5 ] );

Eat;

signal (chopstick [i] );

signal (chopstick[ (i+1) mod 5 ] );

signal(room);

until false;

方法2：当左右筷子都拿起，才吃饭；

Var chopsticks：aray[0,…,4] of semaphore :=[1,1,1,1,1];

Varmutex: semaphore := 1;

Repeat

wait(mutex)

wait (chopstick [i]);

wait(chopstick[ (i+1) mod 5 ]);

signal(mutex);

Eat;

signal (chopstick [i]);

signa( chopstick[ (i+1) mod 5 ]);

until false;

方法3：奇数号哲学家拿左边筷子，再拿右边，偶数号相反。

Var chopsticks：aray[0,…,4] of semaphore :=[1,1,1,1,1];

Repeat

If (imod 2 ==1)

Wait (chopstick[ (i+1) mod 5 ] );

Wait (chopstick [i] );

Else

Wait (chopstick [i] );

Wait (chopstick[ (i+1) mod 5 ] );

Eat;

Ssignal (chopstick [i], chopstick[ (i+1) mod 5 ]);

until false;

第三章

1．有五个进程A、B、C、D、E，到达时间和服务时间如下表所示，请按照**非抢占式的动态**高优先权优先算法，完成下表，并给出计算过程

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 进程名称 | A | B | C | D | E |
| 进程情况 | 到达时间 | 0 | 1 | 2 | 4 | 5 |
| 服务时间 | 4 | 1 | 3 | 6 | 3 |
| 高优先权优先算法 | 等待时间 | 0 | 3 | 3 | 7 | 3 |
| 开始时间 | 0 | 4 | 5 | 11 | 8 |
| 完成时间 | 4 | 5 | 8 | 17 | 11 |
| 周转时间 | 4 | 4 | 6 | 13 | 6 |

当t=4时，A完成，B, C, D 已到达

W\_B=1+3/1=4;

W\_C=1+2/3=5/3;

W\_D=1+0/6=1

当t=5时，B完成，E已到达

W\_C=1+3/3=2;

W\_D=1+1/6=7/6;

W\_E=1+0/3=1

当t=8时，C完成

W\_D=1+4/6=5/3

W\_E=1+3/3=2

综上所述，运行顺序为A->B->C->E->D

2、有6个进程A、B、C、D、E、F，以及三种资源a、b、c，这6个进程对这三种资源的最大需求和已经分配的见下表，假如系统中现有a、b、c的数目分别为1,1,1，请按照能够避免死锁的银行家算法，给出相应的分配顺序，并且写出每次分配后，Work集合的变化。

|  |  |  |
| --- | --- | --- |
| 进程 | 最大需求  a、b、c | 已分配  a、b、c |
| A | 3，2，5 | 1，0，0 |
| B | 4，2，4 | 1，0，0 |
| C | 2，2，2 | 0，0，1 |
| D | 1，1，2 | 0，0，2 |
| E | 2，2，4 | 1，1，0 |
| F | 1，1，3 | 0，0，1 |

分配前，Work=[1，1，1]。

根据最大需求-已分配<=当前Work 的原则：

首先符合条件的是进程D，

运行完毕后，Work=Work+Allocation[D]=[1，1，3]；

然后进程F符合条件，

运行完毕后，Work=Work+Allocation[F]=[1，1，4]；

然后进程E符合条件，

运行完毕后，Work=Work+Allocation[E]=[2，2，4]；

然后进程C符合条件，

运行完毕后，Work=Work+Allocation[C]=[2，2，5]；

然后进程A符合条件，

运行完毕后，Work=Work+Allocation[A]=[3，2，5]；

然后进程B符合条件，

运行完毕后，Work=Work+Allocation[B]=[4，2，5]；

因此，运行顺序为D、F、E、C、A、B。