第 5 章 同步练习题

一、选择题

1. 在为多道程序所提供的可共享的系统资源不足时，可能出现死锁。但是，不适当的（ D ）也可能产生死锁。

A. 进程优先权 B. 资源的线性分配 C. 分配队列优先权 D. 进程推进顺序

2. 在操作系统中，对信号量 S 的 P 原语操作定义中，使进程进入相应阻塞队列等待的条件是（ C ）。

A. S>0 B. S=0 C. S<0 D. S≠0

3. 任何两个并发进程之间(D)。

A.一定存在互斥关系 B.一定存在同步关系 C.一定彼此独立无关 D.可能存在同步或互斥关系

4. 下列不属于排除死锁的方法是(A) 。

A. 加锁 B.避免 C.检测和恢复 D. 预防

5. 产生死锁的必要条件不包括（D ）。

A. 非剥夺分配 B. 互斥作用 C. 部分分配 D. 非环路条件

6. 设与某资源相关联的信号量初值为 3，当前值为 1，若 M 表示该资源的可用个数，N 表示等待资源的进程数，则 M,N 分别是（B ）。

A．0，1 B. 1，0 C. 1，2 D. 2，0

7. 使用一个信号量协调 5 个进程对 3 个同类临界资源的访问，下列哪个信号量值不应该出现（ B ）。

A．0 B．–3 C．–1 D．3

8. 下列关于银行家算法的叙述中，正确的是 （B） 。

A．银行家算法可以预防死锁

B．当系统处于安全状态时，系统中一定无死锁进程

C．当系统处于不安全状态时，系统中一定会出现死锁进程

D．银行家算法破坏了死锁必要条件中的“请求和保持”条件

解析：银行家算法是避免死锁的方法。利用银行家算法，系统处于安全状态时没有死锁进程，答案选 B。

9. 某计算机系统中有 8 台打印机，有 K 个进程竞争使用，每个进程最多需要 3 台打印机。该系统可能**会**发生死锁的 K 的最小值是（C）

解析：不死锁需要 2K+1<=8，最多支持 3 个进程并发。注意问的如果是“不会发生死锁的最大值”就选 B。 4 个以上就死锁，所以会死锁的最小值是 4。

A．2 B.3 C.4 D.5

10. 某时刻进程的资源使用情况如下表所示。此时的安全序列是（D）

A．P1，P2，P3，P4 B．P1，P3，P2，P4 C．P1，P4，P3，P2 D．不存在

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	Rl	R2	R3	R1	R2	R3
P1	2	0	0	0	0	1	0	2	1
P2	1	2	0	1	3	2			

P3	0	1	1	1	3	1			
P4	0	0	1	2	0	0			

解答：按照现有的可用资源，调度序列是 P1->P4, 当 P4 执行完释放资源后，P2 和 P3 对资源 R2 的资源请求都不能得到满足，所以不存在安全序列。

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	Rl	R2	R3	R1	R2	R3
P1	2	0	0	0	0	1	2	2	1
P2	1	2	0	1	3	2			
P3	0	1	1	1	3	1			
P4	0	0	1	2	0	0	2	2	2

11. 假设 5 个进程 P0、P1、P2、P3、P4 共享三类资源 R1、R2、R3，这些资源总数分别为 18、6、22。T0 时刻的资源分配情况如下表所示，此时存在的一个安全序列是(D) （2012）

进程	已分配资源			尚需资源		
	R1	R2	R3	Rl	R2	R3
P0	3	2	3	2	3	7
P1	4	0	3	1	3	3
P2	4	0	5	0	0	6
P3	2	0	4	2	2	1
P4	3	1	4	1	1	0

A. P0, P2, P4, P1, P3 B. P1, P0, P3, P4, P2 C. P2，P1，P0，P3，P4 D. P3, P4, P2, P1, P0

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	Rl	R2	R3	R1	R2	R3
P0	3	2	3	2	3	7	2	3	3
P1	4	0	3	1	3	3			
P2	4	0	5	0	0	6			
P3	2	0	4	2	2	1			
P4	3	1	4	1	1	0			

解答：（1）T0 时刻：P0 和 P2 的需求都满足不了，所以答案 A 、C 排除。

（2）按选项 B 分配：

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	Rl	R2	R3	R1	R2	R3
P0	3	2	3	2	3	7			

P1	4	0	3	1	3	3	6	3	6
P2	4	0	5	0	0	6			
P3	2	0	4	2	2	1			
P4	3	1	4	1	1	0			

P1 运行结束，释放资源后：可用资源向量（6,3,6），不能满足 P0 对资源 R3 的需求，该选项是错误的。

（3）选项 D 的分配情况如下：**P3->P4-> P2-> P1-> P0**

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	Rl	R2	R3	R1	R2	R3
P0	3	2	3	2	3	7	18	6	22
P1	4	0	3	1	3	3	15	4	19
P2	4	0	5	0	0	6	11	4	16
P3	2	0	4	2	2	1	4	3	7
P4	3	1	4	1	1	0	7	4	11

二、判断题

1. 若资源分配图中存在环，则一定产生了死锁。 （F）
2. 计算机系统产生死锁的根本原因是系统中进程太多。 （F）
3. 任何两个并发进程之间一定存在同步或互斥关系。（F）
4. 临界区是指进程中实现进程互斥的那段代码。（T ）
5. 在计算机系统中必须互斥地使用的资源一定是软件资源。（F ）
6. 按银行家算法来分配资源是不会产生死锁的。（T）

三、填空题

1. 每执行一次 P 操作，信号量的数值 S 减 1。若 S<0，则该进程 阻塞 。
2. 设系统的互斥资源 R 共有 m 个，n 个进程共享（n>m）；用 PV 操作实现其互斥，则信号量 S 的变化范围为 m-n<=s<=m 。
3. P 操作相当于 申请资源，V 操作相当于释放 资源。
4. 实现临界区互斥的方法有锁操作法和 信号量 。
5. 某系统中共有 10 台磁带机被 m 个进程竞争，每个进程最多要求 3 台磁带机，当 m 的取值为不超过 4 的整数时，系统不会发生死锁
6. 各并发进程彼此相互等待对方所拥有的资源，在得到对方的资源之前不会释放自己所拥有的资源，从而造成大家都想得到资源而又得不到资源，各并发进程不能继续向前推进的状态，称为 死锁 。
7. 若信号量 S 的初值定义为 10，则针对 S 连续调用 16 次 P 操作，再连续调用 13 次 V 操作后 S 的值应该为 7 。
8. 实现临界区互斥的方法有开关中断法、硬件原子指令、自旋锁和_____（互斥信号量/互斥锁）

四、生产者消费者问题

1.（7 分）某博物馆最多可容纳 500 人同时参观，有一个出入口，该出入口一次仅允许一个人通过。参观者的活动描述如下：

cobegin

参观者进程 i：

{

进门；

参观；

出门；

}

coend

请添加必要的信号量和 P、V 操作，以实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

【答案】

定义两个信号量

Semaphore empty = 500; // 博物馆可以容纳的最多人数（2 分）

Semaphore mutex = 1; // 用于出入口资源的控制（2 分）

```
参观者进程 i; （3 分）
{
P ( empty );
P ( mutex );
进门;
V( mutex );
参观;
P ( mutex );
出门;
V( mutex );
V( empty );
}
```

【评分说明】

① 信号量初值给 1 分，说明含义给 1 分，两个信号量的初值和含义共 4 分。

② 对 mutex 的 P、V 操作正确给 2 分。 ③ 对 empty 的 P、V 操作正确给 1 分。

④ 其它答案，参照①~③的标准给分

2. 三个进程 P1、P2、P3 互斥使用一个包含 N（N>0）个单元的缓冲区。P1 每次用 produce（）生成一个正整数并用 put（）送入缓冲区某一空单元中；P2 每次用 getodd（）从该缓冲区中取出一个奇数并用 countodd（）统计奇数个数；P3 每次用 geteven（）从该缓冲区中取出一个偶数并用 counteven（）统计偶数个数。请用信号量机制实现这三个进程的同步与互斥活动，并说明所定义的信号量的含义。要求用伪代码描述。（2009）

定义信号量 S1 控制 P1 与 P2 之间的同步；S2 控制 P1 与 P3 之间的同步；empty 控制生产者与消费者之间的同步；mutex 控制进程间互斥使用缓冲区。程序如下：

【答案】

Var s1=0,s2=0,empty=N,mutex=1;

```
P1:begin
   x=produce();
   P(empty);
   P(mutex);
   put();
   if（x%2==0）V(s2);elseV(s1);
   V(mutex);
end.
```

```
P2:begin
   P(s1);
   P(mutex);
   Getodd();
   Countodd():=countodd()+1;
   V(mutex);
   V(empty);
end.
```

```
P3:begin
   P(s2);
   P(mutex);
   Geteven();
   Counteven():=counteven()+1;
   V(mutex);
   V(empty);
   end.
```

3.有 A、B 两人通过信箱进行辩论，每人都从自己的信箱中取得对方的问题。将答案和向对方提出的新问题组成一个邮件放入对方的邮箱中，设 A 的信箱最多放 M 个邮件，B 的信箱最多放 N 个邮件。初始时 A 的信箱有 x 个邮件，B 的邮箱为空。辩论者每取出一个邮件，邮件数减 1。当信箱不为空时，辩论者才能从信箱中取邮件，否则等待。当信箱不满时，辩论者才能将新邮件放入信箱，否则等待。**（2015）**

请添加必要的信号量和 P、V 操作，以实现上述过程的同步，要求写出完整过程，并说明信号量的含义和初值。A、B 两人操作过程如下：

```
Code Begin
A{
while(TRUE){
      从 A 的信箱中取出一个邮件；
      回答问题并提出一个新问题；
      将新邮件放入 B 的信箱；
      }
}
```

```
B{
while(TRUE){
      从 B 的信箱中取出一个邮件；
      回答问题并提出一个新问题；
      将新邮件放入 A 的信箱；
      }
}
Code End
```

Semaphore mutexA=1; Semaphore mutexB=1;

Semaphore emptyA=M-x; Semaphore emptyB=N; Semaphore fullA=x; Semaphore fullB=0;

```
Code Begin
A{
while(TRUE){
   P(fullA);
   P(mutexA)
   Get a mail from A_mailbox；
   V(mutexA);
   V(emptyA);
   Answer the question and raise a question;
   P(emptyB);
   P(mutexB)
   send the mail to B；
   V(mutexB);
   V(fullB);
   }
}
```

```
B{
while(TRUE){
   P(fullB);
   P(mutexB)
   Get a mail from B_mailbox；
   V(mutexB);
   V(emptyB);
   Answer the question and raise a question;
   P(emptyA);
   P(mutexA)
   send the mail to A；
   V(mutexA);
   V(fullA);
   }
}
Code End
```

五、简答题

1. 叙述死锁问题及解决办法。

死锁：一组并发进程彼此互相等待对方所拥有的资源，在得到对方的资源之前不释放自己所拥有的资源,若无外力驱动，各进程不能继续向前推进的僵死状态。

死锁的原因：并发进程的资源竞争、进程推进顺序不当

死锁的解决方法分为：（1）死锁的预防：破坏死锁产生的四个必要条件至少其一（互斥条件、不可抢占、请求保持、环路等待）。（2）死锁的避免：分配资源前首先测试系统的资源状况及分配后的安全状态，如银行家算法。（3）死锁的检测和恢复：剥夺资源、进程回滚、撤销进程。

2.（1）实现临界区互斥的有哪几种方法？（2）写出自旋锁和互斥锁的 PV 操作的实现方法。

（1）实现临界区互斥的方法有开关中断法、硬件原子指令、自旋锁和互斥信号量（互斥锁）。

硬件原子指令：循环测试，可能饥饿、死锁；普通用户要使用需要了解特定机器指令的细节，操作系统可以使用。

开关中断法：单处理器系统下在临界区中防止进程调度；缺点：用户控制系统中断非常危险。

自旋锁：“忙等待”，占用 CPU 时间；等待锁时没有上下文切换；适用：临界区持锁时间较短、多处理器系统。

互斥锁：没有忙等待；等待锁时上下文切换。

（2）自旋锁

```
acquire() {
        while (!available)
            ; /* busy wait */
        available = false;;
  }
release() {
        available = true;
  }
```

互斥锁:

```
typedef struct{
    int value;
    struct process    *list;
} semaphore
```

```
wait(semaphore *S) {
    S->value--;
    if (S->value < 0) {
        add this process to S->list;
        block();
    }
}
```

```
signal(semaphore *S) {
    S->value++;
    if (S->value <= 0) {
        remove a process P from S->list;
        wakeup(P);
    }
}
```