操作系统 2022 课后应用题作业 2

1. 假定磁盘有 200 个柱面，编号 $0 \sim 199$，当前移动臂位于 143 号柱面上，并刚刚完成 125 号柱面的服务请求。如果请求队列的先后顺序是：86，147，91，177，94，150，102，175，130；试问：为了完成上述请求，下列算法移动臂所移动的总量分别是多少？并给出移动臂移动的顺序。

(1) 先来先服务算法；

(2) 最短查找时间优先算法；

(3) 双向扫描算法；

(4) 电梯调度算法。

解答：

(1) 先来先服务算法：
移动臂移动的总量为：$(143-86)+(147-86)+(147-91)+(177-91)+(177-94)+(150-94)+(150-102)+(175-102)+(175-102)+(175-130)=57+61+56+86+83+56+48+73+45=565$
移动臂移动的顺序为：$143-86-147-91-177-94-150-102-175-130$

(2) 最短查找时间优先算法：
移动臂移动的总量为：$(150-143)+(150-86)+(177-86)=162$
移动臂移动的顺序为：$143-147-150-130-102-94-91-86-175-177$

(3) 双向扫描算法：
移动臂移动的总量为：$(199-143)+(199-86)=169$
移动臂移动的顺序为：$143-147-150-175-177-199-130-102-94-91-86$

(4) 电梯调度算法：
移动臂移动的总量为：$(177-143)+(177-86)=125$
移动臂移动的顺序为：$143-147-150-175-177-130-102-94-91-86$

2. 有一个磁盘组共有 10 个盘面，每个盘面有 100 个磁道，每个磁道有 16 个扇区。若以扇区为分配单位，现问：

(1) 用位示图管理磁盘空间，则位示图占用多少空间？

(2) 若空白文件目录的每个目录项占 5 个字节，则什么时候空白文件目录大于位示图？

解答：

(1) 该磁盘共有扇区数为 $10 \times 100 \times 16 = 16000$ 个，所以位示图占用空间为 $16000 \div 8 = 2000$ 字节

(2) 当空白文件目录数大于 400 时，空白文件目录大于位示图

3. 假设在 Unix 文件系统中，inode 节点中分别含有 10 个直接地址的索引和一、二、三级间接索引。若设每个盘块有 512B 大小，每个盘块中可存放 128 个盘块地址，则

(1) 一个 1MB 的文件占用多少间接盘块？

(2) 一个 25MB 的文件占用多少间接盘块？

解答：

(1) 直接盘块容量：$512\text{B} \times 10 \div 1024 = 5\text{KB}$
一级间接盘块容量：$128 \times 512\text{B} \div 1024 = 64\text{KB}$
二级间接盘块容量：$128 \times 64\text{KB} = 8192\text{KB}$
三级间接盘块容量：$8192\text{KB} \times 1024 = 1048576\text{KB}$

1MB 的文件占用 10 个直接盘块和 128 个一级间接盘块，二级间接盘块占用数量为 $(1024-5-64)\text{KB} \div 512\text{B} = 1910$ 个

(2) 25MB 的文件占用 10 个直接盘块、128 个一级间接盘块和 $128 \times 128 = 16384$ 个二级间接盘块，三级间接盘块的占用数量为 $(25 \times 1024 - 5 - 64 - 8192)\text{KB} \div 512\text{B} = 34678$ 块

4. 设有 n 个进程共享一个互斥段，如果：(1) 每次只允许一个进程进入互斥段；(2) 每次最多允许 m 个 $(m \leq n)$ 进程同时进入互斥段。

试问：以上两种情况下所采用的信号量初值是否相同？试给出信号量值的变化范围。

解答：

所采用的信号量的初值不同。

对于 (1) 的情况，信号量的初始值为 1，变化范围为 $[-n+1, 1]$。当没有进程进入互斥段时，信号量为 1；最多可能存在 $n-1$ 个进程等待进入互斥段，此时信号量的值为 $-n+1$。

对于 (2) 的情况，信号量的初始值为 m，变化范围为 $[-n+m, m]$。当没有进程进入互斥段时，信号量为 m；最多可能存在 $n-m$ 个进程等待进入互斥段，此时信号量的值为 $-n+m$。

5. 有两个优先级相同的进程 P_1 和 P_2，其各自程序如下，信号量 S_1 和 S_2 的初值均 0。试问 P_1、P_2 并发执行后，x、y、z 的值各为多少？

```
P1(){                          P2(){
    y = 1;                         x = 1;
    y = y + 3;                     x = x + 5;
    V(S1);                         P(S1);
    z = y + 1;                     x = x + y;
    P(S2);                         V(S2);
    y = z + y;                     z = z + x;
}                              }
```

解答：

为了方便描述，对上述进程语句进行编号

```
P1() {                         P2() {
  y = 1;          ①              x = 1;          ⑤
  y = y + 3;      ②              x = x + 5;      ⑥
  V(S1);                         P(S1);
  z = y + 1;      ③              x = x + y;      ⑦
  P(S2);                         V(S2);
  y = z + y;      ④              z = z + x;      ⑧
}                              }
```

①、②、⑤和⑥是不相交语句，可以任何次序交错执行，而结果是唯一的。接着无论系统如何调度进程并发执行，当执行到语句⑦时，可以得到 $x = 10, y = 4$。按 Bernstein 条件，语句③的执行结果不受语句⑦的影响，故语句③执行后得到 $z = 5$。最后，语句④和⑧并发执行，这时得到了两种结果为：

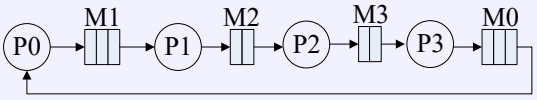
语句④先执行（即执行顺序为③、⑦、④、⑧）的结果为 $x = 10, y = 9, z = 15$。

语句⑧先执行（即执行顺序为③、⑦、⑧、④）的结果为 $x = 10, y = 19, z = 15$。

此外，还有第三种情况，语句③被推迟，直至语句⑧后再执行，即执行顺序为⑦、⑧、③、④。这时 z 的值只可能是 $y + 1 = 5$，故 $y = z + y = 5 + 4 = 9$，而 $x = 10$。

即第三种情况的结果为 $x = 10, y = 9, z = 5$。

6. 四个进程 $P_i(i=0,\cdots,3)$ 和四个信箱 $M_j(j=0,\cdots,3)$，进程间借助相邻信箱传递消息，即 P_i 每次从 M_i 中取一条消息，经加工后送入 $M_{(i+1) \bmod 4}$，其中 M_0、M_1、M_2、M_3 分别可存放 3、3、2、2 个消息。初始状态下，M_0 装了三条消息，其余为空。试以 P、V 操作为工具，写出 $P_i(i=0,\cdots,3)$ 的同步工作算法。



解答：

```
semaphore mutex1, mutex2, mutex3, mutex0; //限制同一信箱同时只能有一个进程操作
mutex1 = mutex2 = mutex3 = mutex0 = 1;
semaphore put1, put2, put3, put0; //限制能否放信件
put1 = 3; put2 = 2; put3 = 2; put0 = 0;
semaphore get1, get2, get3, get0; //限制能否取信件
get1 = 0; get2 = 0; get3 = 0; get0 = 3;
int in0, in1, in2, in3, out0, out1, out2, out3; //存放和取出信件的位置指针
in0 = in1 = in2 = in3 = out0 = out1 = out2 = out3 = 0;
cobegin
```

```
Process P0(){
  while(1){
    P(get0);
      P(mutex0);
        {从M0[out0]取一条消息};
        out0 = (out0 + 1) % 3;
      V(mutex0);
    V(put0);
    {加工消息};
    P(put1);
      P(mutex1);
        {消息存M1[in1]};
        in1 = (in1 + 1) % 3;
      V(mutex1);
    V(get1);
  }
}
```

```
Process P1(){
  while(1){
    P(get1);
      P(mutex1);
        {从M1[out1]取一条消息};
        out1 = (out1 + 1) % 3;
      V(mutex1);
    V(put1);
    {加工消息};
    P(put2);
      P(mutex2);
        {消息存M2[in2]};
        in2 = (in2 + 1) % 2;
      V(mutex2);
    V(get2);
  }
}
```

```
Process P2(){
  while(1){
    P(get2);
      P(mutex2);
        {从M2[out2]取一条消息};
        out2 = (out2 + 1) % 2;
      V(mutex2);
    V(put0);
    {加工消息};
    P(put3);
      P(mutex3);
        {消息存M3[in3]};
        in3 = (in3 + 1) % 2;
      V(mutex3);
    V(get3);
  }
}
```

```
Process P3(){
  while(1){
    P(get3);
      P(mutex3);
        {从M3[out3]取一条消息};
        out3 = (out3 + 1) % 2;
      V(mutex3);
    V(put3);
    {加工消息};
    P(put0);
      P(mutex0);
        {消息存M0[in0]};
        in0 = (in0 + 1) % 3;
      V(mutex0);
    V(get0);
  }
}
```

```
coend
```

7. 有一个阅览室，读者进入时必须先在一张登记表上登记，此表为每个座位列出一个表目，包括座位号、姓名，读者离开时要注销登记信息；假如阅览室共有 100 个座位。试用：(1) 信号量和 PV 操作；(2) 管程，实现用户进程的同步算法。

解答：

(1) 使用信号量和 PV 操作

```
struct {
  char name[10];
  int number;
} A[100];
for (int i = 0; i < 100; i++){
  A[i].number = i; A[i].name = null;
}
semaphore mutex, seat;
mutex = 1; seat = 100;
cobegin
Process readeri(char readername[]){
  P(seat);
  P(mutex);
    for (int i = 0; i < 100; i++){
      if (A[i].name == null){
        A[i].name = readername;
        {读者得到座位i};
        break;
      }
    }
  V(mutex);
  {进入阅览室，座位号i，坐下读书};
  P(mutex);
    A[i].name = null;
  V(mutex);
  V(seat);
  {离开阅览室};
}
coend
```

(2) 使用管程实现

```
type readbook = MONTOR {
  semaphore s;
  int s_count, i ,seatCount;
  char name[100];
  seatCount = 0; //已被占用的座位数
  InterfaceModule IM;
  DEFINE enterroom (), leaveroom();
  USE enter(), leave(), wait(), signal();
```

```
void enterroom(char[] readername){
  enter(IM);
    if (seatCount >= 100)
      wait(s, s_count, IM);
    seatCount++;
    for (int i = 0; i < 100; i++){
      if (name[i] == null){
        name[i] = readername;
        break;
      }
    }
    {获得座位i};
  leave(IM);
}
```

```
void leaveroom(char[] readername){
  enter(IM);
    seatCount--;
    for (int i = 0; i < 100; i++){
      if (name[i] == readername){
        name[i] = null;
        break;
      }
    }
    signal(s, s_count, IM);
  leave(IM);
}
```

```
cobegin
  process readeri(){
    readbook.enterroom(readername);
    {阅读};
    readbook.leaveroom(readername);
    {离开阅览室};
  }
coend
```

8. 在一个盒子里，混装了数量相等的黑白围棋子。现在用自动分拣系统把黑子、白子分开，设分拣系统有二个进程 P_1 和 P_2，其中 P_1 拣白子；P_2 拣黑子。规定每个进程每次拣一子；当一个进程在拣时，不允许另一个进程去拣；当一个进程拣了一子时，必须让另一个进程去拣。试分别使用 (1) PV 操作和 (2) 管程方法写出两进程 P_1 和 P_2 能并发正确执行的程序。

解答：

(1) 使用信号量和 PV 操作

```
semaphore s1, s2;
s1 = 1; s2 = 0;
cobegin
```

Process P1(){
  while(1){
  P(S1);
  {拣白子};
  V(S2);
  }
}
	Process P2(){
	  while(1){
	  P(S2);
	  {拣黑子};
	  V(S1);
	  }
	}

```
coend
```

(2) 使用管程实现

```
type pickup_chess = MONITOR {
  bool white_turn = true;
  semaphore s_white, s_black;
  int s_white_count, s_black_count;
  InterfaceModule IM;
  DEFINE white(), black();
  USE enter(), leave(), wait(), signal();
```

void white(){
  enter(IM);
  if (!white_turn)
  wait(s_white, s_white_count ,IM);
  white_turn = false;
  {拣白子};
  signal(s_black, s_black_count, IM);
  leave(IM);
}
	void black(){
	  enter(IM);
	  if (white_turn)
	  wait(s_black, s_black_count ,IM);
	  white_turn = true;
	  {拣黑子};
	  signal(s_white, s_white_count, IM);
	  leave(IM);
	}

```
cobegin
```

Process P1(){
  pickup_chess.white();
  othres;
}
	Process P2(){
	  pickup_chess.black();
	  othres;
	}

```
coend
```

9. 一组生产者进程和一组消费者进程共享 9 个缓冲区，每个缓冲区可以存放一个整数。生产者进程每次一次性地向 3 个缓冲区中写入整数，消费者进程每次从缓冲区取出一个整数。请用：(1) 信号量和 PV 操作；(2) 管程方法写出能够正确执行的程序。

解答：

(1) 使用信号量和 PV 操作

```
int buf[9];
int count, getptr, putptr;
count = 0; getptr = 0; putptr = 0;
semaphore s1, s2, put, get;
s1 = 1; s2 = 2; put = 0; get = 3;
cobegin
```

```
process producer_i(){
  while(1){
    P(put);
    {生产3个整数};
    P(s1);
      buf[putptr] = 整数1;
      putptr = (putptr + 1) % 9;
      buf[putptr] = 整数2;
      putptr = (putptr + 1) % 9;
      buf[putptr] = 整数2;
      putptr = (putptr + 1) % 9;
    V(s1);
    V(get);
    V(get);
    V(get);
  }
}
```

```
process consumer_j(){
  while(1){
    P(get);
    P(s2);
      int y = buf[getptr];
      count++;
      getptr = (getptr + 1) % 9;
      if (count == 3){
        count = 0;
        V(put);
      }
    V(s2);
    {消费整数y};
  }
}
```

```
coend
```

(2) 使用管程实现

```
type producer_consumer = MONITOR {
  int buf[9];
  int count, getptr, putptr;
  count = 0; getptr = 0; putptr = 0;
  semaphore put, get;
  int put_count, get_count;
  InterfaceModule IM;
  DEFINE put, get;
  USE enter(), leave(), wait(), signal();
```

```
process put(int x1, int x2, int x3){
  enter(IM);
    if (count > 6)
      wait(put, put_count, IM);
    count += 3;
    buf[putptr] = x1;
    putptr = (putptr + 1) % 9;
    buf[putptr] = x2;
    putptr = (putptr + 1) % 9;
    buf[putptr] = x3;
    putptr = (putptr + 1) % 9;
    signal(get, get_count ,IM);
    signal(get, get_count ,IM);
    signal(get, get_count ,IM);
  leave(IM);
}
```

```
process get(){
  enter(IM);
    if (count == 0)
      wait(get, get_count, IM);
    y = buf[getptr];
    getptr = (getptr + 1) % 9;
    count--;
    if (count < 7){
      signal(put, put_count, IM);
    }else if (count > 0){
      signal(get, get_count, IM);
    }
  leave(IM);
}
```

```
cobegin
```

```
process producer_i(){
  while(1){
    {生产3个整数};
    producer_consumer.put(a1, a2, a3);
  }
}
```

```
process consumer_j(){
  while(1){
    y = producer_consumer.get();
    {消费整数y};
  }
}
```

```
coend
```

10. 系统有 A, B, C, D 共 4 种资源，在某时刻进程 P_0, P_1, P_2, P_3 和 P_4 对资源的占有和需求情况如表，试解答下列问题：

进程	Allocation				Claim				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	3	2	0	0	4	4	1	6	2	2
P_1	1	0	0	0	2	7	5	0				
P_2	1	3	5	4	3	6	10	10				
P_3	0	3	3	2	0	9	8	4				
P_4	0	0	1	4	0	6	6	10				

(1) 系统此时处于安全状态吗？试给出一个可能的安全序列。

(2) 若此时进程 P_2 发出 $request1(1, 2, 2, 2)$，系统能分配资源给它吗？为什么？

解答：

(1) 系统处于安全状态，一个安全序列为 $P_0 - P_3 - P_4 - P_1 - P_2$。

进程	Available				C_{ik}-A_{ik}				Allocation				Available+Allocation				Possible
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	
P_0	1	6	2	2	0	0	1	2	0	0	3	2	1	6	5	4	True
P_3	1	6	5	4	0	6	5	2	0	3	3	2	1	9	8	6	True
P_4	1	9	8	6	0	6	5	6	0	0	1	4	1	9	9	10	True
P_1	1	9	9	10	1	7	5	0	1	0	0	0	2	9	9	10	True
P_2	2	9	9	10	2	3	5	6	1	3	5	4	3	12	14	14	True

(2) 不能分配。假设分配给进程 P_2，此时 $Available = (0, 4, 0, 0)$，不能满足任何一个进程，此时系统处于不安全状态，因此不能分配。