一． 实验题目

假定系统有 5 个进程(p0,p1,p2,p3,p4)和三类资源(A，B，C)，各种资源的数量分别为 10，5，7，在 T0 时刻的资源分配情况如下图：

进程	Max	Allocation	Need	Available
P0	7 5 3	0 1 0	7 4 3	3 3 2 （2 3 0）
P1	3 2 2	2 0 0 （3 0 2）	1 2 2 （0 2 0）	
P2	9 0 2	3 0 2	6 0 0	
P3	2 2 2	2 1 1	0 1 1	
P4	4 3 3	0 0 2	4 3 1	

二． 实验目的

死锁会引起计算机工作僵死，因此操作系统中必须防止。本实验的目的在于让学生独立的使用高级语言编写和调试一个系统动态分配资源的简单模拟程序，了解死锁产生的条件和原因，并采用银行家算法有效地防止死锁的发生，以加深对课堂上所讲授的知识的理解。

三． 实验内容

步骤：

创建初始状态：先输入 Resource、Max 和 Allocation，再计算出 Need、Available：

```
    void initial()
    {
        int i,j;
        printf("输入 M 种资源的总数量:\n");
        for(i=0;i<M;i++)
        {
            scanf("%d",&Resource[i]);
            Available[i]=Resource[i];
        }
        printf("输入 N 个进程分别对 M 种资源的最大需求量:\n");
        for(j=0;j<N;j++)
```

```
            for(i=0;i<M;i++)
                scanf("%d",&Max[j][i]);
        printf("输入N个进程获得M种资源的数量:\n");
        for(j=0;j<N;j++)
            for(i=0;i<M;i++)
                scanf("%d",&Allocation[j][i]);
        for(j=0;j<N;j++)
            for(i=0;i<M;i++)
                Need[j][i]=Max[j][i]-Allocation[j][i];
        for(j=0;j<M;j++)
            for(i=0;i<N;i++)
                Available[j]=Available[j]-Allocation[i][j];
}
```

返回同时满足两个条件{①Finish[i]=false; ②Need[i][j]≤Work[j]}的进程下标 i(修改Finish[i]=true)，否则返回-1：

```
    int isfinish()
    {
        int i,j,count;
        for(i=0;i<N;i++)
        {
            for(j=0,count=0;j<M;j++)
                if(Finish[i]==0&&Need[i][j]<=Work[j])
                {
                    count++;
                }
            if(count==3)
            {
                for(j=0;j<M;j++)
                    Work[j]+=Allocation[i][j];
                Finish[i]=1;
                return i;
            }
        }
        return -1;
}
```

判定当前状态是否为安全状态 （返回 true 或 false)，把安全序列的下标放入List[N]数组：

```
    int issafe()
    {
        int i,a,count=0;
        for(i=0;i<M;i++)
            Work[i]=Available[i];
```

```
        for(i=0;i<N;i++)
            Finish[i]=0;
        for(i=0;i<N;i++)
        {
            a=isfinish();
            if(a!=-1)
            {
                List[i]=a;
                count++;
            }
        }
        if(count==5)
            return 1;
        else
            return 0;
}
```

表示第 i 个进程请求 M 类资源 request[M]：

```
    void reqresource(int i, int Request[M])
    {
        int flag,count1,count2;
        int j;
        //Step1:  判断条件 Request[j]≤Need[i][j]
        for(j=0,count1=0;j<M;j++)
            if(Request[j]<=Need[i][j])
                count1++;
        //Step2:  判断条件 Request[j]≤Available[j]
        for(j=0,count2=0;j<M;j++)
            if(Request[j]<=Available[j])
                count2++;
        if(count2!=3)
            printf("\n 尚无足够的资源，第%d 个进程堵塞。\n",i);
        //Step3:  预先分配
        if(count2==3&&count1==3)
        {
            for(j=0;j<M;j++)
            {
                Available[j]=Available[j]-Request[j];
                Allocation[i][j]=Allocation[i][j]+Request[j];
                Need[i][j]=Need[i][j]-Request[j];
            }
            if(issafe()==0)
            {
                printf("\n 不存在安全序列，不是安全状态。\n");
```

```
                for(j=0;j<M;j++)
                {
                    Available[j]=Available[j]+Request[j];
                    Allocation[i][j]=Allocation[i][j]-Request[j];
                    Need[i][j]=Need[i][j]+Request[j];
                }
            }
            else
            {
                printf("\n 是安全序列分配成功!\n");
                printList();
            }
        }
    }
```

主函数：

```
    main()
    {
        int reqid=-1,j,req[M];
        initial();
        printState();
        if(issafe()==0)
        {
            printf("Initial state is unsafe!\n");
        }
        else
        {
            printf("\nInitial state is safe!\n");
            printList();
            printf("Input the id of request process:");
            scanf("%d",&reqid);
            while(reqid>=0 && reqid<N)   //输入进程 id 是否合法
            {
                printf("Input request resources:");
                for(j=0;j<M;j++)
                {
                    scanf("%d",&req[j]);
                }
                reqresource(reqid, req);
                printState();
                printf("Input the id of request process:");
                scanf("%d",&reqid);
            }
        }
```

}

运行结果：

```
输入M种资源的总数量:
10 5 7
输入N个进程分别对M种资源的最大需求量:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
输入N个进程获得M种资源的数量:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
```

```
状态表:
| Process | Max | Allocation | Need | Available |
| P0 | 7 5 3 | 0 1 0 | 7 4 3 | 3 3 2 |
| P1 | 3 2 2 | 2 0 0 | 1 2 2 | |
| P2 | 9 0 2 | 3 0 2 | 6 0 0 | |
| P3 | 2 2 2 | 2 1 1 | 0 1 1 | |
| P4 | 4 3 3 | 0 0 2 | 4 3 1 | |

Initial state is safe!

安全序列表如下:
| Process | Work | Need | Allocation | Work+Alloc | Finish |
| P1 | 3 3 2 | 1 2 2 | 2 0 0 | 5 3 2 | true |
| P3 | 5 3 2 | 0 1 1 | 2 1 1 | 7 4 3 | true |
| P0 | 7 4 3 | 7 4 3 | 0 1 0 | 7 5 3 | true |
| P2 | 7 5 3 | 6 0 0 | 3 0 2 | 10 5 5 | true |
| P4 | 10 5 5 | 4 3 1 | 0 0 2 | 10 5 7 | true |
Input the id of request process:
```

存在一个安全序列<p1, p3, p0, p2, p4>

```
Input the id of request process:1
Input request resources:1 0 2

是安全序列分配成功!

安全序列表如下:
```

Process	Work			Need			Allocation			Work+Alloc			Finish
P1	2	3	0	0	2	0	3	0	2	5	3	2	true
P3	5	3	2	0	1	1	2	1	1	7	4	3	true
P0	7	4	3	7	4	3	0	1	0	7	5	3	true
P2	7	5	3	6	0	0	3	0	2	10	5	5	true
P4	10	5	5	4	3	1	0	0	2	10	5	7	true

```
状态表:
```

Process	Max			Allocation			Need			Available		
P0	7	5	3	0	1	0	7	4	3	2	3	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

存在一个安全序列<p1, p3, p0, p2, p4>

```
Input the id of request process:4
Input request resources:3 3 0

尚无足够的资源，第4个进程堵塞。
状态表:
```

Process	Max			Allocation			Need			Available		
P0	7	5	3	0	1	0	7	4	3	2	3	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

```
Input the id of request process:0
Input request resources:0 2 0

不存在安全序列，不是安全状态。
状态表:
```

Process	Max			Allocation			Need			Available		
P0	7	5	3	0	1	0	7	4	3	2	3	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

```
Input the id of request process:0
Input request resources:0 1 0

是安全序列分配成功!

安全序列表如下:
```

Process	Work			Need			Allocation			Work+Alloc			Finish
P1	2	2	0	0	2	0	3	0	2	5	2	2	true
P3	5	2	2	0	1	1	2	1	1	7	3	3	true
P0	7	3	3	7	3	3	0	2	0	7	5	3	true
P2	7	5	3	6	0	0	3	0	2	10	5	5	true
P4	10	5	5	4	3	1	0	0	2	10	5	7	true

状态表:

Process	Max			Allocation			Need			Available		
P0	7	5	3	0	2	0	7	3	3	2	2	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			