实验三 银行家算法

一、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。

二、实验内容

设计有 n 个进程共享 m 个系统资源的系统，进程可动态的申请和释放资源，系统按各进程的申请动态的分配资源。

系统能显示各个进程申请和释放资源，以及系统动态分配资源的过程，便于用户观察和分析；

题目

可利用资源向量 Available ，它是一个含有 m 个元素的数组，其中的每一个元素代表一类可利用的资源的数目，其初始值是系统中所配置的该类全部可用资源数目。其数值随该类资源的分配和回收而动态地改变。如果 Available（j）=k，标是系统中现有 Rj 类资源 k 个。

最大需求矩阵 Max，这是一个 n×m 的矩阵，它定义了系统中 n 个进程中的每一个进程对 m 类资源的最大需求。如果 Max（i，j）=k，表示进程 i 需要 Rj 类资源的最大数目为 k。

分配矩阵 Allocation，这是一个 n×m 的矩阵，它定义了系统中的每类资源当前一分配到每一个进程的资源数。如果 Allocation（i，j）=k，表示进程 i 当前已经分到 Rj 类资源的数目为 k。Allocation i 表示进程 i 的分配向量，有矩阵 Allocation 的第 i 行构成。

需求矩阵 Need，这是一个 n×m 的矩阵，用以表示每个进程还需

要的各类资源的数目。如果 Need（i，j）=k，表示进程 i 还需要 Rj 类资源 k 个，才能完成其任务。Need i 表示进程 i 的需求向量，由矩阵 Need 的第 i 行构成。

上述三个矩阵间存在关系：Need（i，j）=Max（i，j）-Allocation（i，j）；

银行家算法

Request i 是进程 Pi 的请求向量。Request i （j）=k 表示进程 Pi 请求分配 Rj 类资源 k 个。当 Pi 发出资源请求后，系统按下述步骤进行检查：

如果 Request i ≤Need，则转向步骤 2；否则，认为出错，因为它所请求的资源数已超过它当前的最大需求量。

如果 Request i ≤Available，则转向步骤 3；否则，表示系统中尚无足够的资源满足 Pi 的申请，Pi 必须等待。

系统试探性地把资源分配给进程 Pi，并修改下面数据结构中的数值：

Available = Available - Request i

Allocation i= Allocation i+ Request i

Need i= Need i - Request i

系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。如果安全才正式将资源分配给进程 Pi，以完成本次分配；否则，将试探分配作废，恢复原来的资源分配状态，让进程 Pi 等待。

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：

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3	3	3	2
										（2	3	0）

P1	3	2	2	2	0	0	1	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

安全性算法

设置两个向量。

Work：它表示系统可提供给进程继续运行的各类资源数目，它包含 m 个元素，开始执行安全性算法时，Work = Available。

Finish：它表示系统是否有足够的资源分配给进程，使之运行完成，开始 Finish（I）=false；当有足够资源分配给进程 Pi 时，令 Finish（i）=true；

从进程集合中找到一个能满足下述条件的进程。

Finish（i）= = false；

Need i ≤work；

如找到则执行步骤 3；否则，执行步骤 4；

当进程 Pi 获得资源后，可顺利执行直到完成，并释放出分配给它的资源，故应执行

Work = work + Allocation i

Finish（i）=true；转向步骤 2；

若所有进程的 Finish（i）都为 true，则表示系统处于安全状态；否则，系统处于不安全状态。

内容

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
int N = 5, i;    //进程数
```

```
int Max[5][3];
int Allocation[5][3];
int Need[5][3];
int Available[3] = { 3,3,2 };
int Sequence[5] = { 0 };
int Request[3] = { 0 };
void Information() {
   char buf[60], ch[2];
   //从文件读入数据
   FILE* fp;
   if ((fp = fopen("1.txt", "r")) == NULL) {
      printf("打开失败");
   }
   for (i = 0; i < N; i++) {
      fgets(buf, 60, fp);
      ch[0] = buf[0];
      Max[i][0] = atoi(ch);
      ch[0] = buf[2];
      Max[i][1] = atoi(ch);
      ch[0] = buf[4];
      Max[i][2] = atoi(ch);
   }
   for (i = 0; i < N; i++) {
      fgets(buf, 60, fp);
      ch[0] = buf[0];
```

```
        Allocation[i][0] = atoi(ch);
        ch[0] = buf[2];
        Allocation[i][1] = atoi(ch);
        ch[0] = buf[4];
        Allocation[i][2] = atoi(ch);
    }
    for (i = 0; i < N; i++) {
        fgets(buf, 60, fp);
        ch[0] = buf[0];
        Need[i][0] = atoi(ch);
        ch[0] = buf[2];
        Need[i][1] = atoi(ch);
        ch[0] = buf[4];
        Need[i][2] = atoi(ch);
    }
}
int issafe() {
    int flag = 0,i1=0, j=0;
    int safeIndex = 0;
    int allFinish = 0;
    int work[3] = { 0 };
    int Finish[5] = { 0 };
    //初始化 work
    for (i = 0; i < 3; i++) {
        work[i] = Available[i];
```

```
    }
    while (1)
    {
        for (j = 0; j < 5; j++) //寻找 Need 小于等于 Work 的进程
        {
            if (Finish[j])
                continue;
            flag = true;
            for (i = 0; i < 3; i++)
            {
                if (Need[j][i] <= work[i])
                    continue;
                else
                {
                    flag = false;
                    break;
                }
            }
            if (flag) break; //判断第 j 个进程是否满足 Need 小于等于 Work
        }
            if (flag && (Finish[j] == false)) //是否找到 Need 小于等于 Work 的进程并且进程未完成
            {
                for (i = 0; i < 3; i++)
```

```
            {
                work[i] = work[i] + Allocation[j][i];
            }
            Finish[j] = true; //进程完成
            Sequence[i1] = j; //安全序列
            i1++;
        }
        else
        {
            for (i = 0; i < 5; i++)
            {
                if (Finish[i]) continue;
                else return false;
            }
            return true;
        }
    }
    return true;
}
void request_option(int a)              //资源的请求选择
{
    int i;
    int t = true;
    for (i = 0; i < 3; i++)                 //判断 Requesti 是否小于等于 Need
```

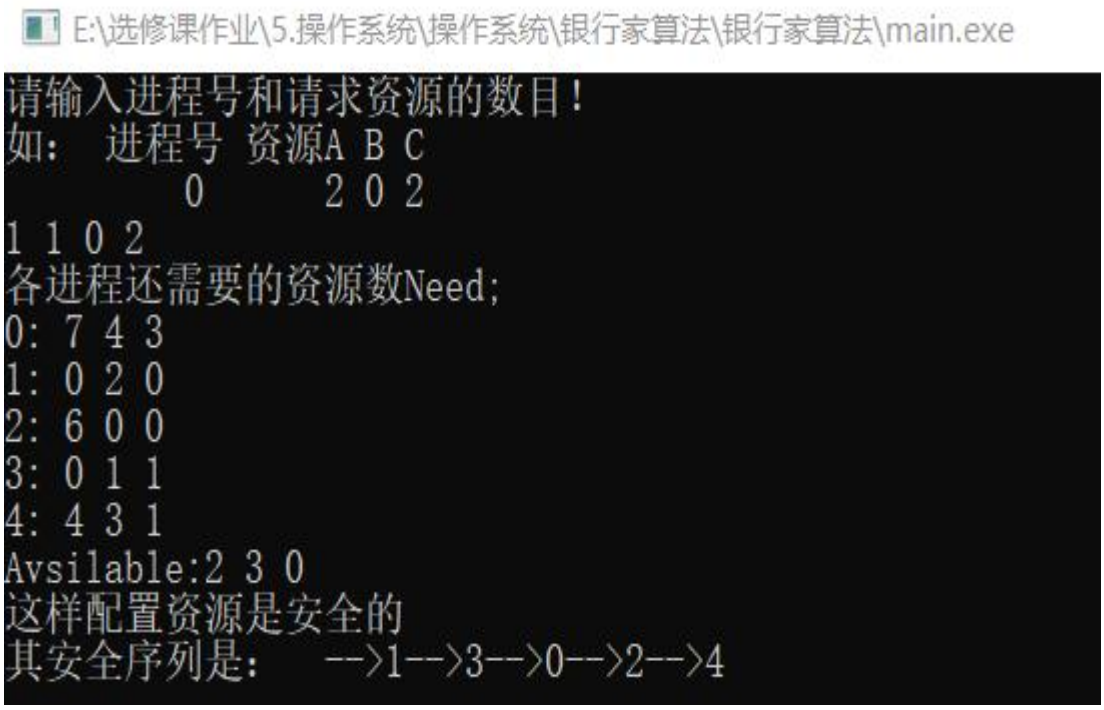
```
    {
        if (Request[i] <= Need[a][i])
            continue;
        else
        {
            printf("错误!            ");            //Requesti 大于 Needi 出现错误，请求失败
                t = false;
            break;
        }
    }
    if (t)
    {
        for (i = 0; i < 3; i++)          //判断 Requesti 是否小于等于 Availablei
        {
            if (Request[i] <= Need[a][i])
                continue;
            else
            {
                printf("错误，进程堵塞!                ");
//Requesti 大于 Availablei 出现错误,进程 Pi 堵塞
                    t = false;
                break;
            }
```

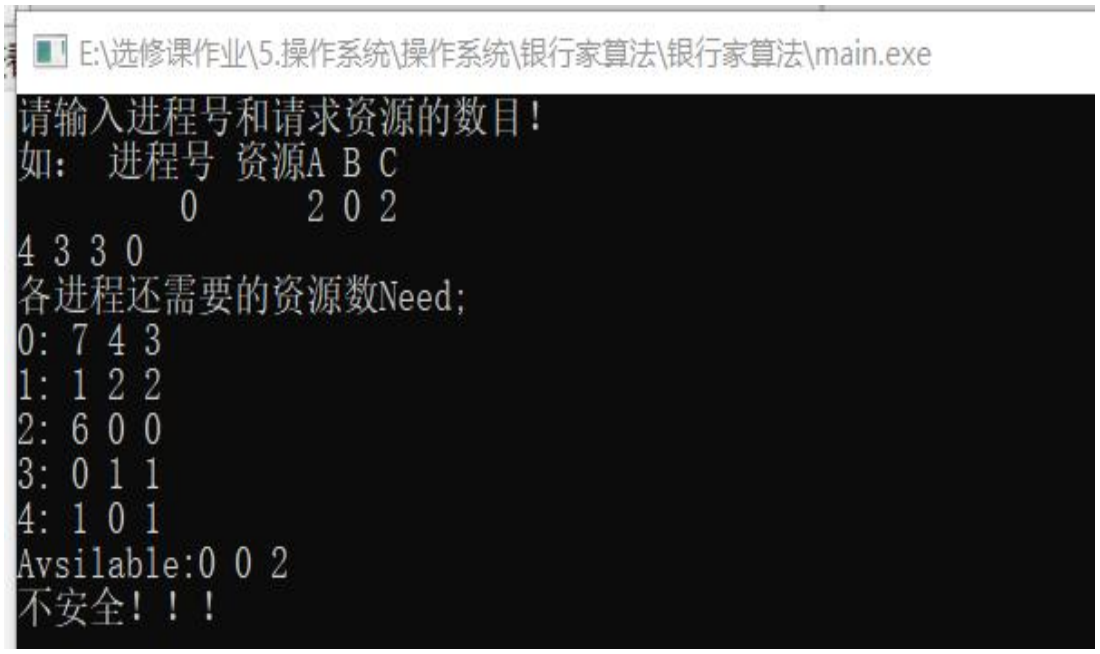
```
        }
    }
    if (t)                  //试分配
    {
        for (i = 0; i < 3; i++)
        {
            Available[i] = Available[i] - Request[i];
            Allocation[a][i] = Allocation[a][i] + Request[i];
            Need[a][i] = Need[a][i] - Request[i];
        }
    }
}
int main() {
    char c;
    int request_m;
    Information();
    printf("                    /********银  行  家  算  法
********/\n");
    printf("确认已经在'1.tet'文档中正确输入各进程的相关信息
后按回车键");
    getchar();
    printf("各进程还需要的资源数Need;\n");
    for (i = 0; i < 5; i++) {
        printf("%d:  %d  %d  %d\n",i,  Need[i][0],  Need[i][1],
Need[i][2]);
```

```
    }
    printf("Avsilable:%d %d %d\n", Available[0], Available[1],
Available[2]);
    if (issafe())
    {
        printf("这样配置资源是安全的\n 其安全序列是：\t");
        for (i = 0; i < 5; i++) {
            printf("-->%d", Sequence[i]);
        }
    }
    printf("有进程发出 Request 请求向量吗？<Enter y or Y>\n");
    c = getchar();
    if (c == 'y' || 'Y') {
        printf("请输入进程号和请求资源的数目！\n");
        printf("如： 进程号 资源 A B C\n");
        printf("          0       2 0 2\n");
        scanf("%d%d%d%d", &request_m, &Request[0], &Request[1],
&Request[2]);
        request_option(request_m);
        printf("各进程还需要的资源数 Need;\n");
        for (i = 0; i < 5; i++) {
            printf("%d: %d %d %d\n", i, Need[i][0], Need[i][1],
Need[i][2]);
        }
        printf("Avsilable:%d    %d    %d\n",    Available[0],
```

```
Available[1], Available[2]);
        if (issafe())
        {
            printf("这样配置资源是安全的\n 其安全序列是：\t");
            for (i = 0; i < 5; i++) {
                printf("-->%d", Sequence[i]);
            }
        }
        else
        {
            printf("不安全！！！");
        }
    }
    else
    {
        printf("欢迎使用！");
    }
}
```

结果


E:\选修课作业\5.操作系统\操作系统\银行家算法\银行家算法\main.exe
请输入进程号和请求资源的数目!
如: 进程号 资源A B C
0 2 0 2
1 1 0 2
各进程还需要的资源数Need;
0: 7 4 3
1: 0 2 0
2: 6 0 0
3: 0 1 1
4: 4 3 1
Avsilable:2 3 0
这样配置资源是安全的
其安全序列是: -->1-->3-->0-->2-->4


E:\选修课作业\5.操作系统\操作系统\银行家算法\银行家算法\main.exe
请输入进程号和请求资源的数目!
如: 进程号 资源A B C
0 2 0 2
4 3 3 0
各进程还需要的资源数Need;
0: 7 4 3
1: 1 2 2
2: 6 0 0
3: 0 1 1
4: 1 0 1
Avsilable:0 0 2
不安全!!!