武汉大学计算机学院

课程实验(设计)报告

专业(班)：计算机科学与技术 三班

学　　号：2013301500100

姓　　名：秦贤康

课程名称：操作系统设计

任课教师：宋伟

2015 年　　12 月　　21 日

说　　明

1、本课程设计报告包括：实习题目、实习内容及设计思想（设计思路、主要数据结构、主要代码结构及代码段分析）、上机实习所用平台及相关软件、调试过程（测试数据设计、测试结果分析）、总结（实习中遇到的问题及解决方法、实习中产生的错误及原因分析、实习体会及收获）。

2、课程设计报告用 A4 纸打印。

标题 4 号宋体加粗，正文 5 号宋体，图表文字标注、程序等用 6 号。

页边距：左边 3.17cm，右边 3.17 cm，上 2.54 cm，下 2.54 cm。

行间距：1.5 倍行距，一页 30 行，每行 39 字。

页脚：1.5 cm，页脚(页码，居中)。

银行家算法

1 实习内容

编写实现银行家算法，实现资源的安全分配。

2 实习题目及设计思想

2.1 实习题目

初始状态下，设置数据结构存储可利用资源向量（Available），最大需求矩阵（MAX），分配矩阵（Allocation），需求矩阵（Need），输入待分配进程队列和所需资源。

设计安全性算法，设置工作向量表示系统可提供进程继续运行的可利用资源数目。

如果进程队列可以顺利执行打印输出资源分配情况,如果进程队列不能顺利执行打印输出分配过程，提示出现死锁位置。

2.2 设计思路

·用一个新的数组保存安全序列

·搜索满足条件的进程时，添加一个哨兵变量

2.3 主要数据结构

```
public int processNum; //进程数
public int resourceNum;//资源数
public int Max[][];
public int Allocation[][];
public int Available[];
public int Need[][];
public int Work[][];
public Boolean Finish[];
public int proceesList[]; //安全序列
```

2.4 主要代码结构及代码段分析

资源申请

```
public void request(int p,int r1,int r2,int r3){
        int step = 1;
        if(r1<=Need[p-1][0] && r2<=Need[p-1][1] && r3<=Need[p-1][2]){
                step = 2;
        }else{
                System.out.println("进程所需要的资源数目超过它所宣布的最大值");
                return;
        }
        if(step == 2 && r1<=Available[0] && r2<=Available[1] && r3<=Available[2]){
```

```
                step=3;
        }else{
                System.out.println("系统中无足够的资源满足进程申请");
                return;
        }
        if(step == 3){
                Available[0] -=r1;
                Available[1] -=r2;
                Available[2] -=r3;
                Allocation[p-1][0]+=r1;
                Allocation[p-1][1]+=r2;
                Allocation[p-1][2]+=r3;
                Need[p-1][0]-=r1;
                Need[p-1][1]-=r2;
                Need[p-1][2]-=r3;
        }
        System.out.println("P"+p+":("+r1+","+r2+","+r3+") 申请资源后的资源分配表");
        show();
        System.out.println("检查 P"+p+" "+Available[0]+Available[1]+Available[2]);
        if(check() == true){
                System.out.println("检查通过，找到安全序列");
                for(int i=0;i<processNum;i++){
                        System.out.print("->"+(proceesList[i]+1));
                }
                System.out.println("");
        }else{
                System.out.println("检查不通过，回退分配操作");
                Available[0] +=r1;
                Available[1] +=r2;
                Available[2] +=r3;
                Allocation[p-1][0]-=r1;
                Allocation[p-1][1]-=r2;
                Allocation[p-1][2]-=r3;
                Need[p-1][0]+=r1;
                Need[p-1][1]+=r2;
                Need[p-1][2]+=r3;
        }
}
```

安全检查：

```
public Boolean check(){
        for(int i=0;i<processNum;i++){Finish[i]=false;};//初始化 finish
        int workTemp[] = new int[]{Available[0], Available[1], Available[2]};
        Boolean flag = true;
        int j = 0;
```

```
                while(flag){
                        flag = false;
                        for(int i=0;i<processNum;i++){
                                if(Finish[i] == false && Need[i][0] <= workTemp[0] && Need[i][1] <= workTemp[1]
&& Need[i][2] <= workTemp[2]){
                                        Work[i][0] = workTemp[0];Work[i][1] = workTemp[1];
                                        Work[i][2] = workTemp[2];

                                        flag = true;
                                        Finish[i] = true;
                                        proceesList[j++]=i;
                                        workTemp[0]=Work[i][0]+Allocation[i][0];
                                        workTemp[1]=Work[i][1]+Allocation[i][1];
                                        workTemp[2]=Work[i][2]+Allocation[i][2];
                                        break;
                                }
                        }
                }
                // 输出
                if(j==processNum-1)return false; //如果所有的都为 true,检查通过
                return true;
        }
```

3 上机实习所用平台及相关软件

OS: mint Mint 17.2 rafaela
Kernel: x86_64 Linux 3.16.0-38-generic
CPU: AMD A8-4500M APU with Radeon HD Graphics @ 1.9GHz
GPU: Gallium 0.4 on AMD ARUBA
RAM: 1522MiB / 3337MiB
javac 1.7.0_91

4 调试过程

```
#!/bin/sh
# 测试两次申请
cat test1|java Bank >test1.out
```

运行 test.sh 自动进行输入输出，只要事先把测试数据写到 test 文件中就可以了

4.1 测试数据设计

2 1 0 1
2 号进程进行 1 0 1 的申请
1 1 0 1
1 号进程进行 1 0 1 的申请

4.2 测试结果分析（红字为说明）

申请资源后的安全检查

```
_____________________________________________________________________________
|  资源|    Work |     Need     |  Allocation   | Work+Allocation |
|进程  |R1 R2 R3|  R1 R2 R3  |  R1  R2  R3  |     R1  R2 R3    | Finish
-----------------------------------------------------------------------
|  P2  | 1  1  2|   1  0  2  |  5   1   1   |      6   2   3   | true
|  P1  | 6  2  3|   2  2  2  |  1   0   0   |      7   2   3   | true
|  P3  | 7  2  3|   1  0  3  |  2   1   1   |      9   3   4   | true
|  P4  | 9  3  4|   4  2  0  |  0   0   2   |      9   3   6   | true
-----------------------------------------------------------------------  初始化检查通过
```

```
_____________________________________________________
|  资源|    Max   | Allocation |   Need   | Available |
|进程  |R1 R2 R3|  R1 R2 R3  |R1 R2 R3|  R1 R2 R3 |
-----------------------------------------------------
|      |        |            |        | 1   1   2 |
|  P2  | 6  1  3|   5  1  1  | 1  0  2|           |
|  P1  | 3  2  2|   1  0  0  | 2  2  2|           |
|  P3  | 3  1  4|   2  1  1  | 1  0  3|           |
|  P4  | 4  2  2|   0  0  2  | 4  2  0|           |显示初始化资源表
-----------------------------------------------------
```

P2:(1,0,1）申请资源后的资源分配表

```
_____________________________________________________
|  资源|    Max   | Allocation |   Need   | Available |
|进程  |R1 R2 R3|  R1 R2 R3  |R1 R2 R3|  R1 R2 R3 |
-----------------------------------------------------
|      |        |            |        | 0   1   1 |
|  P2  | 6  1  3|   6  1  2  | 0  0  1|           |
|  P1  | 3  2  2|   1  0  0  | 2  2  2|           |
|  P3  | 3  1  4|   2  1  1  | 1  0  3|           |
|  P4  | 4  2  2|   0  0  2  | 4  2  0|           |
-----------------------------------------------------  分配给第二个进程 1 0 1 之后的分配表
```

检查 P2 011

申请资源后的安全检查

```
_____________________________________________________________________________
|  资源|    Work |     Need     |  Allocation   | Work+Allocation |
|进程  |R1 R2 R3|  R1 R2 R3  |  R1  R2  R3  |     R1  R2 R3    | Finish
-----------------------------------------------------------------------
|  P2  | 0  1  1|   0  0  1  |  6   1   2   |      6   2   3   | true
|  P1  | 6  2  3|   2  2  2  |  1   0   0   |      7   2   3   | true
|  P3  | 7  2  3|   1  0  3  |  2   1   1   |      9   3   4   | true
|  P4  | 9  3  4|   4  2  0  |  0   0   2   |      9   3   6   | true
-----------------------------------------------------------------------
```

检查通过，找到安全序列

->2->1->3->4 检查通过，并找出了安全序列

系统中无足够的资源满足进程申请 此时进行 1 1 0 1 申请，正如教材 98 页所说，可用资源 0 1 0 已不能满足任何进程的需要，系统不能分配资源

综上，可以看出，本次实验完成了对银行家算法的模拟实现

5 总结

银行家算法是 Dijkstra 给出的具有代表性的死锁避免算法，在模式实现它的过程中，体会到了它的精妙之处。通过本次实验了解到用银行家算法来预防死锁是可靠的，但也是非常保守的，因为它限制了进程对资源的存取，从而降低了进程的并发运行程度。死锁检测并不限制进程对资源的申请，只要有，就分配，但这也可能造成死锁。但由于死锁并不是经常发生的，故大大提高了系统运行的效率。 总之，通过本实验，使我进一步加深理解和掌握银行家算法。