# 山西工程技术学院

# 实验报告

（ 2023 - 2024 学年第 学期）

课程名称：操作系统实验

专业班级：

学　　号：

学生姓名：

任课教师：

2024 年 6 月

# 实验报告

| 实验名称 | 银行家算法 | | | 指导教师 | |
|---|---|---|---|---|---|
| 实验类型 | 验证型 | 实验学时 | 4 | 实验时间 | 2024.6 |

**一、实验目的与要求**

银行家算法是避免死锁的一种重要方法，通过编写一个简单的银行家算法程序，加深了解有关资源申请、避免死锁等概念，并体会和了解死锁和避免死锁的具体实施方法。

**二、实验环境**

**Vscode**

**三、实验内容和步骤**

先对用户提出的请求进行合法性检查，即检查请求是否大于需要的，是否大于可利用的。若请求合法，则进行预分配，对分配后的状态调用安全性算法进行检查。若安全，则分配；若不安全，则拒绝申请，恢复到原来的状态，拒绝申请。

实验使用C语言模拟实现银行家算法，写出程序，并正确运行程序。

```
#include<stdio.h>
#include<stdlib.h>
#define False 0
#define True 1

char NAME[100]={0};
int Max[100][100]={0};
int Allocation[100][100]={0};
int Need[100][100]={0};
int Available[100]={0};
int Request[100]={0};
int Work[100]={0};
int Finish[100]={0};
int Security[100]={0};

int M=100;
int N=100;
```

```
void init()
{

     int i,j,m,n;
    int number,flag;
    char name;
    int temp[100]={0};

    printf("系统可用资源种类为:");
    scanf("%d",&n);
    N=n;
    for(i=0;i<n;i++)
    {
        printf("资源%d 的名称:",i);
        fflush(stdin);
        scanf("%c",&name);
        NAME[i]=name;
        printf("资源%c 的初始个数为:",name);
        scanf("%d",&number);
        Available[i]=number;
    }


    printf("\n 请输入进程的数量:");
    scanf("%d",&m);
    M=m;
    printf("请输入各进程的最大需求矩阵的值[Max]:\n");
    do{
        flag = False;
        for(i=0;i<M;i++)
            for(j=0;j<N;j++)
```

```
            {
                scanf("%d",&Max[i][j]);
                if(Max[i][j]>Available[j])
                    flag = True;
            }
        if(flag)
            printf("资源最大需求量大于系统中资源最大量，请重新输入!\n");
    } while(flag);



    do{
        flag=False;
        printf("请输入各进程已经分配的资源量[Allocation]:\n");
        for(i=0;i<M;i++)
        {
            for(j=0;j<N;j++)
            {
                scanf("%d",&Allocation[i][j]);
                if(Allocation[i][j]>Max[i][j])
                    flag=True;
                Need[i][j]=Max[i][j]-Allocation[i][j];
                temp[j]+=Allocation[i][j];
            }
        }
        if(flag)
            printf("分配的资源大于最大量，请重新输入!\n");
    }while(flag);

    for(j=0;j<N;j++)
        Available[j]=Available[j]-temp[j];
}

void showdata()
```

```
{
    int i,j;
    printf("**********************************************************\n");
    printf("系统目前可用的资源[Available]:\n");
    for(i=0;i<N;i++)
          printf("%c   ",NAME[i]);
    printf("\n");
    for(j=0;j<N;j++)
          printf("%d   ",Available[j]);
    printf("\n");
    printf("系统当前的资源分配情况如下：\n");
    printf("               Max        Allocation     Need\n");
    printf("进程名        ");
    for(j=0;j<3;j++){
        for(i=0;i<N;i++)
            printf("%c   ",NAME[i]);
        printf("       ");
    }
    printf("\n");
    for(i=0;i<M;i++){
        printf(" P%d          ",i);
         for(j=0;j<N;j++)
            printf("%d   ",Max[i][j]);
        printf("       ");
        for(j=0;j<N;j++)
            printf("%d   ",Allocation[i][j]);
        printf("       ");
        for(j=0;j<N;j++)
            printf("%d   ",Need[i][j]);
        printf("\n");
    }
}
int test(int i)
{
```

```
    for(int j=0;j<N;j++)
    {
        Available[j]=Available[j]-Request[j];
        Allocation[i][j]=Allocation[i][j]+Request[j];
        Need[i][j]=Need[i][j]-Request[j];
    }
    return True;
}



int Retest(int i)
{
    for(int j=0; j<N; j++)
    {
        Available[j] = Available[j] + Request[j];
        Allocation[i][j] = Allocation[i][j] - Request[j];
        Need[i][j] = Need[i][j] + Request[j];
    }
    return True;
}



int safe()
{
    int i,j,k=0,m,apply;
    for(j=0;j<N;j++)
          Work[j] = Available[j];
     for(i=0;i<M;i++)
        Finish[i] = False;
    for(i=0;i<M;i++){
        apply=0;
        for(j=0;j<N;j++){
            if(Finish[i]==False && Need[i][j]<=Work[j])
            {
```

```
                apply++;
                if(apply==N)
                {
                    for(m=0;m<N;m++)
                          Work[m]=Work[m]+Allocation[i][m];
                    Finish[i]=True;
                    Security[k++]=i;
                    i=-1;
                }
            }
        }
    }

    for(i=0;i<M;i++){
        if(Finish[i]==False){
            printf("系统不安全\n");
            return False;
        }
    }
     printf("系统是安全的!\n");
     printf("存在一个安全序列:");
    for(i=0;i<M;i++){
        printf("P%d",Security[i]);
        if(i<M-1)
            printf("->");
    }
    printf("\n");
    return True;
}


void bank()
{
    int flag = True;
```

```
int i,j;

printf("请输入请求分配资源的进程号(0-%d):",M-1);
 scanf("%d",&i);

printf("请输入进程 P%d 要申请的资源个数:\n",i);
for(j=0;j<N;j++)
{
    printf("%c:",NAME[j]);
    scanf("%d",&Request[j]);
}

 for (j=0;j<N;j++)
{
    if(Request[j]>Need[i][j])
    {
        printf("进程 P%d 申请的资源大于它需要的资源",i);
        printf("分配不合理，不予分配！\n");
        flag = False;
        break;
    }
    else
    {
            if(Request[j]>Available[j])
        {
            printf("进程%d 申请的资源大于系统现在可利用的资源",i);
            printf("\n");
            printf("系统尚无足够资源，不予分配!\n");
            flag = False;
            break;
        }
    }
 }
```

```
     if(flag) {
        test(i);
        showdata();
        if(!safe())
        {
            Retest(i);
            showdata();
        }
     }
}


int main()
{
    char choice;
    init();
     showdata();
    if(!safe()) exit(0);

    do{
        printf("************************************************************\n");
        printf("\n");
        printf("\n");
        printf("\t-------------------银行家算法演示-----------------\n");
        printf("                          R(r):请求分配    \n");
        printf("                          E(e):退出        \n");
        printf("\t-------------------------------------------------\n");
        printf("请选择：");
        fflush(stdin);
        scanf("%c",&choice);
        switch(choice)
        {
            case 'r':
            case 'R':
```

```
            bank();break;
        case 'e':
        case 'E':
            exit(0);
        default: printf("请正确选择!\n");break;
      }
   } while(choice);
}
```

```
系统可用资源种类为:3
资源0的名称:资源
的初始个数为:10
资源1的名称:资源
的初始个数为:5
资源2的名称:资源
的初始个数为:7

请输入进程的数量:5
请输入各进程的最大需求矩阵的值[Max]:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
请输入各进程已经分配的资源量[Allocation]:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
```

```
系统目前可用的资源[Available]:



3  3  2
系统当前的资源分配情况如下:
          Max          Allocation    Need
进程名









 P0        7  5  3       0  1  0       7  4  3
 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
系统是安全的!
存在一个安全序列:P1->P3->P0->P2->P4
************************************************************


        --------------------银行家算法演示------------------
                    R(r):请求分配
                    E(e):退出
        ----------------------------------------------------
请选择: 请正确选择!
************************************************************
```

**四、实验小结和思考**

在本次实验中，我们实现并测试了银行家算法。银行家算法是用于死锁避免的著名算法，通过检查系统资源的分配状态和进程的资源需求，确保在任何情况下都不会导致系统进入死锁状态。我们不仅掌握了银行家算法的实现和应用，还对死锁处理的不同方法有了更深入的理解。未来的工作中，可以探索将银行家算法与其他死锁处理方法相结合，设计出更为高效和可靠的资源管理策略。

| 实验成绩 | | 批阅日期 | | 批阅人 | |
|---|---|---|---|---|---|