实验四 银行家算法实验

姓名	学号	学院	日期
臧祝利	202011998088	人工智能学院	2022.10.30

实验目的

银行家算法是操作系统中避免死锁的典型算法，通过本实验加深对银行家算法的理解。

实验内容

用C语言或C++编写一个简单的银行家算法模拟程序，实现多个进程争用系统临界资源时的分配过程。要求程序实现：

- 当一进程请求一组资源时，先确定是否有足够的资源分配给该进程。如果有，再进一步计算在将这些资源分配给进程后，系统是否处于不安全状态。如果安全，显示安全序列，将资源分配给该进程，否则进程等待。
- 可以显示当前时刻各进程的的资源分配情况。

实验报告

实验环境

系统：Win10

IDE：vscode

使用`GBK`编码，若程序打开有中文乱码，请更换编码类型；

实验过程

变量定义

```
#define RESOURCE_NUM 10 // 最大资源数
#define PROCESS_NUM 10  // 最大进程数
#define Length 10       // 资源名称的最大长度

int Available[RESOURCE_NUM];  //可用资源矩阵
int MaxRequest[PROCESS_NUM][RESOURCE_NUM];  //最大需求矩阵
int Allocation[PROCESS_NUM][RESOURCE_NUM];  //分配矩阵
int Need[PROCESS_NUM][RESOURCE_NUM];  //需求矩阵
bool Finish[PROCESS_NUM];  //判断系统是否有足够资源分配给各个进程
int safeSeries[PROCESS_NUM];  //安全序列
int Request[RESOURCE_NUM];  //请求资源向量

int resource; //输入的资源数量
int process;  //输入的进程数量
char ResourceName[RESOURCE_NUM][Length];  //各资源名称
```

函数实现

包括如下函数：

- `showinfo()` 显示分配情况函数；
- `issafe()` 安全性算法；
  - 步骤
    - ①初始化 `work[i] = Available[i] ,Finish[i] = false for i from 0 to resource-1`
    - ②查找这样的 `i`，满足
      - `Finish[i] = false`
      - `Need[i]≤work`
      - 若不存在这样的 `i`，跳转至④
    - ③ `work = work + Allocation[i],Finish[i] = true`,返回②
    - ④如果所有的 `i` 都有 `Finish[i] = true`，说明处于安全状态；
- `banker()` 银行家算法；
  - 步骤
    - 检查请求是否超过需要
    - 检查请求是否能够满足
    - 尝试分配资源
    - 安全性检测
      - 通过：输出安全序列
      - 不通过：恢复原来状态；
- `set(int processID)` 尝试将资源分配给 `processID` 进程；
- `reset(int processID)` 恢复 `set` 前的状态；

showinfo()

```
void showInfo()
{
    printf("系统目前可用的资源[Available]:\n");
    for (int i = 0; i < resource; i++)
    {
        printf("%s\t", ResourceName[i]);
    }
    printf("\n");
    for (int i = 0; i < resource; i++)
    {
        printf("%d\t", Available[i]);
    }
    printf("\n");

    printf("\tMax  Allocation  Need\n");
    printf("进程");
    for (int i = 0; i < 3; i++)
    {
        printf("\t");
        for (int j = 0; j < resource; j++)
        {
            printf("%s ", ResourceName[j]);
        }
    }
    printf("\n");

    for (int i = 0; i < process; i++)
    {
```

```
        printf("%d\t", i);
        for (int j = 0; j < resource; j++)
        {
            printf("%d ", MaxRequest[i][j]);
        }
        printf("\t");
        for (int j = 0; j < resource; j++)
        {
            printf("%d ", Allocation[i][j]);
        }
        printf("\t");
        for (int j = 0; j < resource; j++)
        {
            printf("%d ", Need[i][j]);
        }
        printf("\n");
    }
}
```

issafe()

```
bool issafe()
{
    int work[RESOURCE_NUM];  //存放系统可提供资源量
    int safe_index = 0;  //序列下标
    int num = 0;
    for (int i = 0; i < resource; i++)
    {
        work[i] = Available[i];  //初始化work
    }
    for (int i = 0; i < process; i++)
    {
        Finish[i] = false;  //初始化Finish
    }

    for (int i = 0; i < process; i++)  //求安全序列
    {
        num = 0;  //需要的数目小于可利用资源数的资源个数
        for (int j = 0; j < resource; j++)
        {
            if (Finish[i] == false && Need[i][j] ≤ work[j])
            {
                num++;
                //每类资源都少于才可分配
                if (num == resource)
                {
                    for (int k = 0; k < resource; k++)
                    {
                        work[k] += Allocation[i][k];  //更改当前可分配的资源
                    }
                    Finish[i] = true;
                    safeSeries[safe_index++] = i;
                    i = -1;  //使得每次查询都从第一个进程开始;
                }
            }
        }
    }
}

for (int i = 0; i < process; i++)
{
```

```
        if (Finish[i] == false)
        {
            printf("系统不存在安全序列!\n");
            return false;
        }
    }
    showInfo();
    printf("系统安全!\n");
    printf("存在一个安全序列:");
    for (int i = 0; i < process; i++)
    {
        printf("%d", safeSeries[i]);
        if (i ≠ process - 1)
        {
            printf("→");
        }
    }
    printf("\n");
    return true;
}
```

banker()

```
void banker()
{
    bool flag = true; //判断能否进入算法的下一步
    int processID;    //所选择的进程号
    printf("请输入请求分配的资源进程号(0-%d):", process - 1);
    scanf("%d", &processID);

    printf("请输入进程%d要申请的资源个数:\n", processID);
    for (int i = 0; i < resource; i++)
    {
        printf("%s:", ResourceName[i]);
        scanf("%d", &Request[i]);
    }

    for (int i = 0; i < resource; i++)
    {
        if (Request[i] > Need[processID][i]) //判断申请是否大于需求
        {
            printf("进程%d申请的资源大于它需要的资源", processID);
            printf("分配不合理，不予分配!\n");
            flag = false;
            break;
        }
        else
        {
            if (Request[i] > Available[i]) //判断申请是否大于当前可分配资源
            {
                printf("进程%d申请的资源大于系统现在可利用的资源", processID);
                printf("\n");
                printf("系统无足够资源，不予分配!\n");
                flag = false;
                break;
            }
        }
    }
    //尝试分配资源，寻找安全序列；
    if (flag)
```

```
    {
        set(processID); //尝试分配
        if (!issafe())  //寻找安全序列；如果不安全，恢复状态
        {
            reset(processID);
            printf("不予分配!\n");
        }
    }
}
```

set()

```
void set(int processID)
{
    for (int i = 0; i < resource; i++)
    {
        Available[i] = Available[i] - Request[i];
        Allocation[processID][i] = Allocation[processID][i] + Request[i];
        Need[processID][i] = Need[processID][i] - Request[i];
    }
    return;
}
```

reset()

```
void reset(int processID)
{
    for (int i = 0; i < resource; i++)
    {
        Available[i] = Available[i] + Request[i];
        Allocation[processID][i] = Allocation[processID][i] - Request[i];
        Need[processID][i] = Need[processID][i] + Request[i];
    }
    return;
}
```

源代码

```
#include <stdio.h>
#include <stdlib.h>

#define RESOURCE_NUM 10 // 最大资源数
#define PROCESS_NUM 10  // 最大进程数
#define Length 10       // 资源名称的最大长度

int Available[RESOURCE_NUM];                //可用资源矩阵
int MaxRequest[PROCESS_NUM][RESOURCE_NUM]; //最大需求矩阵
int Allocation[PROCESS_NUM][RESOURCE_NUM]; //分配矩阵
int Need[PROCESS_NUM][RESOURCE_NUM];        //需求矩阵
bool Finish[PROCESS_NUM];                   //判断系统是否有足够资源分配给各个进程
int safeSeries[PROCESS_NUM];                //安全序列
int Request[RESOURCE_NUM];                  //请求资源向量

int resource;                              //输入的资源数量
int process;                               //输入的进程数量
char ResourceName[RESOURCE_NUM][Length]; //各资源名称

void showInfo()
```

```
{
    printf("系统目前可用的资源[Available]:\n");
    for (int i = 0; i < resource; i++)
    {
        printf("%s\t", ResourceName[i]);
    }
    printf("\n");
    for (int i = 0; i < resource; i++)
    {
        printf("%d\t", Available[i]);
    }
    printf("\n");

    printf("\tMax  Allocation  Need\n");
    printf("进程");
    for (int i = 0; i < 3; i++)
    {
        printf("\t");
        for (int j = 0; j < resource; j++)
        {
            printf("%s ", ResourceName[j]);
        }
    }
    printf("\n");

    for (int i = 0; i < process; i++)
    {
        printf("%d\t", i);
        for (int j = 0; j < resource; j++)
        {
            printf("%d ", MaxRequest[i][j]);
        }
        printf("\t");
        for (int j = 0; j < resource; j++)
        {
            printf("%d ", Allocation[i][j]);
        }
        printf("\t");
        for (int j = 0; j < resource; j++)
        {
            printf("%d ", Need[i][j]);
        }
        printf("\n");
    }
}

bool issafe()
{
    int work[RESOURCE_NUM]; //存放系统可提供资源量
    int safe_index = 0;     //序列下标
    int num = 0;
    for (int i = 0; i < resource; i++)
    {
        work[i] = Available[i]; //初始化work
    }
    for (int i = 0; i < process; i++)
    {
        Finish[i] = false; //初始化Finish
    }
```

```
for (int i = 0; i < process; i++) //求安全序列
{
    num = 0; //需要的数目小于可利用资源数的资源个数
    for (int j = 0; j < resource; j++)
    {
        if (Finish[i] == false && Need[i][j] ≤ work[j])
        {
            num++;
            //每类资源都少于才可分配
            if (num == resource)
            {
                for (int k = 0; k < resource; k++)
                {
                    work[k] += Allocation[i][k]; //更改当前可分配的资源
                }
                Finish[i] = true;
                safeSeries[safe_index++] = i;
                i = -1; //使得每次查询都从第一个进程开始;
            }
        }
    }
}

for (int i = 0; i < process; i++)
{
    if (Finish[i] == false)
    {
        printf("系统不存在安全序列!\n");
        return false;
    }
}
showInfo();
printf("系统安全!\n");
printf("存在一个安全序列:");
for (int i = 0; i < process; i++)
{
    printf("%d", safeSeries[i]);
    if (i ≠ process - 1)
    {
        printf("→");
    }
    printf("\n");
    return true;
}

void set(int processID)
{
    for (int i = 0; i < resource; i++)
    {
        Available[i] = Available[i] - Request[i];
        Allocation[processID][i] = Allocation[processID][i] + Request[i];
        Need[processID][i] = Need[processID][i] - Request[i];
    }
    return;
}

void reset(int processID)
{
    for (int i = 0; i < resource; i++)
```

```
    {
        Available[i] = Available[i] + Request[i];
        Allocation[processID][i] = Allocation[processID][i] - Request[i];
        Need[processID][i] = Need[processID][i] + Request[i];
    }
    return;
}

void banker()
{
    bool flag = true; //判断能否进入算法的下一步
    int processID;    //所选择的进程号
    printf("请输入请求分配的资源进程号(0-%d):", process - 1);
    scanf("%d", &processID);

    printf("请输入进程%d要申请的资源个数:\n", processID);
    for (int i = 0; i < resource; i++)
    {
        printf("%s:", ResourceName[i]);
        scanf("%d", &Request[i]);
    }

for (int i = 0; i < resource; i++)
{
    if (Request[i] > Need[processID][i]) //判断申请是否大于需求
    {
        printf("进程%d申请的资源大于它需要的资源", processID);
        printf("分配不合理，不予分配!\n");
        flag = false;
        break;
    }
    else
    {
        if (Request[i] > Available[i]) //判断申请是否大于当前可分配资源
        {
            printf("进程%d申请的资源大于系统现在可利用的资源", processID);
            printf("\n");
            printf("系统无足够资源，不予分配!\n");
            flag = false;
            break;
        }
    }
    //尝试分配资源，寻找安全序列；
    if (flag)
    {
        set(processID); //尝试分配
        if (!issafe())  //寻找安全序列；如果不安全，恢复状态
        {
            reset(processID);
            printf("不予分配!\n");
        }
    }
}

int main()
{
    printf("*****************单处理机系统进程调度实现*****************\n");
    printf("请首先输入系统可供资源种类的数量:");
    scanf("%d", &resource);
```

```
    for (int i = 0; i < resource; i++)
    {
        printf("输入资源%d的名称:", i + 1);
        scanf("%s", ResourceName[i]);
        printf("输入资源%d的数量:", i + 1);
        scanf("%d", &Available[i]);
    }
    printf("输入进程的数量:");
    scanf("%d", &process);
    printf("输入各进程的最大需求量(%d*%d矩阵)[MaxRequest]:\n", process, resource);
    for (int i = 0; i < process; i++)
    {
        for (int j = 0; j < resource; j++)
        {
            scanf("%d", &MaxRequest[i][j]);
        }
    }
    printf("输入各进程已经申请的资源量(%d*%d矩阵)[Allocation]:\n", process, resource);
    for (int i = 0; i < process; i++)
    {
        for (int j = 0; j < resource; j++)
        {
            scanf("%d", &Allocation[i][j]);
        }
    }
    //计算Need矩阵
    for (int i = 0; i < process; i++)
    {
        for (int j = 0; j < resource; j++)
        {
            Need[i][j] = MaxRequest[i][j] - Allocation[i][j];
        }
    }

    if (!issafe())
        exit(0);

    while (1)
    {
        printf("*****************银行家算法演示*****************\n");
        printf("\t\t1:分配资源\n\t\t2:显示分配情况\n\t\t0:离开\n");
        printf("************************************************\n");
        printf("请选择功能号:");
        int option;
        scanf("%d", &option);
        if (option == 1)
        {
            banker();
        }
        else if (option == 2)
        {
            showInfo();
            continue;
        }
        else
        {
            printf("已离开!\n");
            break;
        }
    }
```

```
    return 0;
}
```

运行结果

- 输入资源信息、最大需求矩阵、已申请资源矩阵；

  输出目前可用的资源量以及安全序列；

```
*****************单处理机系统进程调度实现*****************
请首先输入系统可供资源种类的数量:3
输入资源1的名称:a
输入资源1的数量:3
输入资源2的名称:b
输入资源2的数量:3
输入资源3的名称:c
输入资源3的数量:2
输入进程的数量:5
输入各进程的最大需求量(5*3矩阵)[MaxRequest]:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
输入各进程已经申请的资源量(5*3矩阵)[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]:
a       b       c
3       3       2
        Max  Allocation  Need
进程    a b c   a b c   a b c
0       7 5 3   0 1 0   7 4 3
1       3 2 2   2 0 0   1 2 2
2       9 0 2   3 0 2   6 0 0
3       2 2 2   2 1 1   0 1 1
4       4 3 3   0 0 2   4 3 1
系统安全!
存在一个安全序列:1->3->0->2->4
*****************银行家算法演示*****************
                1:分配资源
                2:显示分配情况
                0:离开
************************************************
请选择功能号:
```

- 输入1，为1进程分配1 0 2

```
请选择功能号:1
请输入请求分配的资源进程号(0-4):1
请输入进程1要申请的资源个数:
a:1 0 2
b:c:系统目前可用的资源[Available]:
a       b       c
2       3       0
        Max  Allocation  Need
进程    a b c   a b c   a b c
0       7 5 3   0 1 0   7 4 3
1       3 2 2   3 0 2   0 2 0
2       9 0 2   3 0 2   6 0 0
3       2 2 2   2 1 1   0 1 1
4       4 3 3   0 0 2   4 3 1
系统安全!
存在一个安全序列:1->3->0->2->4
*****************银行家算法演示*****************
                1:分配资源
                2:显示分配情况
                0:离开
************************************************
请选择功能号:
```

- 输入1，为4进程分配3 3 0

```
请选择功能号:1
请输入请求分配的资源进程号(0-4):4
请输入进程4要申请的资源个数:
a:3 3 0
b:c:进程4申请的资源大于系统现在可利用的资源
系统无足够资源，不予分配!
*****************银行家算法演示*****************
                1:分配资源
                2:显示分配情况
                0:离开
************************************************
请选择功能号:
```

- 输入1，为0进程分配0 2 0

```
请选择功能号:1
请输入请求分配的资源进程号(0-4):0
请输入进程0要申请的资源个数:
a:0 2 0
b:c:系统不存在安全序列!
不予分配!
*****************银行家算法演示*****************
                1:分配资源
                2:显示分配情况
                0:离开
************************************************
请选择功能号:
```

- 输入2，显示分配情况

```
请选择功能号:2
系统目前可用的资源[Available]:
a       b       c
2       3       0
        Max  Allocation  Need
进程    a b c   a b c   a b c
0       7 5 3   0 1 0   7 4 3
1       3 2 2   3 0 2   0 2 0
2       9 0 2   3 0 2   6 0 0
3       2 2 2   2 1 1   0 1 1
4       4 3 3   0 0 2   4 3 1
*****************银行家算法演示*****************
                1:分配资源
                2:显示分配情况
                0:离开
************************************************
请选择功能号:
```

- 输入3，离开程序；

```
请选择功能号:3
已离开!
PS D:\Codes>
```

实验总结

编写完这个实验后，对银行家算法加深了理解，深刻学习到了操作系统在为进程分配资源的过程；

为了实现银行家算法，必须设置若干数据结构，并且要先检测系统是否处于安全状态。