操作系统

实验报告

实验名称： 演示银行家算法

院系： 计算机学院·网络空间安全学院

班级：

学号：

姓名：

指导老师：匡林爱

日期：2020年11月3日

1. **实验目的和要求：**

1．设计并实现银行家算法的程序，以教科书的例题为例子，演示银行家算法。目的是加深对银行家算法的理解。

2．修改程序使之可以解决任意有关银行家算法的问题。

1. **实验内容：**

### 数据结构

1．可利用资源向量Available

是一个含有m个元素的数组，其中每一个元素代表一类可用资源数目，m是资源种类数。

2．最大需求矩阵Max

是一个n×m矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求数。每一行对应一个进程；每一列对应一类资源。

3．分配矩阵Allocation

是一个n×m矩阵，它定义了系统中每一类资源当前已分配给每一个进程的资源数。

4．需求矩阵Need

是一个n×m矩阵，用于表示每个进程尚需的各类资源数。

1. **实验步骤：**

在Windows XP + VC++ 6.0环境下设计程序，采用面向对应程序设计(OOP)技术。

**1．银行家算法函数Banker()**

银行家算法函数是本实验设计的重要内容，其程序流程图如图一所示

获得进程Pi的资源请求向量Request

Request≤Need[i]?

Request≤Available [i]?

系统试探着把资源分配给进程Pi

Y

Y

出错

Pi等待

N

N

修改Available、Need

和Allocation

调用安全性检查函数

系统状态仍是安全?

返回

将本次资源分配作废，恢复原来的资源分配状态

N

真正将资源分配给进程Pi

返回

**图一 银行家算法流程图**

1. **安全性算法函数**

安全性算法函数流程图如图二所示。

设置工作向量Work = Available

标志向量Finish，其初值为：

Finish[i]=false (i=1,2,…,n)

寻找一个进程Pi

Finish[i]=False and

Need[i]≤Work?

进程Pi可获得所需全部资源，可执行结束并释放分配给它的资源。

令：Work[j]+ = Allocation[i, j]

Finish[i] = True

N

Y

所有Finish[i] = True?

返回“安全”

返回

“不安全”

N

Y

**图二 安全性算法流程图**

1. **实验源程序**

**banker.cpp**

#include <iostream>

using namespace std;

class Banker {

int \*Allocation; // 分配矩阵

int \*Need; // 尚需矩阵

int \*Available; // 可用资源向量

int \*squeue; // 安全序列

int safe; // safe=1,存在安全序列

public:

int m, n; // m为资源种类数，n为进程数

Banker(int, int); // 构造函数

~Banker(); // 析构函数

bool Ifsafe(); // 安全性检查函数

int banker(int k, int \*Request); // 银行家算法

void Print(); // 输出资源分配情况

void Getmn(int &, int &); // 获取资源种类数m和进程数n

};

Banker::Banker(int m0, int n0) // 构造函数

{

int i, j;

m = m0; n = n0; safe = 0;

Allocation = new int[n\*m];

Need = new int[n\*m];

Available = new int[m];

squeue = new int[n];

cout << "输入分配矩阵Allocation：\n";

for (i = 0; i < n; i++)

{

cout << "输入分配矩阵的第 " << i + 1 << " 行：";

for (j = 0; j < m; j++)

cin >> Allocation[i\*m + j];

}

cout << "\n输入尚需矩阵Need：\n";

for (i = 0; i < n; i++)

{

cout << "输入尚需矩阵的第 " << i + 1 << " 行：";

for (j = 0; j < m; j++)

cin >> Need[i\*m + j];

}

cout << "\n输入可用资源向量Available：";

for (i = 0; i < m; i++)

cin >> Available[i];

}

Banker::~Banker()

{

delete[]Allocation; // 释放分配矩阵空间

delete[]Need; // 释放尚需矩阵空间

delete[]Available; // 释放可用资源向量

delete[]squeue; // 释放安全序列空间

}

// 判断系统状态是否安全。Finish:标志；Work:工作向量。

bool Banker::Ifsafe()

{

int \*Work = new int[m]; // 定义工作向量

int \*Finish = new int[n]; // 定义标志数组

int i, j, flag, k = 0;

for (i = 0; i < n; i++)

Finish[i] = 0; // 置未标记标志

for (j = 0; j < m; j++)

Work[j] = Available[j];

do {

flag = 0;

for (i = 0; i < n; i++)

{

if (Finish[i] == 0)

{

for (j = 0; j < m; j++) // 判断Need[i]是否不超过Work

if (Need[i\*m + j] > Work[j]) break;

if (j == m) { // 若Pi的需求Need[i]≤当前可用资源能Work

Finish[i] = 1; // 置Pi能完成的标志

flag = 1; // 为了继续循环(查找)

squeue[k++] = i; // 顺便记下安全序列

for (j = 0; j < m; j++)//Pi完成后，它所占用的资源归入Work中

Work[j] += Allocation[i\*m + j];

}

}

}

} while (flag);

delete[]Work;

for (i = 0; i < n; i++)

if (Finish[i] == 0) // 若有进程未完成，则系统状态不安全

{

delete[]Finish;

return false; // 返回“不安全”

}

delete[]Finish;

return true; // 返回“安全”

}

// 银行家算法。k:进程序号; Request:该进程的请求向量

int Banker::banker(int k, int \*Request)

{

int i;

for (i = 0; i < m; i++)

if (Request[i] > Need[k\*m + i])

{

cout << "\nError : Request of P" << k << " > Max[" << k << "]\n";

return 0;

}

for (i = 0; i < m; i++)

if (Request[i] > Available[i])

{

cout << "\nRequest > Available,P" << k << " is blocked\n";

return 0;

}

for (i = 0; i < m; i++) // 系统试探分配资源

{

Available[i] -= Request[i];

Allocation[k\*m + i] += Request[i];

Need[k\*m + i] -= Request[i];

}

safe = Ifsafe(); // 判断是否安全

if (safe) // 若系统状态仍是安全的

{

cout << "\n满足进程P" << k << "的请求(";

for (i = 0; i < m - 1; i++)

cout << Request[i] << ',';

cout << Request[i] << ")后，系统状态仍然是安全的。安全序列为：\n";

for (i = 0; i < n - 1; i++)

cout << 'P' << squeue[i] << ',';

cout << 'P' << squeue[i] << endl;

}

else // 若系统状态已是不安全

{

cout << "\n如果满足进程P" << k << "的资源请求(";

for (i = 0; i < m - 1; i++)

cout << Request[i] << ',';

cout << Request[i] << ")，系统将进入不安全状态，" << endl;

cout << "故系统将拒绝该进程的请求。" << endl;

for (i = 0; i < m; i++) // 将试探分配作废

{

Available[i] += Request[i];

Allocation[k\*m + i] -= Request[i];

Need[k\*m + i] += Request[i];

}

}

return safe;

}

void Banker::Print() // 输出当前系统的资源分配情况

{

int i, j;

cout << "\n Allocation Need Available\n\n";

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

cout.width(3);

cout << Allocation[i\*m + j];

}

cout << '\t';

for (j = 0; j < m; j++)

{

cout.width(3);

cout << Need[i\*m + j];

}

if (i == 0)

{

cout << "\t";

for (j = 0; j < m; j++)

{

cout.width(3);

cout << Available[j];

}

}

cout << endl;

}

cout << endl;

}

void Banker::Getmn(int &x, int &y)

{

x = m;

y = n;

}

int main()

{

int i, stat;

int k, m, n; // m,n分别是资源种类数和进程数

Banker b(3, 5); // 定义对象b，其资源种类数和进程数分别为3和5

b.Getmn(m, n);

int \*Request = new int[m]; // 进程申请资源向量

b.Print(); // 显示当前资源分配情况

stat = b.Ifsafe(); // 判断T0时刻状态是否安全

if (stat)

cout << "当前系统状态是安全的" << endl;

else {

cout << "当前系统状态是不安全的" << endl;

delete[]Request;

return 0;

}

while (1) // 当输入的进程号出错时，循环结束

{

cout << "\n请输入进程号 (0--" << n - 1 << ", -1=Exit) : ";

cin >> k;

if (k >= n)

{

cout << "\nError : The number of Process is out of range !" << endl;

continue;

}

if (k < 0) break;

cout << "请输入进程P" << k << "申请的资源数: ";

for (i = 0; i < m; i++)

cin >> Request[i];

stat = b.banker(k, Request);

b.Print();

}

delete[]Request;

return 1;

}

1. **实验结果**



1. **实验总结：**

**通过这次实验，我了解并掌握运用了银行家算法，理解了该算法的内在逻辑结构，同时对进程资源分配的问题有了更深的领悟，如我觉得这次实验的核心是通过设置工作向量和标志向量来对是否可以分配资源进行一个判断，同时设置请求向量通过尝试分配之后调用安全性检查函数来作判断，之后根据判断结果决定是否分配资源，总而言之，这次实验让我能够理解和运用一种新的算法，对我的帮助很大。**