# 实验二 银行家算法

# 专业 软件工程 姓名 毛济洲 学号 20182344050

## 一．实验目的

## 1、 理解银行家算法。

## 2、 掌握进程安全性检查的方法与资源分配的方法。

## 二．实验内容

### 编写调试模拟银行家算法的程序，并以下面给出的例子验证所编写的程序的正确性。

### 背景知识

银行家算法：

银行家算法最初原为银行系统设计，以确保银行在发放现金贷款时，不会发生不能满足 所有客户需要的情况。在 OS 设计中，也可以用它来避免死锁。 为实现银行家算法，每个新进程在进入系统时它必须申明在运行过程中，可能需要的每 种资源类型的最大单元数目，其数目不应超过系统所拥有的资源总量。当某一进程请求时， 系统会自动判断请求量是否小于进程最大所需，同时判断请求量是否小于当前系统资源剩余 量。若两项均满足，则系统试分配资源并执行安全性检查算法。

安全性检查算法 :

安全性检查算法用于检查系统进行资源分配后是否安全，若安全系统才可以执行此次分 配；若不安全，则系统不执行此次分配。 安全性检查算法原理为：在系统试分配资源后，算法从现有进程列表寻找出一个可执行 的进程进行执行，执行完成后回收进程占用资源；进而寻找下一个可执行进程。当进程需求 量大于系统可分配量时，进程无法执行。当所有进程均可执行，则产生一个安全执行序列， 系统资源分配成功。若进程无法全部执行，即无法找到一条安全序列，则说明系统在分配资 源后会不安全，所以此次分配失败。

## 三．实验步骤

**（1）       任务分析：**

   以下是创建进程的关键知识：

   进程创建和撤销需要调用CreateProcess（）和TerminateProcess()。CreateProcess()实际上是运行一个新的可执行程序，这里，需要设置其必备的参数。而终止进程则首先是要找到运行进程的句柄，然后终止该进程。

## 四．实验总结

## 银行家算法中的数据结构

## 1）可利用资源向量Available

## 是个含有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数目。如果Available［j］=K，则表示系统中现有Rj类资源K个。

## 2）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，则表示进程i需要Rj类资源的最大数目为K。

## 3）分配矩阵Allocation

## 这也是一个n×m的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果Allocation［i,j］=K，则表示进程i当前已分得Rj类资源的数目为K。

## 4）需求矩阵Need。

## 这也是一个n×m的矩阵，用以表示每一个进程尚需的各类资源数。如果Need［i,j］=K，则表示进程i还需要Rj类资源K个，方能完成其任务。

## Need［i,j］=Max［i,j］-Allocation［i,j］

## 银行家算法

## 设Requesti是进程Pi的请求向量，如果Requesti［j］=K，表示进程Pi需要K个Rj类型的资源。当Pi发出资源请求后，系统按下述步骤进行检查：

## (1)如果Requesti［j］≤Need［i,j］，便转向步骤(2)；否则认为出错，因为它所需要的资源数已超过它所宣布最大值。

## (2)如果Requesti［j］≤Available［j］，便转向步骤(3)；否则，表示尚无足够资源，Pi须等待。

## (3)系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：

## Available［j］=Available［j］-Requesti［j］;

## Allocation［i,j］=Allocation［i,j］+Requesti［j］;

## Need［i,j］=Need［i,j］-Requesti［j］;

## 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。若安全，才正式将资源分配给进程Pi，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程Pi等待。

## 安全性算法

## 1）设置两个向量：

## 工作向量Work: 它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work=Available;

## 工作向量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。开始时先做Finish［i］=false; 当有足够资源分配给进程时， 再令Finish［i］=true。

## 2）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,j］≤Work［j］；若找到，执行 (3)，否则，执行 (4)

## 3）当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：

## Work［j］=Work［i］+Allocation［i,j］;

## Finish［i］=true;

## go to step 2;

## 4）如果所有进程的Finish［i］=true都满足， 则表示系统处于安全状态；否则，系统处于不安全状态;

## 五．附录源代码

#include <iostream>

using namespace std;

//全局变量定义

int Available[100]; //可利用资源向量Available ,是个含有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数目。如果Available[j] = K，则表示系统中现有Rj类资源K个。

int Max[50][100]; //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，则表示进程i需要Rj类资源的最大数目为K。

int Allocation[50][100]; //已经分配矩阵

int Need[50][100]; //进程需求矩阵

int Request[50][100]; //M个进程还需要N类资源的资源量

int Finish[50];//

int p[50];

int m, n; //M个进程,N类资源

//安全性算法

int Safe()

{

int i, j, l = 0;

int Work[100]; //可提供给进程各类资源资源数组

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

Work[i] = Available[i];//在执行安全算法开始时，可提供的各类资源数目=系统现有各类资源数目;Work=Available;

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

Finish[i] = 0;//表示系统是否有足够的资源分配给进程

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

{

if (Finish[i] == 1)//工作向量等于1,进程执行

continue;

else

{

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

{

if (Need[i][j] > Work[j])

break;

}

if (j == n)

{

Finish[i] = 1;

for (int k = 0; k < n; k++)

Work[k] = Work[k] + Allocation[i][k];//直至完成，并释放出分配给它的资源，故应执行可提供资源数量更新

p[l++] = i;

i = -1;

}

else continue;

}

if (l == m)

{

cout << "系统是安全的" << '\n';

cout << "系统安全序列是:\n";

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

{

cout << p[i];

if (i != l - 1)

cout << "-->";

}

cout << '\n';

return 1;

}

}

return 0;

}

//银行家算法

int main()

{

int i, j, mi;

cout << "输入进程的数目:\n";

cin >> m;

cout << "输入资源的种类:\n";

cin >> n;

cout << "输入每个进程最多所需的各类资源数,按照" << m << "x" << n << "矩阵输入\n";

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

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

cin >> Max[i][j];//将用户输入的资源需求放进最大需求矩阵数组里

cout << "输入每个进程已经分配的各类资源数,按照" << m << "x" << n << "矩阵输入\n";

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

{

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

{

cin >> Allocation[i][j];//分配矩阵

Need[i][j] = Max[i][j] - Allocation[i][j];//进程需要的各类资源数=进程最大需求资源数-已分配资源数

if (Need[i][j] < 0)

{

cout << "你输入的第" << i + 1 << "个进程所拥有的第" << j + 1 << "个资源错误，请重新输入:\n";

j--;

continue;

}

}

}

cout << "请输入各个资源现有的数目:\n";

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

cin >> Available[i];

Safe();

while (1)

{

cout << "输入要申请的资源的进程号：（第一个进程号为0，第二个进程号为1，依此类推）\n";

cin >> mi;

cout << "输入进程所请求的各个资源的数量\n";

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

cin >> Request[mi][i];

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

{

if (Request[mi][i] > Need[mi][i])

{

cout << "所请求资源数超过进程的需求量！\n";

return 0;

}

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

{

cout << "所请求资源数超过系统所有的资源数！\n";

return 0;

}

}

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

{

Available[i] = Available[i] - Request[mi][i];//可利用资源=可利用资源-进程需要

Allocation[mi][i] = Allocation[mi][i] + Request[mi][i];//已分配资源=已分配资源+进程需要

Need[mi][i] = Need[mi][i] - Request[mi][i];//进程还需资源=进程还需资源-进程需要

}

if (Safe())

cout << "同意分配请求\n";

else

{

cout << "对不起.你的请求被拒绝…\n";

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

{

Available[i] = Available[i] - Request[mi][i];//可利用资源=可利用资源-进程需要

Allocation[mi][i] = Allocation[mi][i] + Request[mi][i];//已分配资源=已分配资源+进程需要

Need[mi][i] = Need[mi][i] - Request[mi][i];//进程还需资源=进程还需资源-进程需要

}

}

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

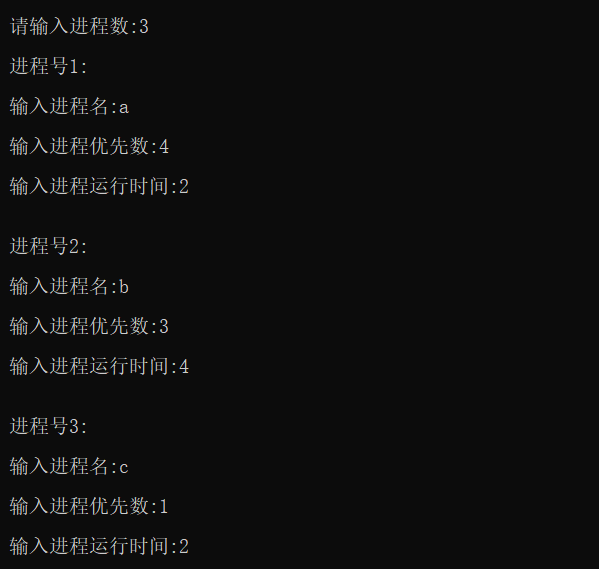
Finish[i] = 0;

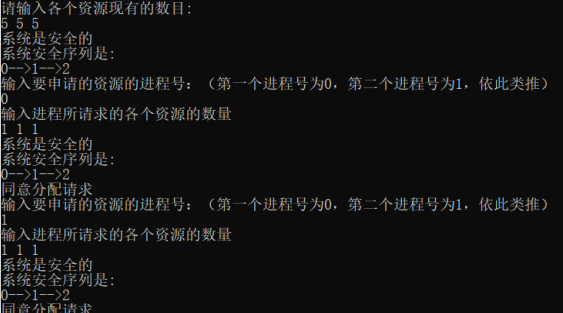
}

return 0;

}

## 六、调试分析





# 六.实验小结：

1.银行家算法是一种用来避免操作系统死锁出现的有效算法。

2.死锁：是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

3.死锁的发生必须具备以下四个必要条件：

1）互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放。

2）请求和保持条件：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。

3）不抢占条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

4）循环等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，···，Pn}中的P0正在等待一个P1占用的资源；P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源。

4.避免死锁算法中最有代表性的算法就是Dijkstra E.W 于1968年提出的银行家算法，银行家算法是避免死锁的一种重要方法，防止死锁的机构只能确保上述四个条件之一不出现，则系统就不会发生死锁。

5.为实现银行家算法，系统必须设置若干数据结构，同时要解释银行家算法，必须先解释操作系统安全状态和不安全状态。

6.安全序列:是指一个进程序列{P1，…，Pn}是安全的，即对于每一个进程Pi(1≤i≤n），它以后尚需要的资源量不超过系统当前剩余资源量与所有进程Pj (j < i )当前占有资源量之和。7.安全状态：如果存在一个由系统中所有进程构成的安全序列P1，…，Pn，则系统处于安全状态。

8.安全状态一定是没有死锁发生。不安全状态：不存在一个安全序列。不安全状态不一定导致死锁。