华侨大学

操作系统实验报告

实验名称：预防进程死锁的银行家算法

班 级： 21软件工程

班 号： 1

学 号： 2125121035

姓 名： 林梓豪

2023年11月25 日

# 1、实验目的

通过这次实验，加深对进程死锁的理解，进一步掌握进程资源的分配、死锁的检测和安全序列的生成方法。

# 2、实验内容

**问题描述：**

设计程序模拟预防进程死锁的银行家算法的工作过程。假设有系统中有n个进程P1, … ,Pn，有m类可分配的资源R1, … ,Rm，在T0时刻，进程Pi分配到的j类资源为Allocationij个，它还需要j类资源Need ij个，系统目前剩余j类资源Workj个，现采用银行家算法进行进程资源分配预防死锁的发生。

程序要求如下：

1）判断当前状态是否安全，如果安全，给出安全序列；如果不安全给出理由。

2）对于下一个时刻T1，某个进程Pk会提出请求Request(R1, … ,Rm)，判断分配给P k进程请求的资源之后。

3）输入：进程个数n，资源种类m，T0时刻各个进程的资源分配情况（可以运行输入，也可以在程序中设置）；

4）输出：如果安全输出安全的进程序列，不安全提示信息。

**实现提示：**

用C++语言实现提示：

1）程序中进程调度时间变量描述如下：

int Available[MaxNumber];

int Max[MaxNumber][MaxNumber];

int Allocation[MaxNumber][MaxNumber];

int Need[MaxNumber][MaxNumber];

int Request[MaxNumber];

int SafeOrder[MaxNumber];

2）进程调度的实现过程如下：

* 变量初始化；
* 接收用户输入n，m，（输入或者默认的）Allocationij，Need ij；
* 按照银行家算法判断当前状态安全与否，安全给出安全序列，不安全给出提示；
* 如果安全，提示用户输入下一时刻进程Pk的资源请求Request(R1, … ,Rm)；
* 如果不安全或者无新请求则退出。

**实验要求：**

1)上机前认真复习银行家算法，熟悉资源分配和安全检查过程；

2)上机时独立编程、调试程序；

3)根据具体实验要求，完成好实验报告（包括实验的目的、内容、要求、源程序、实例运行结果截图）。

# 3、实验开发环境

|  |  |
| --- | --- |
| 操作系统 | Windows 11 |
| 开发ide | Visual studio 2017 |

# 4、程序主要构成部分及其算法说明

（1）定义进程结构体

int Available[3] = { 3,3,2 };//系统可用资源

int Max[5][3] = { {7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3} };//进程最大资源需求量

int Allocation[5][3] = { {0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2} };//系统已经给进程分配的资源

int Need[5][3] = { {7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1} };//进程的资源最大需求量

int Work[3];//系统可提供给进程继续运行所需要的各类资源数目

string Finish[5] = { "false" ,"false" , "false" , "false" , "false" };

int SafeArray[5];//安全序列

int Request[3];//请求资源向量

1. 处理进程发出的资源请求的算法

* 若Requesti[j]＞Need[i,j]，产生出错条件，因为进程Pi对资源的请求量已超过其说明的最大数量；否则，转到步骤2。
* 如果Requesti[j]＞Available[j]，则进程Pi必须等待，这是因为系统现在没有可用的资源；否则，转到步骤3。
* 如果系统可以给进程Pi分配所请求的资源，则应对有关数据结构进行修改：

Available[j] = Available[j]-Requesti[j]; (j=1,2,……,n)

Allocation[i,j] = Allocation[i,j] + Requesti[j]; (i=1,2,……,m)

Need[i,j] = Need[i,j] - Requesti[j];

* 系统执行安全性检查，查看此时系统状态是否安全。如果安全，就给进程Pi 实际分配资源；否则，即系统是不安全的，则Pi等待，作废本次试探性分配，并且把资源分配状态恢复成3之前的情况。

int RequestLessNeed(int n) {

for (int i = 0; i < 3; i++) {

if (Request[i] > Need[n][i]) {

return 0;

}

}

return 1;

}

int RequestLessAvailable(int n) {

for (int i = 0; i < 3; i++) {

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

return 0;}

}

return 1;

}

void RequestResourse() {

cout << "请输入发出资源请求的进程：";

int n;

cin >> n;

cout << "请依次输入所请求的资源数量：";

for (int i = 0; i < 3; i++) {

cin >> Request[i];

}

if (RequestLessNeed(n)) {

if (RequestLessAvailable(n)) {

for (int i = 0; i < 3; i++) {

Allocation[n][i] += Request[i];

Available[i] -= Request[i];

Need[n][i] -= Request[i];

}

SafeCheck();

}

else {

cout << "P" << n << "请求的资源向量已超过系统可用资源向量，请求失败！让其继续等待..." << endl;

cout << "-------------------------------------------------------------------" << endl;

return;

}

}

else {

cout << "P" << n << "请求的资源向量已超过其最大需求向量，请求失败！让其继续等待..." << endl;

cout << "-------------------------------------------------------------------" << endl;

return;

}

}

（3）安全性检查算法

* 安全性检查算法是银行家算法的核心，一般会有某个进程发出一个资源请求向量，我们只需要执行上面银行家算法的前三步，就会得到一个更新后的Allocation和Need矩阵，再按照上例的安全性算法进行判断，此时系统是否处于安全状态，就能直到系统是否能立即满足该进程提出的资源请求。

int NeedLessWork(int n) {

for (int i = 0; i < 3; i++) {

if (Work[i] < Need[n][i]) {

return 0;

}

}

return 1;

}

void ShowSafe(int i) {

cout << "P" << i << "\t";

for (int j = 0; j < 3; j++) {

cout << Work[j] << " ";

}

cout << "\t\t";

for (int j = 0; j < 3; j++) {

cout << Need[i][j] << " ";

}

cout << "\t\t";

for (int j = 0; j < 3; j++) {

cout << Allocation[i][j] << " ";

}

cout << "\t\t";

for (int j = 0; j < 3; j++) {

cout << Work[j] + Allocation[i][j] << " ";

}

cout << "\t\t";

cout << Finish[i];

cout << endl;

}

void IsSafe(int n) {

for (int i = 0; i < 5; i++) {

if (Finish[i] == "false" && NeedLessWork(i)) {

Finish[i] = "true";

SafeArray[n] = i;

n++;

ShowSafe(i);

for (int j = 0; j < 3; j++) {

Work[j] += Allocation[i][j];

}

break;

}

}

int mult = 1;

//如果五个标志都为1即都已经完成，则打印安全序列，否则继续执行安全性检查算法

for (int k = 0; k < 5; k++) {

if (Finish[k] == "false")

mult \*= 0;

else

mult \*= 1;

}

if (mult == 0) {

IsSafe(n);

}

else {

SafeLine();

}

}

void SafeCheck() {

cout << "试探着将资源分配给它后，系统安全情况分析如下：" << endl;

cout << "进程\tWork\t\tNeed\t\tAllocation\tWork+Allocation\tFinish" << endl;

for (int i = 0; i < 3; i++) {

Work[i] = Available[i];

}

IsSafe(0);

}

（4）输出模块

void Show() {

cout << "T0时刻的系统资源分配情况如下：" << endl;

cout << "进程名\tMax\t\tAllocation\tNeed\t\tAvailable" << endl;

for (int i = 0; i < 5; i++) {

cout << "P" << i << "\t";

for (int j = 0; j < 3; j++) {

cout << Max[i][j] << " ";

}

cout << "\t\t";

for (int j = 0; j < 3; j++) {

cout << Allocation[i][j] << " ";

}

cout << "\t\t";

for (int j = 0; j < 3; j++) {

cout << Need[i][j] << " ";

}

cout << "\t\t";

if (i == 0) {

for (int j = 0; j < 3; j++) {

cout << Available[j] << " ";

}

}

cout << endl;

}

}

void SafeLine() {

cout << "当前系统处于安全状态..." << endl;

cout << "其中一个安全序列为：";

for (int i = 0; i < 5; i++) {

if (i == 4)cout << "P" << SafeArray[i];

else cout << "P" << SafeArray[i] << "-->";

}

cout << endl;

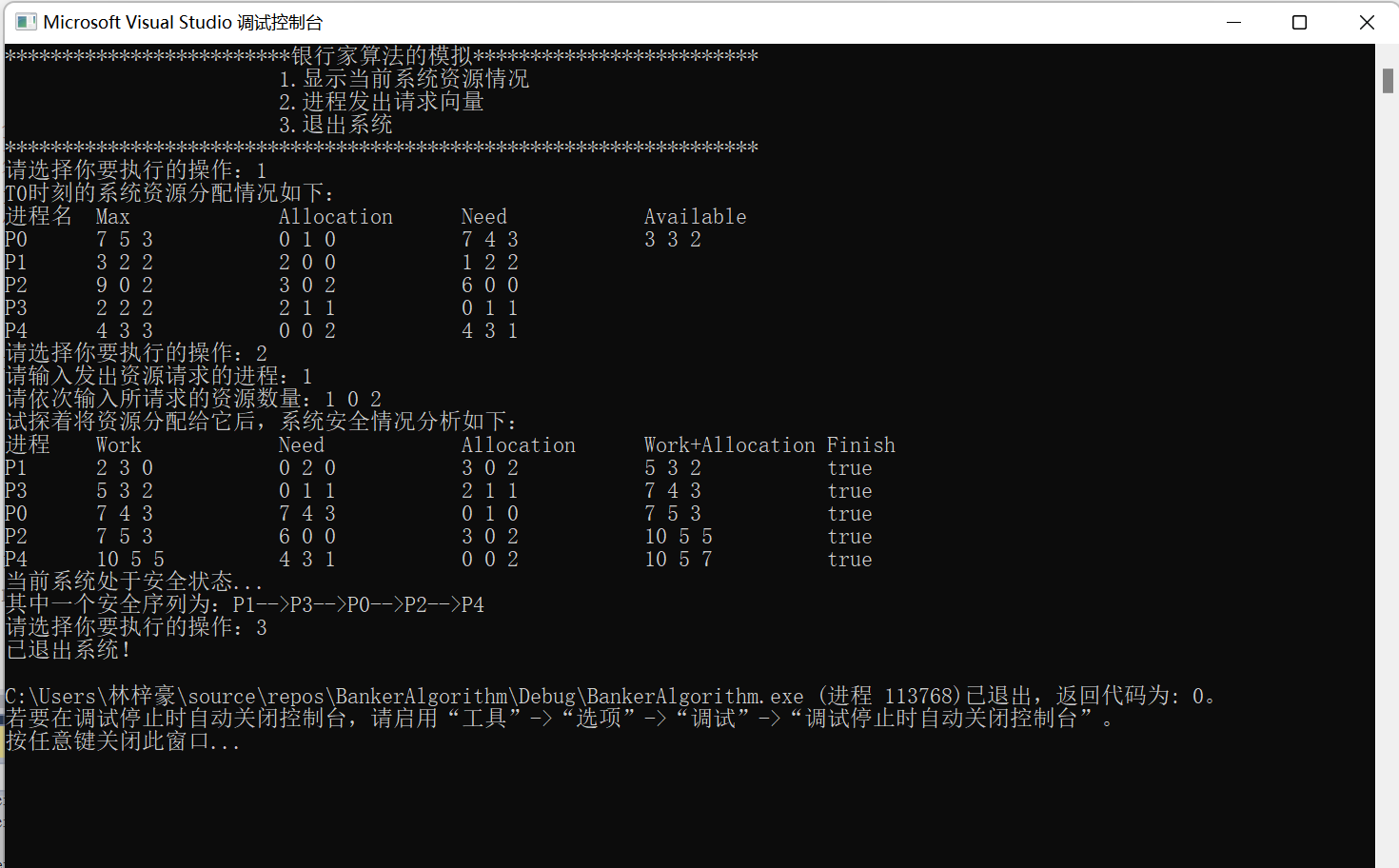
for (int i = 0; i < 5; i++) {

Finish[i] = "false";

}

}

# 5、调试过程和运行结果



# 6、实验收获体会

在预防进程死锁的银行家算法的操作系统实验中，我对死锁现象及其预防有了更深入的了解。以下是我的一些心得体会：

1. 死锁现象的严重性：死锁会导致系统中的进程无法继续执行，进而影响整个系统的正常运行。因此，了解死锁产生的原因和如何预防死锁具有重要意义。

2. 银行家算法的原理：银行家算法通过预先定义系统的资源分配策略，以确保系统始终处于安全状态。在进行资源分配时，算法会检查分配后的系统是否仍然安全。如果安全，则进行实际分配；如果不安全，则撤销预分配，拒绝进程的资源请求。这种算法可以有效地预防死锁的发生。

3. 实验过程的挑战性：实现银行家算法需要对进程和资源的动态分配进行精确的控制。在实验过程中，我需要深入理解算法原理，分析进程的资源需求，合理设置资源向量和矩阵，并编写相应的程序代码。这对我理解操作系统底层原理和编程技能都提出了较高要求。

4. 实验成果的成就感：通过实验，我成功实现了银行家算法，可以正确判断进程的资源请求是否会导致死锁。当进程请求资源时，系统可以根据预先设定的规则进行安全分配。这让我感受到了解决问题的喜悦，提高了我对操作系统的兴趣和理解。

5. 对现实应用的启示：银行家算法不仅可以应用于操作系统中的资源分配，还可以应用于其他领域，如项目管理、任务调度等。通过预先制定合理的规则，并在执行过程中不断检查和调整，可以有效避免问题的发生。

6. 深入研究的方向：实验让我认识到，预防死锁的方法不止一种，还有其他算法如资源分配图算法等。未来可以继续研究这些算法，以提高我对死锁预防的理解和应用能力。

总之，在预防进程死锁的银行家算法实验中，我不仅提高了自己的编程技能，还对操作系统底层原理和死锁预防有了更深入的认识。这将对我未来的学习和研究工作产生积极影响。

# 7、改进意见

（1）异常处理：在处理进程资源请求时，添加异常处理机制，以应对可能的输入错误。例如，检查进程编号是否在范围内，确保输入的资源数量合法等。

（2）代码优化：优化代码性能，例如，在循环中使用局部变量，减少全局变量的访问次数。