实验三 进程的死锁避免算法

（银行家算法实现）

1. 实验目的

根据银行家算法的思想，编写程序，解决并发进程的死锁问题。

1. 实验内容
2. 进程的死锁避免算法。编写一段程序，模拟银行家算法，解决进程的死锁问题。
3. 利用VC++6.0实现上述程序设计和调试操作，根据提示输入相应的资源请求，对于算法操作的成功与否提供一定的提示框。
4. 通过阅读和分析实验程序，熟悉进程的死锁问题。
5. 实验步骤
6. 程序设计

本实验要求设计并实现银行家算法。

银行家算法是死锁避免的经典算法，其核心思想是：进程动态地申请资源，每次申请资源时系统都执行安全状态检查算法判断本次申请是否会造成系统处于不安全状态，如果不安全则阻塞进程；如果安全状态，则完成资源分配。

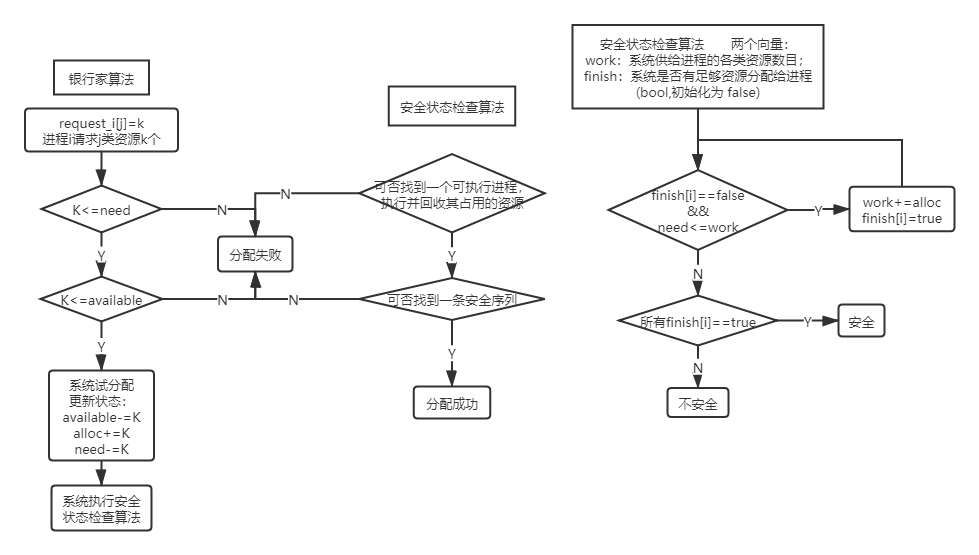
安全状态检查算法的思想是找到一个安全序列，使所有进程都能执行完毕。如果找到，则处于安全状态，否则为不安全状态。

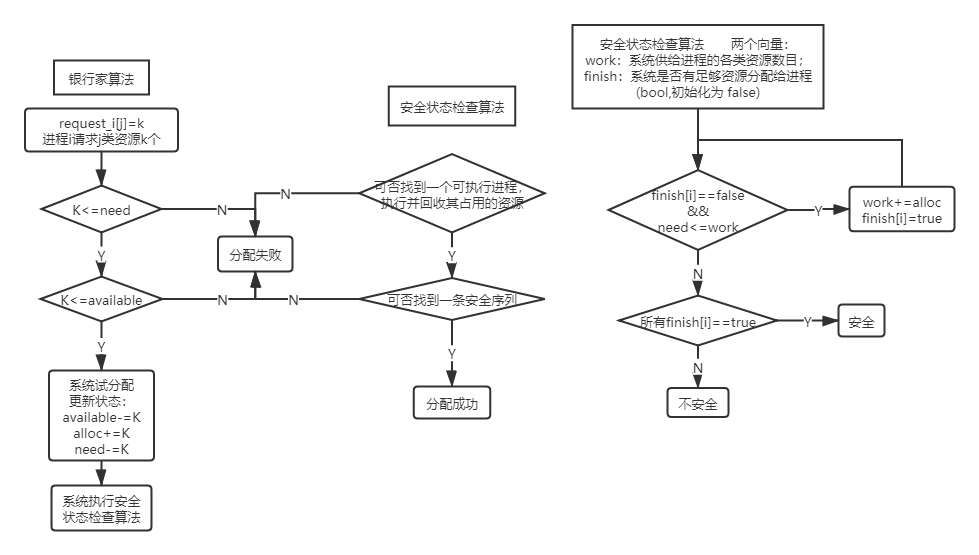
思路：

1)进程一开始向系统提出最大需求量.

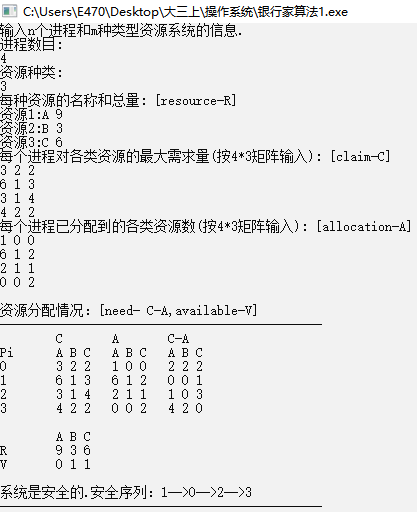
2)进程每次提出新的需求都统计是否超出它事先提出的最大需求量.

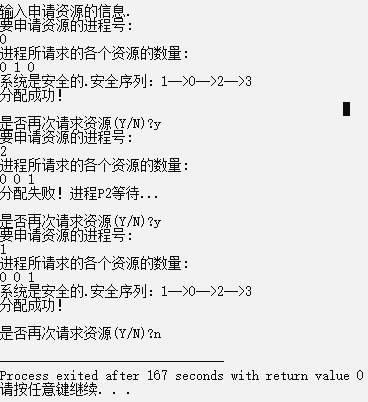
3)若正常,则判断该进程所需剩余量(包括本次申请)是否超出系统所掌握的剩余资源量,若不超出则执行安全性算法检测，判断在此次资源分配后，系统是否处于安全状态，若安全则分配,否则恢复原来的资源分配状态，让进程等待(阻塞进程).





1. 执行结果





1. 主要程序结构（附注释）
2. 数据结构及符号说明

采用一位数组和二维数组作数据存储，定义如下：

const int N=50,M=100;

string Rname[M]; //资源名称

int resource[M]; //资源总量

int available[M]; //剩余

int claim[N][M]; //总需 n个进程 m种资源

int alloc[N][M]; //分配

int need[N][M]; //还需:need=claim-alloc

int finish[N]; //完成标志

int p[N]; //安全序列

int request[N][M]; //请求资源

1. 主要算法

//安全状态检查算法

bool safe (){

int work[M];//系统可提供的各类资源数目,用以保护原数据结构有关值

work=available;

finish=0;

while(i<n){

if (finish[i]==1) continue;

else{//满足finish[i]==0

if (need>work) break;

if (j==m){//满足need<=work

finish[i]=1;

work+=alloc;

p[l++]=i;

i=-1;//重置i 从头检查未执行的进程

}else continue;

}

if (安全序列装满 所有进程结束 ){

return true;

}

}

return false;

}

//银行家算法

if (request[pi][i]>need[pi][i]){

//error

}else if (request[pi][i]>available[i]){

//阻塞进程

}else{

//系统试分配 更新系统状态

available[i]-=request[pi][i];

alloc[pi][i]+=request[pi][i];

need[pi][i]-=request[pi][i];

}

if (safe()) //安全->分配资源

else //不安全->恢复原来的资源分配状态,让进程等待(阻塞进程)

1. 问题及解决方法

问题：请求资源后输出安全序列时少了最后一个

解决：if (safe(n,m)) cout<<"分配成功！\n";//这里原来写成safe(m,n)造成输出安全序列时少了最后一个 ，n-进程数，m-资源数，不能弄反了，注意细节！

1. 实验体会

通过本次实验，了解了银行家算法的模拟实现，同时因为理论是实践的基础，在实践的过程中也扎实了我的理论基础。加深了我对银行家算法的问题的理解。同时自行完成编程也一定程度上锻炼了我的自学能力和遇到问题解决问题的能力。