

**操作系统课程设计报告**

学院：

班级：

姓名：

学号：

指导教师：

1. **目的与要求**

了解系统的资源分配情况,操作系统在对进程所需资源进行分配的时候，如何有效的避免死锁问题，并能够用银行家算法实现对申请资源的进程找出一个最优的安全序列，以此来分配各类资源。掌握当进程所需资源得不到满足的时候，怎样确定一个阻塞队列，把这些进程的PCB合理放入。要求使用随机模拟分配算法来观察进程因为得不到资源而产生死锁的现象。

1. **主要内容**

设计一个可以设计4个并发进程共享10个同类资源的系统，各进程动态地申请和释放资源；用银行家算法和随机分配算法分别设计资源分配程序，观察系统的运行情况；

确定一组进程依次申请的资源数序列，运行上述程序，显示各进程依次申请的资源情况。

银行家算法破坏了产生死锁的环路等待条件，即不可能产生循环等待，从而可以避免死锁的发生，因此设计一个随机算法来方便观察分配资源是产生死锁的情况。

1. **设备与环境**

实现语言：Java

编辑工具：NetBeans IDE 7.0.1 编译环境：jdk1.7.0

系统环境：Windows 10

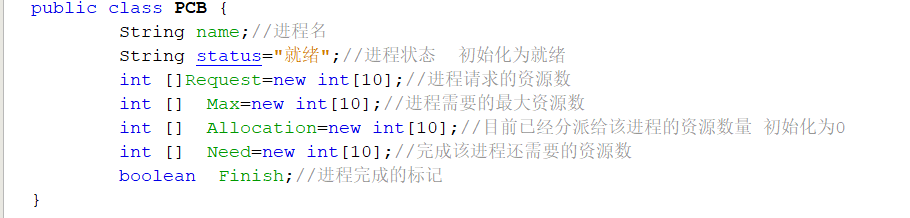
1. **实现过程**

系统设计了两大模块，一是对银行家算法的模拟，第二模块是随机分配模拟。

设计使用的数据结构：



代码实现：



实现流程图：



对银行家算法的实现描述：

使用的数据结构：

1. Available[m]，m表示系统内资源类别数，例如Available[j]=k，说明可用于分配的j类资源有k个。
2. Max[n,m]，n表示进程个数，m表示系统内资源类别数。若Max[i，j]=k。说明进程Pi最多可申请j类资源k个。
3. Allocation[n，m]，n，m同上。如果Allocation[i，j]=k，说明进程Pi已分配j类资源k个。
4. Need[n，m]，n，m同上。如果Need[i，j]=k，说明进程Pi尚缺j类资源k个。

Max[i，j]=Allocation[i，j]+Need[i，j]。

银行家算法具体实现：

把多个申请资源的进程保存到一个Vector矢量数组中，代表这些进程是同时在申请资源请求，对着这些进程进行依次处理，处理到当前进程所请求的资源数量的时候先执行一遍安全性检测，计算机资源满足当前进程的申请（判断满足的条件是Requesti<=needi 并且Requesti<=Available ），在具体代码中是调用的allocate(PCB p,int [] Available)进行检测，如果成功就进行资源尝试分配，然后执行安全性检测算法（调用Check\_Secrity .java类），只有找到一个安全序列，说明分配给该进程提出请求的资源后，至少有一个进程处于执行状态，不会陷入死锁状态，这时才真正的把进程申请的资源分配给此进程，若没有找到一个安全序列，则不进行资源分配（这也是使用银行家算法执行资源分配的时候不会产生死锁的原因）。不论是否分配成功，执行完对该进程的处理都会调用Display类输出所有进程的当前状态信息，在接着对Vector中的进程执行银行家算法。

随机分配具体实现：

执行随机分配模拟算法，多个进程进行提出资源请求的时候，当计算机资源满足的时候就直接进行分配，不会进行一些列安全检测，若再某一时刻，该进程提出资源请求，系统资源不能满足其执行需要的总资源，则进程直接处于死锁状态。只有当得到全部资源的进程执行完成，释放资源后，再判断该进程所需资源是否能够得到满足。

银行家算法流程图：

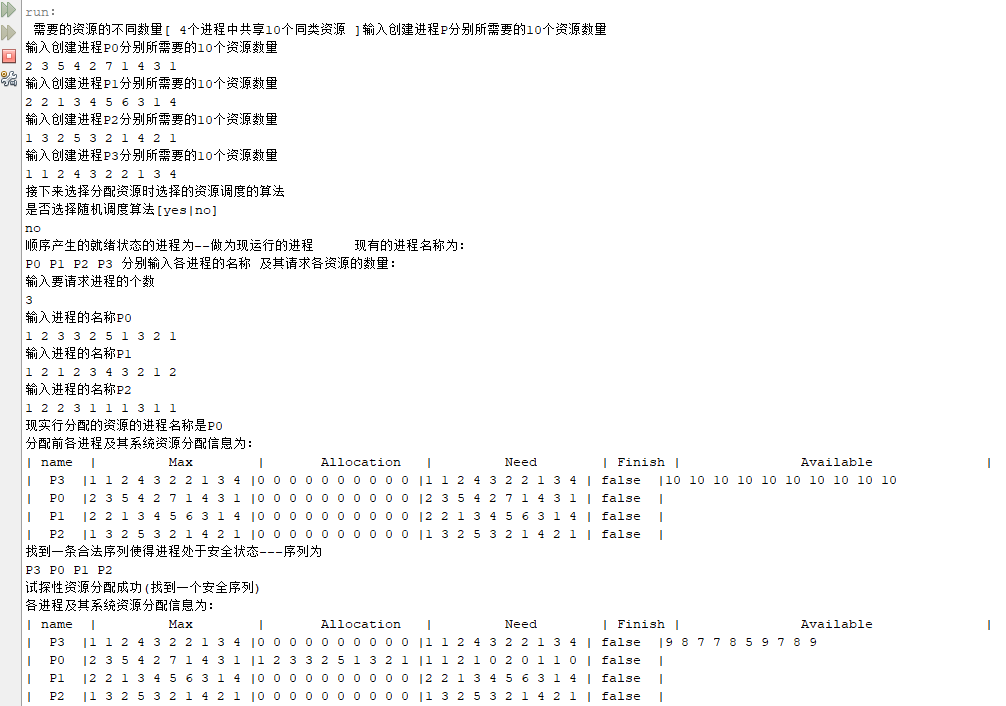


随机分配流程图：



**五、结果与分析**

运行结果截图：

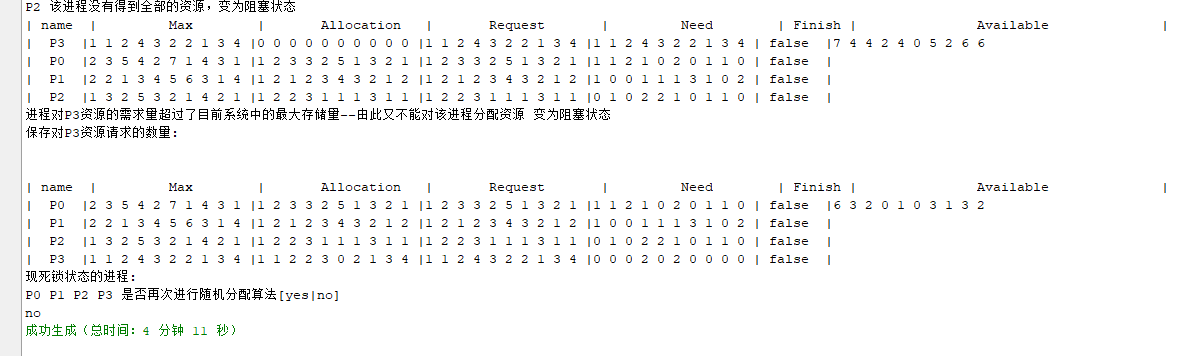
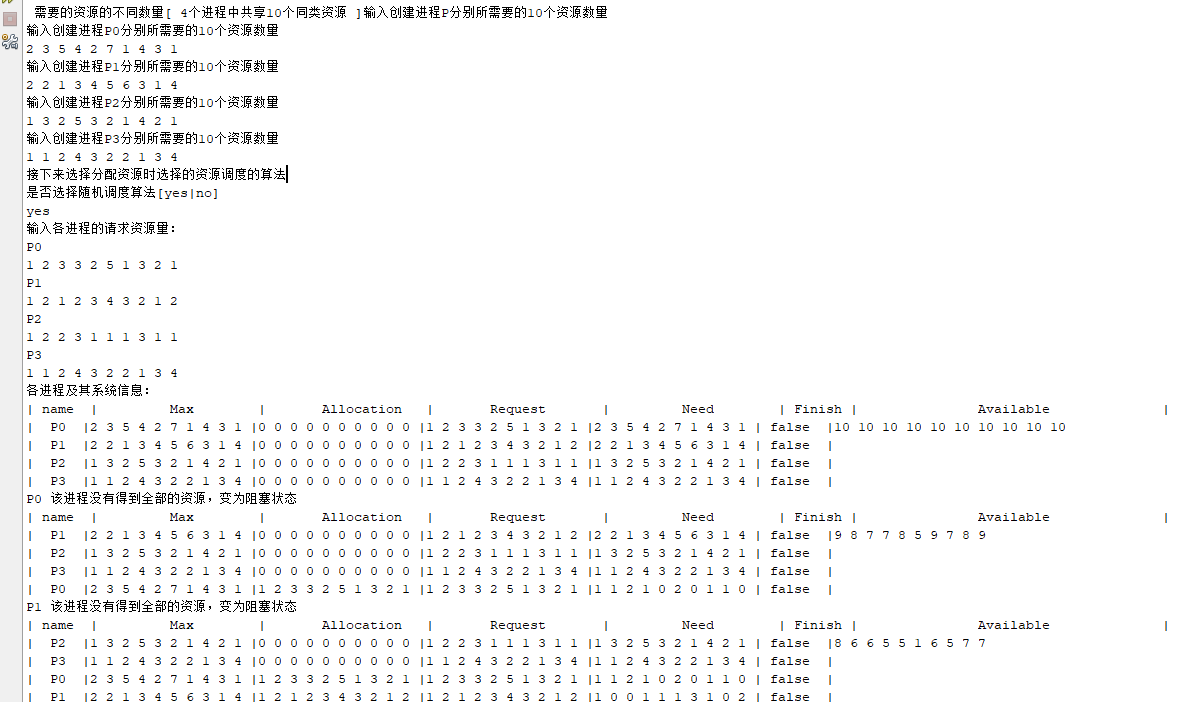




运行分析：

开始输入的是4个进程同时申请10类资源的数量，选择no的时候进入银行家算法，这些进程的PCB保存在Vector里面，会显示出哪些进程没有完成，然后输入请求资源进程的数量，在这儿实现了多个进程同时申请资源的情况，开始打印出的信息是所有进程的资源信息和系统内资源的情况，来方便的观察到当前进程申请资源进行应行家算法之前的分配境况，接着执行了安全性算法检测，找到一条安全序列会显示出来，若没有找到安全序列，则会显示出未找到这样的安全序列进行提示，系统也会收回之前尝试分配的资源。

随机算法：



选择yes的时候进入随机分配算法对申请资源的进程进行依次处理，对处理的进程满足所申请的资源就直接进行分配，否则陷入死锁状态，所有的进程处理完成之后，会显示出最终处于死锁进程的名称。

1. **心得体会**

本次操作系统的课程设计，通过银行家算法和随机模拟算法的设计与实现加深了我对进程之间的调度关系的理解，掌握了死锁的预防，避免，检测和解除的基本原理。 通过本次课程设计，锻炼了我们研究，设计，调试操作系统模块的能力，在互相学习共同努力的过程中，不断提高着自身的动手能力，从知识体系的完备性来讲，在以后如果有时间的话，从驱动程序开始写，做出一个完整的操作系统设计。

1. **附录**

代码：

**PCB.java:**

public class PCB {

String name;//进程名

String status="就绪";//进程状态 初始化为就绪

int []Request=new int[10];//进程请求的资源数

int [] Max=new int[10];//进程需要的最大资源数

int [] Allocation=new int[10];//目前已经分派给该进程的资源数量 初始化为0

int [] Need=new int[10];//完成该进程还需要的资源数

boolean Finish;//进程完成的标记

}

**NewMain.java:**

import java.io.\*;

import java.util.\*;

import java.util.Queue;

import java.util.LinkedList;

import java.util.Vector;

import java.lang.\*;

import java.util.Enumeration;

public class NewMain {

public static void main(String[] args) {

// TODO code application logic here

int [] Available={3,100,100,100,100,100,100,100,100,100};//定义系统内的资源初值

Scanner cin=new Scanner(System.in);

System.out.print(" 需要的资源的不同数量[ 4个进程中共享10个同类资源 ]");

PCB [] P=new PCB[4];

Vector<PCB> ve=new Vector<PCB>();//定义一个进程的就绪队列

int maxsource,num=0;

int total=100;//设置对应的10个资源的总数的数量都是100

System.out.println("输入创建进程P"+"分别所需要的10个资源数量");

for(int i=0;i<4;i++)

{

P[i]=new PCB();///现在是为他分配内存空间

P[i].name="P"+i;

System.out.println("输入创建进程P"+i+"分别所需要的10个资源数量");

for(int j=0;j<10;++j)

{

num=cin.nextInt();

while(num>Available[j])

{

System.out.println("该进程执行所对应资源需求总数超过了，计算机内的资源总数无法执行，重新输入"+(i+1)+"进程需要的第"+(j+1)+"个资源数");

num=cin.nextInt();

}

//记录在PCB中：

P[i].Max[j]=P[i].Need[j]=num;

}

ve.addElement(P[i]);

}

//选择是随机算法还是银行家算法程序

System.out.println("接下来选择分配资源时选择的资源调度的算法");

System.out.println("是否选择随机调度算法[yes|no]");

String tag=cin.next();

if("yes".equals(tag))

{

Random random=new Random(ve,Available);

while(true){

random.enter();

System.out.println("是否再次进行随机分配算法[yes|no]");

String str=cin.next();

if("yes".equals(str))

{

NewMain.main(null);

}

else

break;

}

}

else

{

Bank bank=new Bank(ve,Available);

bank.enter();

}

}

}

**Bank.java:**

import java.io.\*;

import java.util.\*;

import java.util.Queue;

import java.util.LinkedList;

public class Bank {

Vector<PCB> ve=new Vector<PCB>();//定义一个进程的就绪队列

private PCB p=new PCB();

int []Available=new int[10];

int [] Work=new int [10];

Try\_to\_allocate Try=new Try\_to\_allocate();//不分配空间就没有动态内存报错

Check\_Secrity cs;

Dispaly ds=new Dispaly();

public Bank( Vector<PCB> q,int []Available){

this.ve=q;

this.Available=Available;

}

Scanner cin=new Scanner(System.in);

void enter()

{

while(true) {

StringBuffer []name=new StringBuffer[4];///为数组分配空间

System.out.println("顺序产生的就绪状态的进程为--做为现运行的进程 现有的进程名称为：");

for(int i=0;i<ve.size();++i)

{

if(ve.get(i).Finish!=true)

System.out.print(ve.get(i).name+" ");

}

System.out.print("");

System.out.println("分别输入各进程的名称 及其请求各资源的数量：");

int num,n;

System.out.println("输入要请求进程的个数");

n=cin.nextInt();

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

{

System.out.print("输入进程的名称");

name[i]=new StringBuffer();//为每一个对象分配空间

name[i].append(cin.next());// =cin.nextLine();

for(int j=0;j<ve.size();++j)

{

p=ve.get(j);

if(p.name.equals(name[i].toString())==true)

{

ve.remove(j);

Try.Request\_function(p);//输入请求资源的数量并判断是否符合该进程的最大申请量

ve.addElement(p);

break;

}

}

}

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

{

int gg;

for( gg=0;gg<ve.size();++gg){

if(ve.get(gg).name.equals(name[tt].toString())==true)

{

p=ve.get(gg);

break;

}

}

System.out.println("现实行分配的资源的进程名称是"+p.name);

System.out.println("分配前各进程及其系统资源分配信息为：");

ds.display(ve, Available);

boolean tag=true;

for(int i=0;i<10;++i)

{

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

{

tag=false;

break;

}

}

if(tag==true){

Try.allocate(p,Available);

}

//安全性检验算法

cs=new Check\_Secrity(Available,ve);

if(tag==true)

{

if(cs.Check()==1) {

System.out.println("试探性资源分配成功(找到一个安全序列)");

boolean f=true;

for(int i=0;i<10;++i)

{

if(p.Need[i]!=0)

{

f=false;

break;

}

}

if(f)

{

int k;

for( k=0;k<ve.size();++k)

{

if(ve.get(k).name.equals(p.name)==true)

{

for(int i=0;i<10;++i)

{

Available[i]+=ve.get(k).Allocation[i];

ve.get(k).Allocation[i]=0;

ve.get(k).Need[i]+=ve.get(k).Request[i];

}

break;

}

}

ve.get(k).Finish=true;

}

System.out.println("各进程及其系统资源分配信息为：");

ds.display(ve, Available);

}else

{

System.out.println("试探性分配资源不成功，收回试探性分配资源，各进程和系统内可获得性资源为：");

int k;

for( k=0;k<ve.size();++k)

{

if(ve.get(k).name.equals(p.name)==true)

{

for(int i=0;i<10;++i)

{

ve.get(k).Allocation[i]-=p.Request[i];

ve.get(k).Need[i]+=p.Request[i];

Available[i]+=p.Request[i];

}

break;

}

}

ds.display(ve, Available);

}

}

else

{

System.out.println("试探性分配资源不成功，收回试探性分配资源，各进程和系统内可获得性资源为：");

ds.display(ve, Available);

}

}

System.out.println("是否继续进行银行家分配算法|[yes|no]");

String s;

s=cin.next();

if(s.equals("no")==true)

{

NewMain.main(null);

break;

}else

{

continue;

}

}

}

}

**Check\_Secrity.java:**

import java.util.Queue;

import java.util.LinkedList;

import java.util.Vector;

import java.lang.\*;

import java.util.Enumeration;

public class Check\_Secrity {

private PCB pp=new PCB();

int [] Work=new int[10];

Vector<PCB> ve=new Vector<PCB>();//定义一个进程的就绪队列

public Check\_Secrity()

{

}

public Check\_Secrity(int []Available,Vector<PCB> q)

{

for(int i=0;i<10;++i)

{

Work[i]=Available[i];

}

this.ve=q;

}

int Check()

{

StringBuffer [] str\_name=new StringBuffer[4];

int cnt=0;

int time=0;

Vector<PCB> vv =new Vector<PCB>();

while(cnt!=ve.size())

{

pp=ve.get(0);

ve.remove(0);

time++;

int sum=0;

for(int i=0;i<10;++i)

{

sum+=pp.Need[i];

}

if(pp.Finish==false&&sum!=0){

boolean f=true;

for(int i=0;i<10;++i)

{

if(pp.Need[i]>Work[i])

{

f=false;

break;

}

}

if(f==true)

{

for(int i=0;i<10;++i)

{

Work[i]+=pp.Allocation[i];

}

ve.addElement(pp);

cnt++;

vv.addElement(pp);

}

else

{

ve.addElement(pp);

}

}

else

{

cnt++;

ve.addElement(pp);

vv.addElement(pp);

}

if(time>10)

{

System.out.println("不存在这样的安全序列使进程处于安全状态--会出现死锁");

break;

}

}

if(cnt==ve.size())

{

System.out.println("找到一条合法序列使得进程处于安全状态---序列为");

for(int i=0;i<vv.size();++i)

{

System.out.print(vv.get(i).name+" ");

}

System.out.println("");

return 1;

}

return 0;

}

}

**Random.java**

import java.io.\*;

import java.util.\*;

import java.util.Queue;

import java.util.LinkedList;

public class Random {

Vector<PCB> ve=new Vector<PCB>();

private PCB p=new PCB();

int [] Available=new int[10];//{3,100,100,100,100,100,100,100,100,9}; int [] Work=new int [10];

Try\_to\_allocate Try=new Try\_to\_allocate();//不分配空间就没有动态内存报错

Dispaly ds=new Dispaly();

Scanner cin=new Scanner(System.in);

public Random(Vector<PCB> ve,int []Available){

this.ve=ve;

this.Available=Available;

}

void enter()

{

System.out.println("输入各进程的请求资源量：");

int num;

for(int i=0;i<ve.size();++i) {

System.out.println(ve.get(i).name+" ");

for(int j=0;j<10;++j)

{

num=cin.nextInt();

while(num>ve.get(i).Need[j])

{

System.out.println("请求输入的资源数量太大从新输入");

num=cin.nextInt();

if(num<=p.Need[i])

break;

}

ve.get(i).Request[j]=num;

}

}

System.out.println("各进程及其系统信息：");//输出请求后的信息

ds.display1(ve, Available);

int cnt=0;

while(!ve.isEmpty()) { cnt++;

p=ve.get(0);

if(cnt==5)

{

System.out.println("现死锁状态的进程:");

for(int i=0;i<ve.size();++i)

for(int j=0;j<10;++j)

{

if(ve.get(i).Need[j]!=0)

{

System.out.print(ve.get(i).name+" ");

break;

}

}

break;

}

ve.remove(0);//并删除

boolean flag=true;

if(Try.judgement(p,Available)==1)

{

System.out.println("进程对"+p.name+"资源的需求量超过了目前系统中的最大存储量--由此又不能对该进程分配资源 变为阻塞状态");

System.out.println("保存对"+p.name+"资源请求的数量:");

Try.allocate(p, Available);

/////

ve.addElement(p);

}

else//系统内的资源得可以满足该进程的申请就直接分配，不执行安全性检查

{

Try.allocate(p, Available);

for(int i=0;i<10;++i)

{

if(p.Need[i]!=0)

{

flag=false;

}

}

if(flag==true)//所有的Need得到满足 ，进程曾执行完成

{

System.out.println(p.name+"进程得到全部的资源,执行完成,并从PCB中删除");

p.Finish=true;

//回收资源

for(int i=0;i<10;++i)

{

Available[i]+=p.Request[i];

p.Allocation[i]-=p.Request[i];

p.Need[i]+=p.Request[i];

p.Request[i]=0;

}

}

else

{

System.out.println(p.name+" 该进程没有得到全部的资源，变为阻塞状态");

ve.addElement(p);

}

}

ds.display1(ve, Available);

}

}

}

**Try\_to\_allocate.java:**

import java.io.\*;

import java.util.\*;

import java.util.Queue;

import java.util.LinkedList;

public class Try\_to\_allocate{

Scanner cin=new Scanner(System.in);

void allocate(PCB p,int [] Available)

{

for(int i=0;i<10;++i)

{

if(Available[i]<p.Request[i])

{

p.Allocation[i]+=Available[i];

p.Need[i]-=Available[i];

Available[i]=0;

System.out.println("");

}

else

{

p.Allocation[i]+=p.Request[i];

p.Need[i]-=p.Request[i];

Available[i]-=p.Request[i];

}

}

}

int judgement(PCB p,int []Available)

{

for(int i=0;i<10;++i)

{

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

{

return 1;

}

}

return 0;

}

void Request\_function(PCB p )

{

int num;

for(int i=0;i<10;++i)

{

num=cin.nextInt();

while(num>p.Need[i])

{

System.out.println("从新");

num=cin.nextInt();

if(num<=p.Need[i])

break;

}

p.Request[i]=num;

}

}

}

**Display.java**

import java.io.\*;

import java.util.\*;

import java.util.Queue;

import java.util.LinkedList;

class Dispaly{

void display(Vector<PCB> ve,int []Available){

PCB temp=new PCB();

System.out.println("| name | Max | Allocation | Need | Finish | Available |");

for(int i=0;i<ve.size();++i)

{

temp=ve.get(i);

System.out.print("| "+temp.name+" |");

for(int j=0;j<10;++j)

{

System.out.print(temp.Max[j]+" ");

}

System.out.print("|");

for(int j=0;j<10;++j)

{

System.out.print(temp.Allocation[j]+" ");

}

System.out.print("|");

for(int j=0;j<10;++j)

{

System.out.print(temp.Need[j]+" ");

}

System.out.print("| ");

System.out.print(temp.Finish+" |");

if(i==0)

{

for(int j=0;j<10;++j)

{

System.out.print(Available[j]+" ");

}

}

System.out.println("");

}

}

void display1(Vector<PCB> ve,int []Available){

PCB temp=new PCB();

System.out.println("| name | Max | Allocation | Request | Need | Finish | Available |");

for(int i=0;i<ve.size();++i)

{

temp=ve.get(i);

System.out.print("| "+temp.name+" |");

for(int j=0;j<10;++j){

System.out.print(temp.Max[j]+" ");}

System.out.print("|");

for(int j=0;j<10;++j){

System.out.print(temp.Allocation[j]+" ");}

System.out.print("|");

for(int j=0;j<10;++j){

System.out.print(temp.Request[j]+" ");}

System.out.print("|");

for(int j=0;j<10;++j){

System.out.print(temp.Need[j]+" ");}

System.out.print("| ");

System.out.print(temp.Finish+" |");

if(i==0){{

System.out.print(Available[j]+" ");}}

System.out.println("");

}

}

}