**ncu2**

**实 验 报 告**

**实验课程： 操作系统**

**学生姓名： 程婧**

**学 号： 6109117016**

**专业班级： 计科可视化171班**

**2019年 11月 11日**

** 南昌大学实验报告**

学生姓名： 程婧 学 号： 6109117016 专业班级： 计科可视化171 班

实验类型：■ 验证 □ 综合 □ 设计 □ 创新 实验日期： 2019.11.11 实验成绩：

1. **实验目的**

通过实验加强对银行家安全算法的理解和掌握。

1. **实验内容**

熟悉避免死锁发生的方法，死锁与安全序列的关系，编程实现银行家算法，要求输出进程的安全序列。

1. **实验要求**

1、需写出设计说明；

2、设计实现代码及说明

3、运行结果；

**四、实验步骤**

#include <stdio.h> //程序需要的头文件

#include <stdlib.h> //程序需要的头文件

#include <string.h> //字符串处理头文件，用于Strcmp（）函数

int max[5][3];//开始定义银行家算法中需要用到的数据

int allocation[5][3];//定义已分配矩阵

int need[5][3];//定义还需矩阵

int available[3];//定义可用矩阵

int request[5][3];//需求矩阵

char \*finish[5];//检查安全序列数组

int safe[5];//保存安全序列数组

int n, i, m;//定义变量

int k = 0;

int j = 1;

int work[3];

int works[5][3];

void start(); //表示程序开始

void end(); //表示程序结束

void input(); //输入数据

void output(); //输出数据

void change(); //系统分配资源，原有资源情况改变

void outputsafe(); //输出安全序列的资源分配情况

int check(); //核心算法 判断当前资源分配情况是否安全

void main() //主程序开始

{

start();//程序开始

input();//开始输入数据

printf("这是当前输入的资源分配情况：\n");

output();

if (check() == 0) //若check函数返回0，表示输入的初始数据找不到安全序列，程序结束

{

end();

exit(0);

}

for (; j == 1;) //当有多个进程请求资源时，循环开始。此处可修改为while（j==1）{}

{

printf("输入Request[i]中 i的值为0-4："); //输入发出请求向量的进程及请求向量

scanf("%d", &i);

printf("P%d的请求向量%d：", i, i);

inputreq:

for (n = 0; n<3; n++)

scanf("%d", &request[i][n]);

if (request[i][0]>need[i][0] || request[i][1]>need[i][1] || request[i][2]>need[i][2]) //判断请求资源是否大于可用资源

goto inputreq;

//对以上goto语句进行修改，代码如下

/\*\*

do{

for(n = 0; n <3; n++)

scanf("%d", &request[i][n]);

}while(request[i][0]>need[i][0] || request[i][1]>need[i][1] || request[i][2]>need[i][2]);

\*/

if (request[i][0] <= available[0] && request[i][1] <= available[1] && request[i][2] <= available[2])//判断系统是否有足够资源提供分配

{

change(); //分配资源

j = 0;

}

else

printf("P%d等待\n", i);

if (j == 0) //j=0表示已分配资源

{

printf(" 当前分配情况：\n"); //输出分配资源后的系统资源分配情况

output();

if (check() == 0) //若找不到安全系列，则之前的资源分配无效

{

printf(" 本次分配不安全，恢复原来的状态\n");

for (m = 0; m<3; m++) //恢复分配资源前的系统资源状态

{

available[m] += request[i][m];

allocation[i][m] -= request[i][m];

need[i][m] += request[i][m];

}

output(); //输出系统资源状态

}

}

printf("是否还有Request？\n是输入1\n否输入0：\n");

scanf("%d", &j); //若还有进程请求资源，j=1，之前的for循环条件满足

}

end();

}

void start() //表示银行家算法开始

{

printf(" 银行家算法开始 ! \n");

}

void end() //表示银行家算法结束

{

printf(" 算法结束\n");

}

void input() //输入银行家算法起始各项数据

{

for (n = 0; n<5; n++)

{

printf("P%d\n", n);

printf("Max:");//输入最大资源需求

for (m = 0; m<3; m++)

scanf("%d", &max[n][m]);

printf("Allocation:");//输入已分配资源

for (m = 0; m<3; m++)

scanf("%d", &allocation[n][m]);

for (m = 0; m<3; m++)//计算还需分配的资源

need[n][m] = max[n][m] - allocation[n][m];

}

printf("Available:");//输入可用的资源数量

for (m = 0; m<3; m++)

scanf("%d", &available[m]);

}

void output() //输出系统现有资源情况

{

printf("资源情况 Max Allocation Need Available \n");

printf("进程 A B C A B C A B C A B C \n");

for (n = 0; n<5; n++)

{

printf("P%d%9d%3d%3d%5d%3d%3d%6d%3d%3d", n, max[n][0], max[n][1], max[n][2], allocation[n][0], allocation[n][1], allocation[n][2], need[n][0], need[n][1], need[n][2]);

if (n == 0) printf("%6d%3d%3d\n", available[0], available[1], available[2]);

else printf("\n");

}

}

void change() //当Request[i,j]<=Available[j]时，系统把资源分配给进程P[i],Available[j]和Need[i,j]发生改变

{

for (m = 0; m<3; m++)

{

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

allocation[i][m] += request[i][m];

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

}

}

void outputsafe() //输出安全序列的资源分配表

{

printf("该序列的资源分配如下：\n");

printf("资源情况 Work Need Allocation Work+Allocation Finish\n");

printf("进程 A B C A B C A B C A B C\n");

for (n = 0; n<5; n++)

printf("P%d%9d%3d%3d%5d%3d%3d%5d%3d%3d%6d%3d%3d%12s\n", safe[n], works[safe[n]][0], works[safe[n]][1], works[safe[n]][2], need[safe[n]][0], need[safe[n]][1], need[safe[n]][2], allocation[safe[n]][0], allocation[safe[n]][1], allocation[safe[n]][2], works[safe[n]][0] + allocation[safe[n]][0], works[safe[n]][1] + allocation[safe[n]][1], works[safe[n]][2] + allocation[safe[n]][2], finish[n]);

}

int check()//安全性算法

{

for (m = 0; m<3; m++) //数组work和finish初始化

work[m] = available[m];

for (n = 0; n<5; n++)

{

finish[n] = "false";//finsh数组初始化为false

safe[n] = 0;

}

k = 0;

for (m = 0; m<5; m++)//采用冒泡的思想一步一步逐渐排序可执行的队列

for (n = 0; n<5; n++)

if (strcmp(finish[n], "false") == 0 && need[n][0] <= work[0] && need[n][1] <= work[1] && need[n][2] <= work[2]) //查找可以分配资源但尚未分配到资源的进程

{

safe[k] = n;

works[safe[k]][0] = work[0];

works[safe[k]][1] = work[1];

works[safe[k]][2] = work[2];

work[0] += allocation[n][0];//进程执行后释放出分配给它的资源

work[1] += allocation[n][1];

work[2] += allocation[n][2];

finish[n] = "ture"; //finish[n]变为ture以示该进程完成本次分配

k++;

}

for (m = 0; m<5; m++) //判断是否所有进程分配资源完成

{

if (strcmp(finish[m], "false") == 0)

{

printf("找不到安全序列，系统处于不安全状态。\n");

return 0;//找不到安全序列，结束check函数，返回

}

else

if (m == 4) //此处m=4表示所有数组finish的所有元素都为ture

{

printf("找到安全系列 P%d->P%d->P%d->P%d->P%d, 系统是安全的\n", safe[0], safe[1], safe[2], safe[3], safe[4]);

j = 1;

outputsafe();//输出安全序列的资源分配表

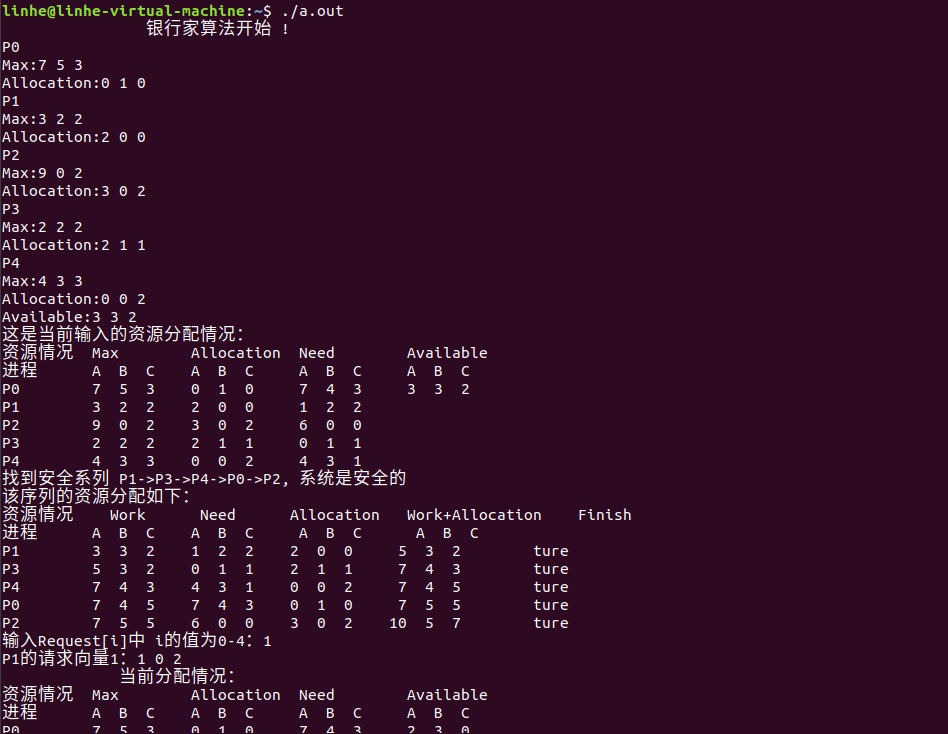
}

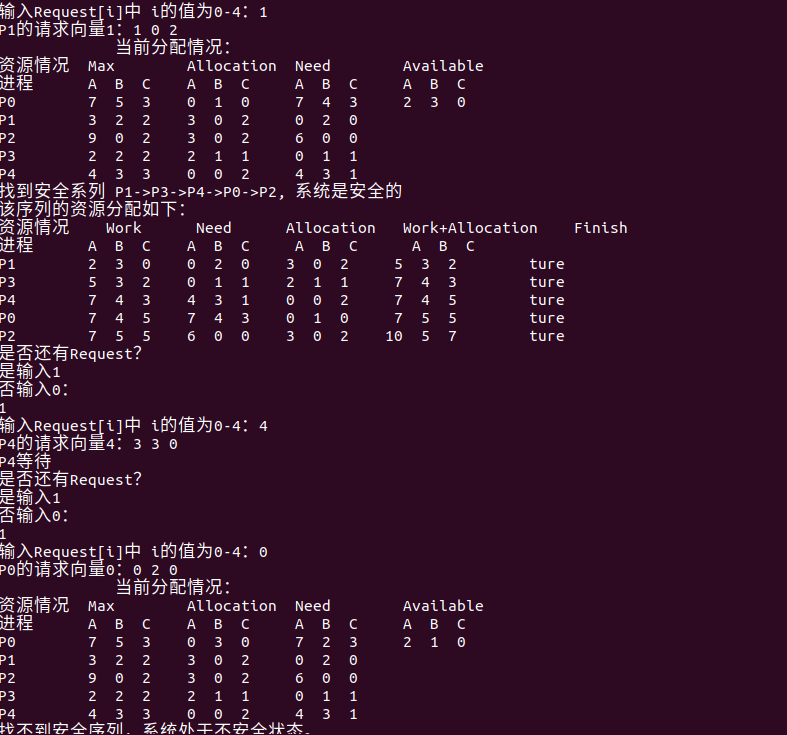
}

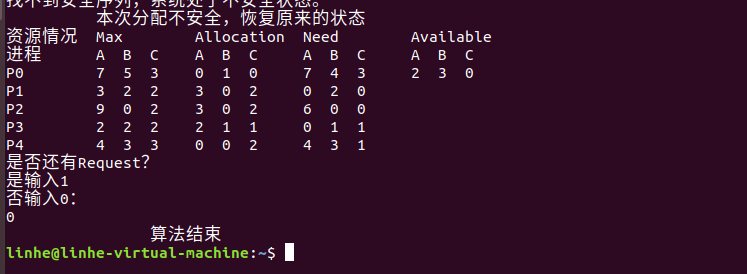
return 1;

}

**五、实验数据及处理结果**







1. **实验体会或对改进实验的建议**

本次实验内容较难，银行家算法手工实现较好理解但是代码比较难理解，利用书中讲解银行家算法的数据输入程序检测结果并带入代码进行理解，最后理解了判断当前系统是否安全的算法。经典进程同步问题在代码实现上对我有一定的难度，以后我在理解操作系统算法的同时还会加强编程实现的能力。