

《操作系统实训》指导书

一、实训目的

通过操作系统实训，主要强化学生对本课程基础知识的掌握；使学生理论联系实际，加强学生的动手能力；加深对操作系统的基本概念、工作原理和实现方法等理论知识的理解；掌握操作系统各个部分之间的有机联系，从而了解操作系统在整个计算机系统中的地位 and 作用，巩固和加强与本课程相关的其他计算机课程的知识，提高对计算机专业知识理解的系统性和完整性，并加强合作完成系统的团队精神和提高程序设计的能力。

二、实训内容

本实训的内容为实现一个模拟操作系统。要求使用实验室所提供的安装有 C 语言编程环境的计算机，模拟采用多道程序设计方法的单用户操作系统，该操作系统包括进程管理、存储管理、设备管理和文件管理四部分。

三、实训设备

- 1、PC 计算机
- 2、VC++等软件系统。

四、实训任务及要求

根据实训内容，认真完成模拟操作系统的实现，模拟操作系统需包括进程管理、存储管理、设备管理和文件管理四部分。实训的基本原理主要包括操作系统中的进程的同步与互斥；常用的进程调度算法；地址重定位；动态页式存储管理技术的页面淘汰算法；设备管理中设备的分配和回收；用死锁避免方法来处理申请独占设备可能造成的死锁；磁盘调度算法等。

本实训结束后，需要学生提交实训的源代码及可执行程序，并提交实训报告。

五、实训基本操作方法

1. 搜集与整理，设定操作系统所面临的操作需求；
2. 设计各部分的实现方案；
3. 程序开发；
4. 程序测试；
5. 系统集成；
6. 提交源程序，完成实训报告。

六、实训项目

任务一 分析操作系统所面临的操作需求

【实训目的】

使学生理解操作系统所面临的操作需求，掌握操作系统中的进程管理、存储管理、设备管理和文件管理等功能。

【实训内容】

1. 分析操作系统所面临的操作需求；
2. 熟悉实训环境；
3. 资料搜集与整理，进行实训的前期准备。

【预习要求】

操作系统的功能及实现的基本原理。

【实训步骤】

1. 分析操作系统所面临的操作需求：进程管理、存储管理、设备管理和文件管理，进一步熟悉各模块的工作原理；
2. 根据操作需求，进行系统的整体设计，画出系统总体的功能模块图，如下图 1 所示；

3. 根据上一步得出的功能模块图，进行资料的搜集与整理，并熟悉实训环境，为之后实训任务的完成打下坚实的基础。

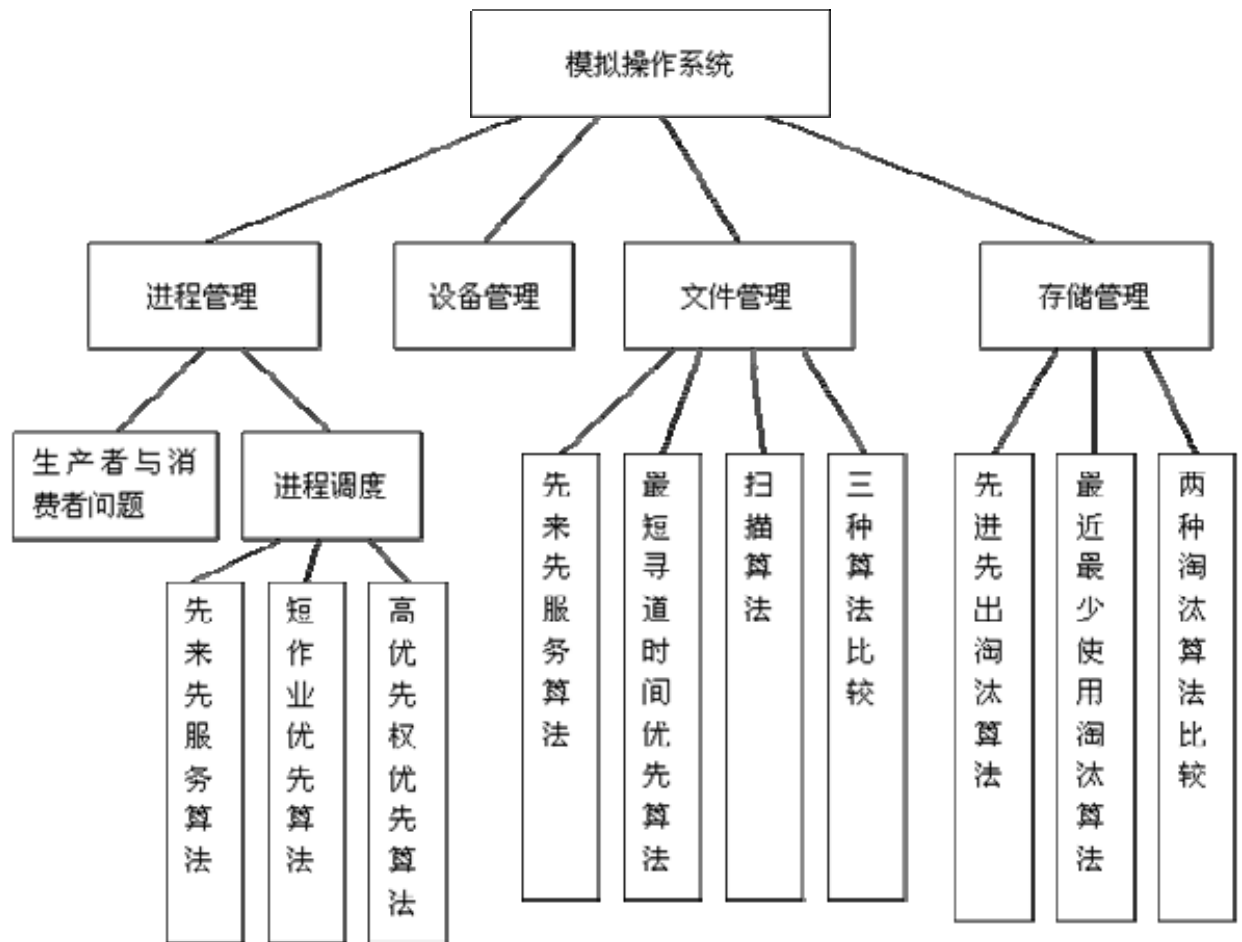


图 1 系统功能模块图

【注意事项】

操作系统中各模块之间的功能划分。

【思考题】

1. 操作系统中各模块有怎样的功能？
2. 它们之间有怎样的联系？
3. 针对某一特定的应用环境，如何完善操作系统的功能？

任务二 进程管理

【实训目的】

掌握临界区的概念及临界区的设计原则；掌握信号量的概念、PV 操作的含义以及应用 PV 操作实现进程的同步与互斥；分析进程争用资源的现象，学习解决进程互斥的方法；掌握进程的状态及状态转换；掌握常用的进程调度算法。

【实训内容】

1. 分析进程的同步与互斥现象，编程实现经典的进程同步问题——生产者消费者问题的模拟；

2. 编写允许进程并行执行的进程调度程序，在常用的进程（作业）调度算法：先来先服务算法、短作业优先算法、最高响应比优先算法、高优先权优先算法等调度算法中至少选择三种调度算法进行模拟，并输出平均周转时间和平均带权周转时间。

【预习要求】

进程同步与互斥的概念及实现方法；进程调度的作用及常用的调度算法。

【实训步骤】

1. 分析计算机系统中对资源的分配与释放过程：计算机系统每个进程都可以消费或生产某类资源。当系统中某一进程使用某一资源时，可以看作是消耗，且该进程称为消费者。而当某个进程释放资源时，则它就相当一个生产者。

2. 定义生产者消费者问题中的各数据结构，并初始化信号量；

3. 创建生产者与消费者进程，利用信号量实现生产者与消费者之间的同步与互斥；可参考的部分源代码如下：

```
include "windows.h"
#include "conio.h"
#include "stdio.h"

#define MAX 20    //定义缓冲池的最大容量是 20
int count=5;     //初始产品的数量为 5
void Proclucer()//生产者函数
{
    while(1)
```

```

{
    if(count >= MAX)
    {
        printf("缓冲池已满!等待 1 秒!\n");
        Sleep(3000);
    }
    else
    {
        count++;
        printf("生产了一个产品!当前产品的总数量是: %d\n\n", count);
        Sleep(1300);          //注意毫秒为单位
    }
}

void Consumer()    //消费者函数
{
    while(1)
    {
        if(count == 0)
        {
            printf("缓冲池已空!等待 2 秒!\n");
            Sleep(2000);
        }
        else
        {
            count--;
            printf("取出了一个产品!当前产品的数量是: %d \n\n", count);
            Sleep(2000);
        }
    }
}

```

```

}

HANDLE ahThread;

HANDLE bhThread;

HANDLE hThread;

int  tStop()    //停止函数
{
    int l=getchar();
    return l;
}

void Start()    //开始函数
{
    ahThread=CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Proclucer, NULL, 0, NULL);
    bhThread=CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Consumer, NULL, 0, NULL);
    hThread=CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)tStop, NULL, 0, NULL);    //多线程

    int  IsStop=tStop();

    if(IsStop==0)    //满足停止条件
    {
        CloseHandle(ahThread);
        CloseHandle(bhThread);
        CloseHandle(hThread);
    }
}

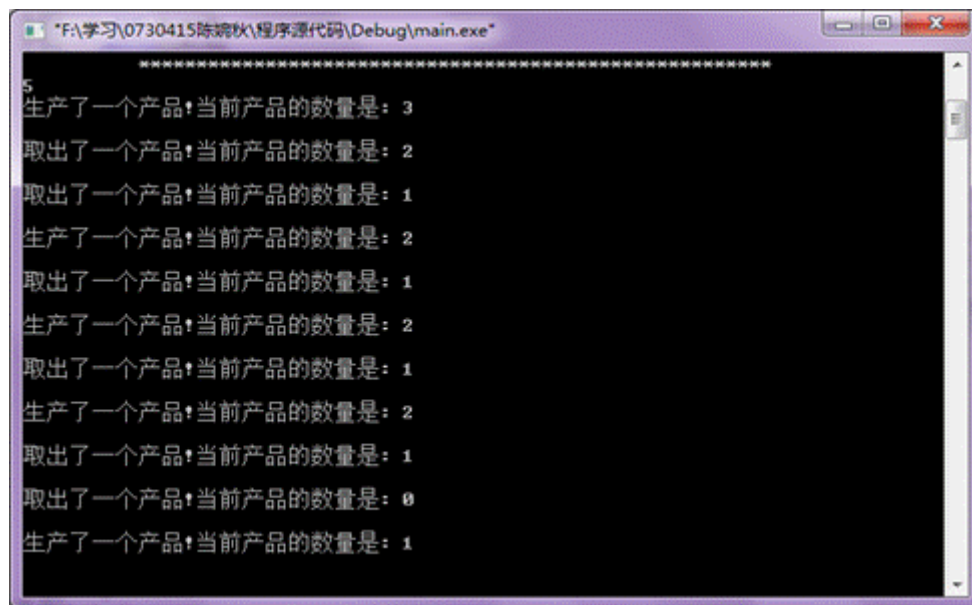
void shengcanzexiaoheiz()    //主函数
{
    printf("*****会出现返回不了主界面的情况*****");

    Start();    //开始

    printf("\n");
}

```

4. 运行并测试程序，运行界面如下图 2 所示：



```
*****
5
生产了一个产品!当前产品的数量是: 3
取出了一个产品!当前产品的数量是: 2
取出了一个产品!当前产品的数量是: 1
生产了一个产品!当前产品的数量是: 2
取出了一个产品!当前产品的数量是: 1
生产了一个产品!当前产品的数量是: 2
取出了一个产品!当前产品的数量是: 1
生产了一个产品!当前产品的数量是: 2
取出了一个产品!当前产品的数量是: 1
取出了一个产品!当前产品的数量是: 0
生产了一个产品!当前产品的数量是: 1
```

图 2 生产者与消费者问题程序模拟效果图

5. 分析常用的进程调度算法的工作原理，优先级法可被用作业或进程的调度策略。以下步骤以优先级法为例说明实训步骤。图 3 高优先权算法的工作流程：

首先，系统或用户按某种原则为作业或进程指定一个优先级来表示该作业或进程所享有的调度优先级权。该算法的核心是确定进程或作业的优先级。确定优先级的方法分为两类：静态法和动态法。静态法根据作业或进程的静态特性，在作业或进程开始执行之前就确定它们的优先级，一旦开始执行之后就不能改变。动态法则不然，它把作业或进程的静态特性和动态特性结合起来确定作业或进程的优先级，随着作业或进程的执行过程，其优先级不断变化。本实训指导书以动态法为例进行说明。静态优先级的调度算法实现虽然简单，系统开销小，但由于静态优先级一旦确定之后，直到执行结束为止始终保持不变，从而系统效率较低，调度性能不高，而动态优先级的系统效率较高，调度性能也高。

6. 高优先权算法的设计，下图 4 所示：

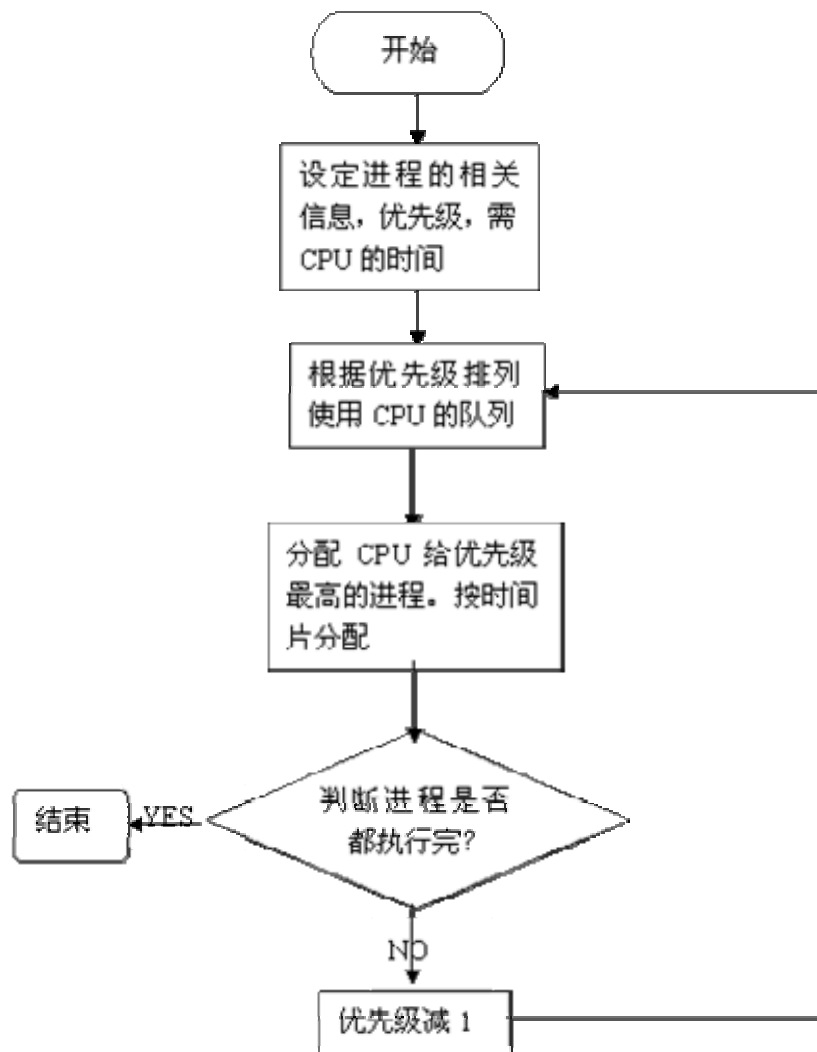


图 3 高优先权算法的流程

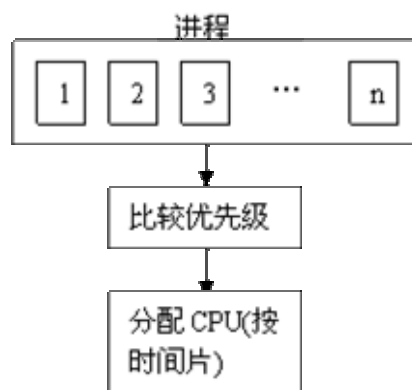


图 4 高优先权算法的设计

先读入进程，比较进程的优先级，排列出分配 CPU 的队列，按时间片分配 CPU，一个时间片后，优先级减 1，再一次比较优先级，再排分配 CPU 的队列，按时间片分配 CPU，直到进程全部执行完毕。

每个进程有一个进程控制块（PCB）表示。进程控制块可以包含如下信息：进程名、优先数、到达时间、需要运行时间、已用 CPU 时间、进程状态等等。进程的优先数及需要的运行时间可以事先人为地指定（也可以由随机数产生）。进程的到达时间为输入进程的时间。进程的运行时间以时间片为单位进行计算。

每个进程的状态可以是就绪 W（Wait）、运行 R（Run）、或完成 F（Finish）三种状态之一。就绪进程获得 CPU 后都只能运行一个时间片。用已占用 CPU 时间加 1 来表示。如果运行一个时间片后，进程的已占用 CPU 时间已达到所需要的运行时间，则撤消该进程，如果运行一个时间片后进程的已占用 CPU 时间还未达所需要的运行时间，也就是进程还需要继续运行，此时应将进程的优先数减 1（即降低一级），然后把它插入就绪队列等待 CPU。

每进行一次调度程序都打印一次运行进程、就绪队列、以及各个进程的 PCB，以便进行检查。

7. 根据以上算法编写程序，最后计算平均周转时间与平均带权周转时间。

【注意事项】

动态优先权与静态优先权的区别。

【思考题】

针对某一具体应用环境，如何选择合适的调度算法？

任务三 存储管理

【实训目的】

掌握物理内存和虚拟内存的基本概念；掌握重定位的基本概念及其要点，理解逻辑地址与绝对地址；掌握各种存储管理的实现方法，包括基本原理、地址变换和缺页中断、主存空间的分配及分配算法；掌握常用淘汰算法。

【实训内容】

编写一个模拟的动态页式存储管理程序，实现对动态页式存储的淘汰算法的模拟（在先进先出淘汰算法、最近最少使用淘汰算法、最不经常使用淘汰算法三种算法中至少选择两种进行模拟）并计算各个算法的缺页率；并且页面淘汰算法在淘汰一页时，只将该页在页表中抹去，而不再判断它是否被改写过，也不将它写回到辅存。

【预习要求】

常用的存储管理方法及其基本原理；物理内存与虚拟内存、逻辑地址与绝对地址的概念；常用的淘汰算法。

【实训步骤】

以先进先出淘汰算法为例说明动态页式存储管理的实现过程：

1. 产生一个需要访问的指令地址流，它是一系列需要访问的指令的地址。为不失一般性，你可以适当地（用人工指定地方法或用随机数产生器）生成这个序列，使得 50% 的指令是顺序执行的。25% 的指令均匀地散布在前地址部分，25% 的地址是均匀地散布在后地址部分；
2. 指定合适的页面尺寸（例如以 1K 或 2K 为 1 页）；
3. 指定内存页表的最大长度，并对页表进行初始化；
4. 每访问一个地址时，首先要计算该地址所在的页的页号，然后查页表，判断该页是否在主存——如果该页已在主存，则打印页表情况；如果该页不在主存且页表未满，则调入一页并打印页表情况；如果该页不在主存且页表已满，则按 FIFO 页面淘汰算法淘汰一页后调入所需的页，打印页表情况； 逐个地址访问，直到所有地址访问完毕。
5. 存储管理算法的流程图如图 5 所示：
6. 根据图 5 编写并运行程序，程序运行界面如图 6 所示。

【注意事项】

页面尺寸的指定；如何选择合适的页面淘汰算法以保证较低的缺页率。

【思考题】

各种不同的页面淘汰算法有哪些优缺点？为什么会产生页面抖动？什么是 belady 现象？这种现象该如何避免？

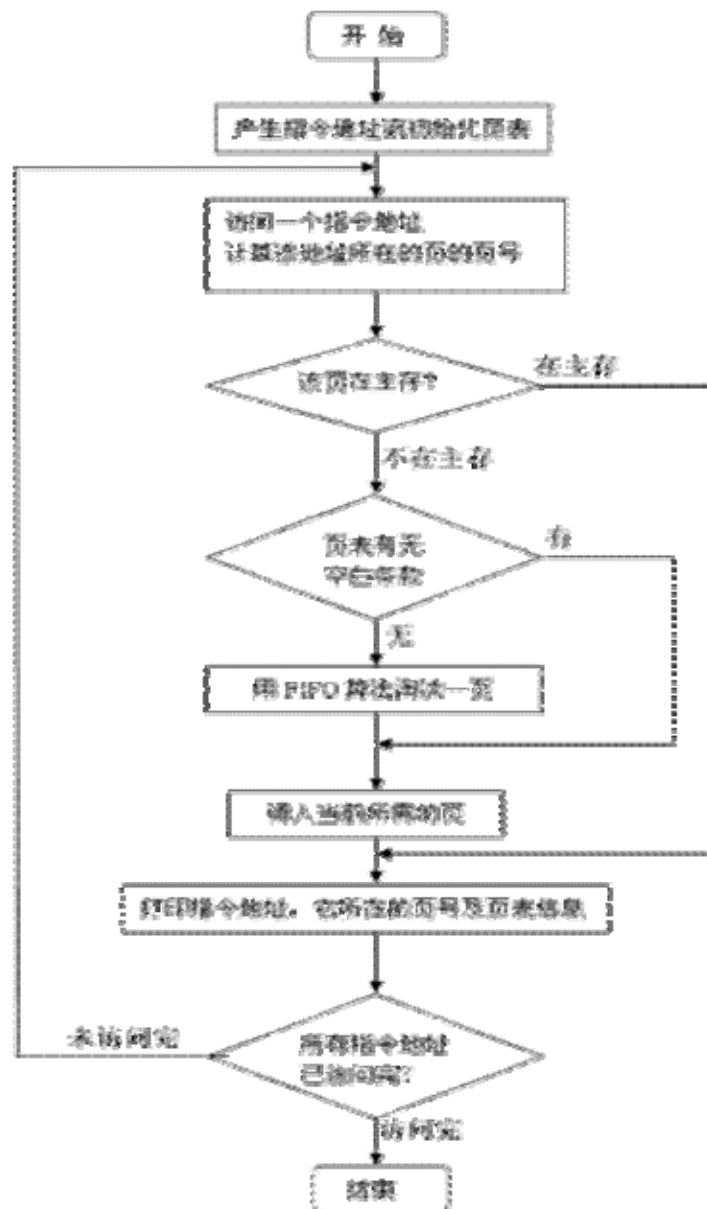


图 5 动态页式存储管理的流程图



图 6 采用先进先出淘汰算法的动态页式存储管理运行界面

任务四 设备管理

【实训目的】

掌握独占设备的使用方式，以及设备的分配和回收；掌握用死锁避免方法来处理申请独占设备可能造成的死锁。

【实训内容】

用死锁避免方法来处理申请独占设备可能造成的死锁，程序实现对银行家算法的模拟。

【预习要求】

设备的分类；独占设备的分配与回收；处理死锁的方法。

【实训步骤】

1. 设计银行家算法的数据结构：

假设有 M 个进程 N 类资源，则有如下数据结构：

MAX[M*N] M 个进程对 N 类资源的最大需求量

AVAILABLE[N] 系统可用资源数

ALLOCATION[M*N] M 个进程已经得到 N 类资源的资源量

NEED[M*N] M 个进程还需要 N 类资源的资源量

2. 分析银行家算法的实现过程，流程图如下图 7 所示：

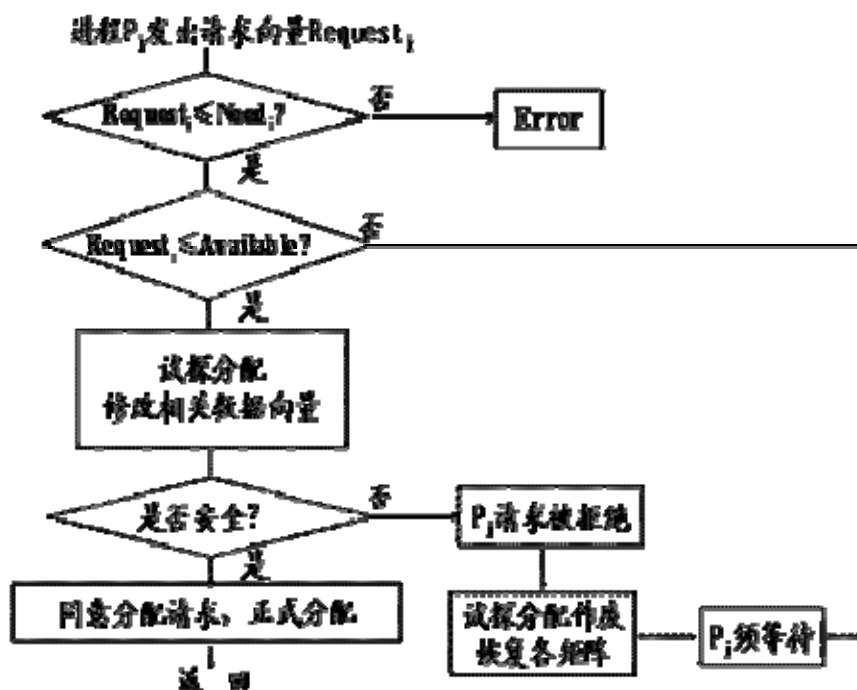


图 7 银行算法的实现流程

3. 根据流程图编写程序，部分源代码如下所示：

```
#include "string.h"
#include <stdio.h>
#include <stdlib.h>
#define M 5
#define N 3
#define FALSE 0
#define TRUE 1
/*M 个进程对 N 类资源最大资源需求量*/
int MAX[M][N]={{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};
```

```

/*系统可用资源数*/
int AVAILABLE[N]={10, 5, 7};
/*M 个进程对 N 类资源最大资源需求量*/
int ALLOCATION[M][N]={ {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0} };
/*M 个进程已经得到 N 类资源的资源量 */
int NEED[M][N]={ {7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3} };
/*M 个进程还需要 N 类资源的资源量*/
int Request[N]={0, 0, 0}; //初始化需要申请的资源数目
void YinHangJia() //银行家算法的实现
{
    int i=0, j=0;
    char flag='Y';
    void showdata();
    void changdata(int);
    void rstordata(int);
    int chkerr(int);
    showdata();
    while(flag=='Y' || flag=='y')
    {
        i=-1;
        while(i<0 || i>=M)
        {
            printf("请输入需申请资源的进程号 (从 0 到");
            printf("%d", M-1);
            printf(", 否则重输入!) :");
            scanf("%d", &i);
            if(i<0 || i>=M) printf("输入的进程号不存在, 重新输入!\n");
        }
        printf("请输入进程");
        printf("%d", i);
        printf("申请的资源数\n");
        for (j=0; j<N; j++)
        {
            printf("资源");

```

```

printf("%d", j);
printf(":");
scanf("%d",&Request[j]);
if(Request[j]>NEED[i][j])//第一步判断申请的资源数是否大于需要的资源数
{
printf("进程");
printf("%d", i);
printf("申请的资源数大于进程");
printf("%d", i);
printf("还需要");
printf("%d", j);
printf("类资源的资源量!申请不合理, 出错!请重新选择!\n");
flag='N';
break;
}
else
{
if(Request[j]>AVAILABLE[j])//第二步判断申请的资源数是否大于可用资源数
{
printf("进程");
printf("%d", i);
printf("申请的资源数大于系统可用");
printf("%d", j);
printf("类资源的资源量!申请不合理, 出错!请重新选择!\n");
flag='N';
break;
}
}
}
if(flag=='Y' || flag=='y')
{
changdata(i);
if(chkerr(i))
{

```

```

rstordata(i);
showdata();
}
else
showdata();
}
else
showdata();
printf("\n");
scanf("%c",&flag);
}
}
void showdata()
{
int i,j;
printf("系统可用的资源数为:\n");
printf(" ");
for (j=0;j<N;j++){
printf("    资源");
printf("%d",j);
printf(":");
printf("%d",AVAILABLE[j]);
}
printf("\n");
printf("各进程还需要的资源量:\n");
for (i=0;i<M;i++)
{
printf("    进程");
printf("%d",i);
printf(":");
for (j=0;j<N;j++){
printf("资源");
printf("%d",j);
printf(":");

```



```

printf("%d", NEED[i][j]);
}
printf("\n");
}
printf("各进程已经得到的资源量: \n");
for (i=0; i<M; i++)
{
printf("    进程");
printf("%d", i);
/*printf(":\n");*/
for (j=0; j<N; j++) {
printf("资源");
printf("%d", j);
printf(":");
printf("%d", ALLOCATION[i][j]);
}
printf("\n");
}
}

void changdata(int k)
{
int j;
for (j=0; j<N; j++)//修改数据结构的值
{
AVAILABLE[j]=AVAILABLE[j]-Request[j];
ALLOCATION[k][j]=ALLOCATION[k][j]+Request[j];
NEED[k][j]=NEED[k][j]-Request[j];
}
};

void rstordata(int k)
{
int j;
for (j=0; j<N; j++)//修改数据结构的值
{

```

```

AVAILABLE[j]=AVAILABLE[j]+Request[j];
ALLOCATION[k][j]=ALLOCATION[k][j]-Request[j];
NEED[k][j]=NEED[k][j]+Request[j];
}
};

int chkerr(int s)
{
int WORK, FINISH[M], temp[M];
int i, j, k=0;
for(i=0; i<M; i++) FINISH[i]=FALSE;
for(j=0; j<N; j++)//用安全性检查算法判断系统是否安全（即是否为 true）
{
WORK=AVAILABLE[j];
i=s;
while(i<M)
{
if (FINISH[i]==FALSE&&NEED[i][j]<=WORK)
{
WORK=WORK+ALLOCATION[i][j];
FINISH[i]=TRUE;
temp[k]=i;
k++;
i=0;
}
else
{
i++;
}
}
for(i=0; i<M; i++)
if(FINISH[i]==FALSE)
{
printf("\n");
printf("系统不安全!!! 本次资源申请不成功!!!\n");

```

```

printf("\n");
return 1;
}
}

printf("\n");
printf("经安全性检查，系统安全，本次分配成功.\n");
printf("\n");
printf(" 本次安全序列：");
for(i=0;i<M;i++){
printf("进程");
printf("%d",temp[i]);
printf("->");
}
printf("\n");
return 0;
}

```

4. 测试并运行程序，运行界面如图 8 所示：

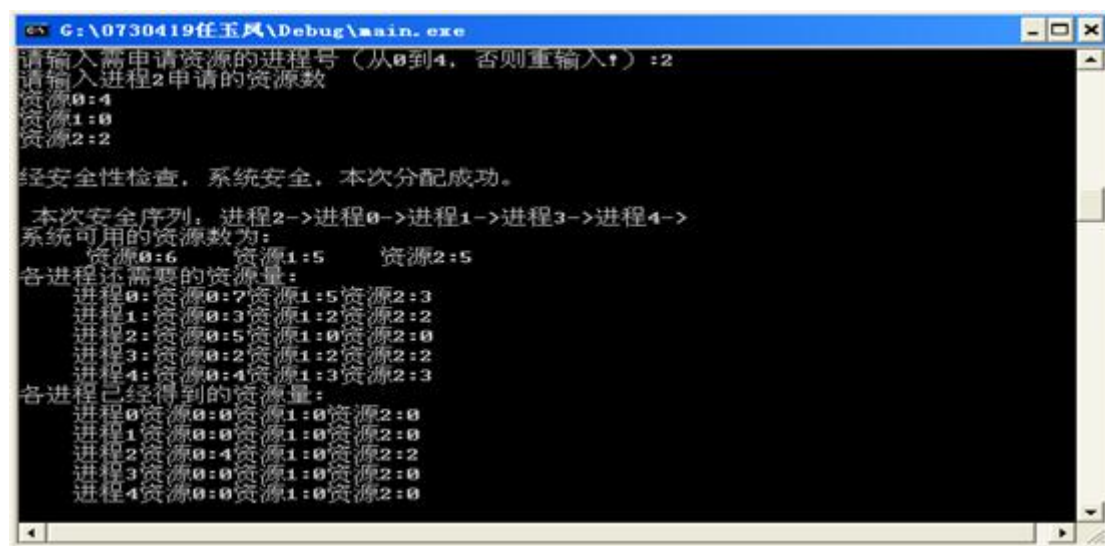


图 8 银行家算法运行界面

【注意事项】

独占设备的分配方式。

【思考题】

如果产生了死锁，应如何解除？

任务五 文件管理

【实训目的】

掌握文件的存取方法；掌握文件的逻辑结构和物理结构；掌握存储空间的分配和回收；掌握磁盘管理与调度。

【实训内容】

用程序模拟磁盘的调度过程，并计算各磁盘调度算法包括先来先服务算法、最短寻道时间优先算法、扫描算法和循环扫描算法的平均寻道长度。

【预习要求】

文件的逻辑结构和物理结构；常用的磁盘调度算法。

【实训步骤】

1. 分析常用的磁盘调度算法，熟悉其基本原理。
2. 自行设定起始扫描磁道号及最大磁道号数，程序根据设定值随机产生 n 个磁道号进行模拟（n 也可自行设定）；
3. 编写程序实现磁盘调度算法，并显示该算法寻道顺序并计算出寻道总数和平均寻道数；对各种算法的优劣进行比较，得出比较结果。部分源代码如下所示：

```
#include "stdio.h"
```

```
#include "stdlib.h"
```

```

void CopyL(int Sour[],int Dist[] ,int x); //数组 Sour 复制到数组 Dist,
复制到 x 个数

void SetDI(int DiscL[]); //随机生成磁道数

void Print(int Pri[],int x); //打印输出数组 Pri

void DelInq(int Sour[],int x,int y);

//数组 Sour 把 x 位置的数删除,并把 y 前面的数向前移动,y 后的数保持不变(即
会出现 2 个 y)

void FCFS(int Han,int DiscL[]); //先来先服务算法(FCFS)

void SSTF(int Han,int DiscL[]); //最短寻道时间优先算法(SSTF)

int SCAN(int Han,int DiscL[],int x,int y); //扫描算法(SCAN)

void CSCAN(int Han,int DiscL[]); //循环扫描算法(CSCAN)

void N_Step_SCAN(int Han1,int DiscL[]); //N 步扫描算法

void PaiXu(); //寻道长度由低到高排序

void Pri();

int Nall=0;

int Best[5][2]; //用作寻道长度由低到高排序时存放的数组

int Limit=0; //输入寻找的范围磁道数 i

int Jage;

float Aver=0;

int fa5()

{

```

```

int i;

int DiscLine[10]; //声明准备要生成的随机磁道号的数组

int Hand; //磁道数

int Con=1;

int n;

while(Con==1)

{

    Jage=0;

    printf("\n 请输入初始的磁道数(0<n<65536):");

    scanf("%d",&Hand);

    printf("\n+ 输入寻找的范围:");

    scanf("%d",&Limit);

    if(Limit>65536){

        printf("超出范围!");

    }

    else{

        printf("      _____      \n");

        printf("      ||      操作系统课程设计      ||      \n");

        printf("      _____||      磁盘调度算法      ||_____ \n");


        printf("      ||      ||      ||      ||      \n");

        printf("      \n");

    }

}

```

printf(“		\
n”);		
printf(“	1. 先来先服务算法 (FCFS)	\n
”);		
printf(“		\
n”);		
printf(“	2. 最短寻道时间优先算法 (SSTF)	\n
”);		
printf(“		\
n”);		
printf(“	3. 扫描算法 (SCAN)	\n
”);		
printf(“		\
n”);		
printf(“	4. 循环扫描算法 (CSCAN)	\n
”);		
printf(“		\
n”);		
printf(“	5. N 步扫描算法 (NStepScan)	\n
”);		
printf(“		\
n”);		
printf(“	6. 各类算法的比较	\
n”);		
printf(“		\
n”);		

```

printf( " || || \n");

printf( " || _____ || \n");

printf( " └─┬───────────┬─┘ \n");
printf( " └───────────┘ \n");

scanf("%d",&n);

if(n==0) exit(0);

printf("\n");

switch(n)
{

case 1:

SetDI(DiscLine); //随机生成磁道数

FCFS(Hand, DiscLine); //先来先服务算法 (FCFS)

break;

case 2:

SetDI(DiscLine); //随机生成磁道数

SSTF(Hand, DiscLine); //最短寻道时间优先算法 (SSTF)

break;

case 3:

SetDI(DiscLine); //随机生成磁道数

SCAN(Hand, DiscLine, 0, 9); //扫描算法 (SCAN)

```



```

break;

case 4:

    SetDI(DiscLine); //随机生成磁道数

    CSCAN(Hand, DiscLine); //循环扫描算法 (CSCAN)

    break;

case 5:

    SetDI(DiscLine); //随机生成磁道数

    N_Step_SCAN(Hand, DiscLine); //N 步扫描算法 (NStepScan)

    break;

case 6:

    SetDI(DiscLine); //随机生成磁道数

    FCFS(Hand, DiscLine); //先来先服务算法 (FCFS)

    SSTF(Hand, DiscLine); //最短寻道时间优先算法 (SSTF)

    SCAN(Hand, DiscLine, 0, 9); //扫描算法 (SCAN)

    CSCAN(Hand, DiscLine); //循环扫描算法 (CSCAN)

    N_Step_SCAN(Hand, DiscLine); //N 步扫描算法 (NStepScan)

    PaiXu(); //寻道长度由低到高排序

    printf("\n\n+ 寻道长度由低到高排序:");

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

    {

        printf("%4d ", Best[i][0]);

```

```

    }

    break;

}

printf("\n\n+ 是否继续(按 0 结束, 按 1 继续)?");

scanf("%5d",&Con);

}

}

}

```

//数组 Sour 复制到数组 Dist, 复制到 x 个数

```
void CopyL(int Sour[],int Dist[] ,int x)
```

```

{

    int i;

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

    {

        Dist[i]=Sour[i];

    }

}

```

//打印输出数组 Pri

```
void Print(int Pri[],int x)
```

```

{

    int i;

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

    {

        printf("%5d",Pri[i]);

    }

}

//随机生成磁道数

void SetDI(int DiscL[])

{

    int i;

    for(i=0;i<=9;i++)

    {

        DiscL[i]=rand()%Limit;//随机生成 10 个磁道号

    }

    printf("+ 需要寻找的磁道号:");

    Print(DiscL, 9); //输出随机生成的磁道号

    printf("\n");

}

//数组 Sour 把 x 位置的数删除，并把 y 前面的数向前移动，y 后的数保持不变
(即会出现 2 个 y)

void DelInq(int Sour[],int x,int y)

{

```

```

int i;

for(i=x;i<y;i++)

{

    Sour[i]=Sour[i+1];

    x++;

}

}

//先来先服务算法(FCFS)

void FCFS(int Han,int DiscL[])

{

    int RLine[10]; //将随机生成的磁道数数组 DiscL[]复制给数组 RLine[]

    int i,k,All,Temp; //Temp 是计算移动的磁道距离的临时变量

    All=0; //统计全部的磁道数变量

    k=9; //限定 10 个的磁道数

    CopyL(DiscL,RLine,9); //复制磁道号到临时数组 RLine

    printf("\n+ 按照 FCFS 算法磁道的访问顺序为:");

    All=Han-RLine[0];

    for(i=0;i<=9;i++)

    {

        Temp=RLine[0]-RLine[1];//求出移动磁道数,前一个磁道数减去后一个磁道数得出临时的移动距离

        if(Temp<0)

```

```

    Temp=(-Temp); //移动磁道数为负数时, 算出相反数作为移动磁道数

    printf("%5d", RLine[0]);

    All=Temp+All; //求全部磁道数的总和

    DelInq(RLine, 0, k); //每个磁道数向前移动一位

    k--;

}

Best[Jage][1]=All; //Best[][1] 存放移动磁道数

Best[Jage][0]=1; //Best[][0] 存放算法的序号为:1

Jage++; //排序的序号加 1

Aver=((float) All)/10; //求平均寻道次数

printf("\n+ 移动磁道数:<%5d> ", All);

printf("\n+ 平均寻道长度:%0.2f* ", Aver);

}

//最短寻道时间优先算法 (SSTF)

void SSTF(int Han, int DiscL[])

{

    int i, j, k, h, All;

    int Temp; //Temp 是计算移动的磁道距离的临时变量

    int RLine[10]; //将随机生成的磁道数数组 Disc1[] 复制给数组 RLine[]

    int Min;

    All=0; //统计全部的磁道数变量

```

```

k=9; //限定 10 个的磁道数

CopyL(DiscL, RLine, 9); //复制磁道号到临时数组 RLine

printf("\n+ 按照 SSTF 算法磁道的访问顺序为:");

for(i=0; i<=9; i++)

{

    Min=64000;

    for(j=0; j<=k; j++) //内循环寻找与当前磁道号最短寻道的时间的磁道号

    {

        if(RLine[j]>Han) //如果第一个随机生成的磁道号大于当前的磁道号，执行下一句

            Temp=RLine[j]-Han; //求出临时的移动距离

        else

            Temp=Han-RLine[j]; //求出临时的移动距离

        if(Temp<Min) //如果每求出一次的移动距离小于 Min，执行下一句

        {

            Min=Temp; //Temp 临时值赋予 Min

            h=j; //把最近当前磁道号的数组下标赋予 h

        }

    }

    All=All+Min; //统计一共移动的距离

    printf("%5d", RLine[h]);

    Han=RLine[h];

```

```

    DelInq(RLine, h, k); //每个磁道数向前移动一位

    k--;

}

Best[Jage][1]=All;//Best[][1]存放移动磁道数

Best[Jage][0]=2;//Best[][0]存放算法的序号为:2

Jage++; //排序序号加 1

Aver=((float)All)/10;//求平均寻道次数

printf("\n+ 移动磁道数:<%5d> ", All);

printf("\n+ 平均寻道长度:%0.2f* ", Aver);

}

//扫描算法(SCAN)

int SCAN(int Han, int DiscL[], int x, int y)

{

    int j, n, k, h, m, All;

    int t=0;

    int Temp;

    int Min;

    int RLine[10]; //将随机生成的磁道数数组 Disc1[]复制给数组 RLine[]

    int Order;

    Order=1;

    k=y;

```

```

m=2; //控制 while 语句的执行，即是一定要使当前磁道向内向外都要扫描到

All=0; //统计全部的磁道数变量

CopyL(DiscL, RLine, 9); //复制磁道号到临时数组 RLine

printf("\n+ 按照 SCAN 算法磁道的访问顺序为:");

Min=64000;

for(j=x; j<=y; j++) //寻找与当前磁道号最短寻道的时间的磁道号
{
    if(RLine[j]>Han) //如果第一个随机生成的磁道号大于当前的磁道号，执行
    下一句

        Temp=RLine[j]-Han; //求出临时的移动距离

    else

        Temp=Han-RLine[j]; //求出临时的移动距离

    if(Temp<Min)

    {

        Min=Temp; //Temp 临时值赋予 Min

        h=j; //把最近当前磁道号的数组下标赋予 h

    }

}

All=All+Min;

printf("%5d", RLine[h]);

if(RLine[h]>=Han) { //判断磁道的移动方向，即是由里向外还是由外向里

    Order=0;

```



```

    t=1;

}

Han=RLine[h];

DelInq(RLine,h,k); //每个磁道数向前移动一位

k--;

while(m>0)

{

    if(Order==1) //order 是判断磁盘扫描的方向标签，order 是 1 的话，磁道
    向内移动

    {

        for(j=x;j<=y;j++)

        {

            h=-1;

            Min=64000;

            for(n=x;n<=k;n++) //判断离当前磁道最近的磁道号

            {

                if(RLine[n]<=Han)

                {

                    Temp=Han-RLine[n];

                    if(Temp<Min)

                    {

                        Min=Temp; //Temp 临时值赋予 Min

```

```

        h=n;  //把最近当前磁道号的数组下标赋予 h

    }

}

}

if(h!=-1)

{

    All=All+Min;  //叠加移动距离

    printf("%5d", RLine[h]);

    Han=RLine[h]; //最近的磁道号作为当前磁道

    DelInq(RLine, h, k);

    k--;

}

}

    Order=0;  //当完成向内的移动，order 赋予 0，执行 else 语句，使磁道向外移动

    m--;  //向内完成一次，m 减一次，保证 while 循环执行两次

}

else  //order 是 0 的话，磁道向外移动

{

    for(j=x; j<=y; j++)

    {

        h=-1;

```

```

Min=64000;

for(n=x;n<=k;n++) //判断离当前磁道最近的磁道号

{

    if(RLine[n]>=Han)

    {

        Temp=RLine[n]-Han;

            if(Temp<Min)

            {

                Min=Temp;    //Temp 临时值赋予 Min

                    h=n;    //把最近当前磁道号的数组下标赋予 h

            }

        }

    }

if(h!=-1)

{

    All=All+Min;    //叠加移动距离

    printf("%5d", RLine[h]);

    Han=RLine[h];    //最近的磁道号作为当前磁道

    DelInq(RLine, h, k);

    k--;

}

```

```

    }

    Order=1; //当完成向内的移动, order 赋予 0, 执行 else 语句, 使磁道向外移动

    m--; //向内完成一次, m 减一次, 保证 while 循环执行两次

}

}

NA11=NA11+A11;

if((y-x)>5)

{

    Best[Jage][1]=A11;//Best[][1]存放移动磁道数

    Best[Jage][0]=3;//Best[][0]存放算法的序号为:3

    Jage++; //排序序号加 1

    Aver=((float)A11)/10;//求平均寻道次数

    printf("\n+ 移动磁道数:<%5d> ",A11);

    printf("\n+ 平均寻道长度:%0.2f* ",Aver);

}

if(t==1) printf("\n+ 磁道由内向外移动");

else printf("\n+ 磁道由外向内移动");

return(Han);

}

//循环扫描算法(CSCAN)

void CSCAN(int Han,int DiscL[])

```

```

{

int j, h, n, Temp, m, k, All, Last, i;
int RLine[10]; //将随机生成的磁道数数组 Disc1[]复制给数组 RLine[]

int Min;
int tmp=0;
m=2;
k=9;
All=0; //统计全部的磁道数变量
Last=Han;
CopyL(DiscL, RLine, 9); //复制磁道号到临时数组 RLine
    printf("\n+ 按照 CSCAN 算法磁道的访问顺序为:");
while(k>=0)
{
    for(j=0; j<=9; j++) //从当前磁道号开始, 由内向外搜索离当前磁道最
近的磁道号
    {
        h=-1;
        Min=64000;
        for(n=0; n<=k; n++)
        {
            if(RLine[n]>=Han)
            {
                Temp=RLine[n]-Han;
                if(Temp<Min)
                {
                    Min=Temp;
                    h=n;
                }
            }
        }
    }
}

```

```

    }
}
if(h!=-1)
{
    All=All+Min; //统计一共移动的距离
    printf("%5d", RLine[h]);
    Han=RLine[h];
    Last=RLine[h];
    DelInq(RLine, h, k);
    k--;
}
}
if(k>=0)
{ tmp=RLine[0];
    for(i=0;i<k;i++)//算出剩下磁道号的最小值
    {
        if(tmp>RLine[i]) tmp=RLine[i];
    }
    Han=tmp;//把最小的磁道号赋给 Han
    Temp=Last-tmp;//求出最大磁道号和最小磁道号的距离差
    All=All+Temp;
}
}
Best[Jage][1]=All;//Best[][1]存放移动磁道数
Best[Jage][0]=4;//Best[][0]存放算法的序号为:4
Jage++;//排序序号加 1
Aver=((float)All)/10;//求平均寻道次数
printf("\n+ 移动磁道数:<%5d> ", All);
printf("\n+ 平均寻道长度:%0.2f* ", Aver);
}

```

```

//N 步扫描算法(NStepScan)

void N_Step_SCAN(int Han1, int DiscL[])
{
    int i, m, k;
    int RLine1[10];
    NA11=0;
    m=2;
    k=9;  //限定 10 个的磁道数
    i=-1;
    CopyL(DiscL, RLine1, 9);  //复制磁道号到临时数组 RLine
    printf("\n+ 按照 N_Step_SCAN 算法磁道的访问顺序为:");
    for(m=0;m<2;m++)  //由于限定 10 磁道数，将 10 个磁道数分为两组，每
组 5 个磁道数，每个组按照 SCAN 算法执行，该循环循环 2 次
    {
        Han1=SCAN(Han1, RLine1, i+1, i+5);
        i=i+5;
    }
    Best[Jage][1]=NA11;//Best[][1]存放移动磁道数
    Best[Jage][0]=5;//Best[][0]存放算法的序号为:5
    Aver=((float)NA11)/10;//求平均寻道次数
    printf("\n+ 移动磁道数:<%5d> ", NA11);
    printf("\n+ 平均寻道长度:%0.2f* ", Aver);
}

//寻道长度由低到高排序
void PaiXu()
{
    int i, j, Temp;
    for(i=0;i<5;i++)
    {
        for(j=0;j<4;j++)

```

```

{
    if (Best[j][1]>Best[j+1][1]) //如果前一个算法的移动磁道距离大
于后一个移动磁道数，执行下面语句
    {
        Temp=Best[j+1][1];    //从这起下三行执行冒泡法将移动距离大小
排序，排完后则执行每个算法的排序
        Best[j+1][1]=Best[j][1];
        Best[j][1]=Temp;
        Temp=Best[j+1][0]; //将每个算法的序号用冒泡法排序
        Best[j+1][0]=Best[j][0];
        Best[j][0]=Temp;
    }
} }

```

4. 测试并运行程序，运行界面如图 9 所示：



图 9 磁盘调度算法运行界面

【注意事项】

避免扫描算法中的饥饿现象。

【思考题】

如何在文件管理模块增加如下功能：

1. 改变目录：格式：cd <目录名>
2. 显示目录：格式：dir [<目录名>]
3. 创建目录：格式：md <目录名>
4. 除目录：格式：rd <目录名>
5. 新建文件：格式：edit <文件名>
6. 删除文件：格式：del <文件名>
7. 退出文件系统：exit

任务六 模块的集成，书写实训报告

【实训目的】

增强学生的编程能力及团队精神。

【实训内容】

1. 将以上四个模块集成为一个较完整的模拟操作系统。
2. 进行系统中整体代码的调试和优化
3. 归纳并解决实训过程中遇到的问题，书写实训报告
4. 提交源程序和实训报告。

【预习要求】

操作系统中各模块之间的有机联系。

【实训步骤】

1. 四个模块集成为一个较完整的模拟操作系统。
2. 进行系统中整体代码的调试和优化
3. 归纳并解决实训过程中遇到的问题，书写实训报告
4. 提交源程序和实训报告。

【注意事项】

1. 源代码格式规范，注释不少于三分之一；
2. 对程序的每一部分要有详细的设计分析说明；
3. 程序执行的每个步骤要有具体的提示内容或输出；
4. 画出程序的基本结构框图和流程图；
5. 提交完整程序代码、可执行程序、实训设计报告等相关文档。

【思考题】

在实训过程中，你是如何解决遇到的问题？实训成果有哪些有待完善的地方？应如何完善？

七、实训考核标准及方式

《操作系统实训》考核主要检验学生对操作系统相关理论掌握的情况以及应用高级语言对基本原理和方法进行编程实现的能力。本课程以编程实现模拟操作系统为主，对操作系统的基本原理以及编程过程中常见的错误采用讲解演示的方法进行集体辅导，对个别问题采用单独辅导的方式进行，鼓励学生通过自己动手、讨论的方式解决问题。

本课程考核由出勤情况和上机表现、题目完成的质量、实训报告以及回答提问等方式进行综合评定。其中出勤情况和上机表现占 20%，题目完成的质量占 50%、实训报告以及回答提问占 30%。