

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 操作系统实验 | | | | |
| 实验编号： | 实验七 | | | | |
| 实验名称： | 实现三种磁盘调度算法 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18级计算机科学与技术创新班 | | | |
| 实验日期： | 2020.6.14 | | | | |
| 实验室： |  | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 一、实验目的

# 编程实现四种调度算法：

# 1) FCFS

# 2) SSTF

# 3) SCAN

# 二、实验要求

# 1) 结果要求可视化展示

# ⒊ 实现提示

# 根据课本所给的三种算法的原理，编程分别实现。在给出相同的

# 磁盘访问的请求序列和读写头起始位置，比较三种算法磁头服务序

# 列、平均寻道长度。

# 4. 输入和输出

# 1）输入

# 请求数、请求序列和读写头起始位置。

# 2）输出

# 磁头服务序列、平均寻道长度。

# 三、设计

【定义所有抽象数据类型，自定义函数的功能详细描述（伪代码表示），以及主程序的流程图。】

#include<stdio.h>

#include<math.h>

int num,sum,start,max;//磁道个数，磁道移动总数，开始磁道数，最长磁道号

int m=0,n=0,s[100],s1[100],c1[50],c2[50];

//s[]数组记录磁道序列

//用开始磁道数start来将序列划分成两个数组c1,c2

void creat()//初始化操作

{

int i,j,t;

printf("请输入磁道的个数：\n");

scanf("%d",&num);

printf("请输入从哪个磁道开始：\t\n");

scanf("%d",&start);

printf("请输入最长磁道号：\n");

scanf("%d",&max);

for(j=0;j<num;j++)

{

printf("请输入第%d个磁道\n",j+1);

scanf("%d",&s[j]);

if(s[j]>max) //当出现磁道数大于最长磁道号时，给出错误提示

{

printf("ERROR\n");

break;

}

}

printf("访问序列如下\n");

for(i=0;i<num;i++)//输出访问序列

{

printf("\t%d\t\n",s[i]);

}

int su=start;//将开始寻道的磁道赋值给变量su,以方便保存变量

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

if(su>s[i])//用开始磁道数start来将序列划分成两个数组c1,c2

c1[m++]=s[i];//初始时m为0

else

c2[n++]=s[i];//初始时n为0

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

for(j=i;j<m;j++)

if(c1[i]<c1[j])//将c1数组从大到小排序

{

t=c1[i];

c1[i]=c1[j];

c1[j]=t;

}

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

for(j=i;j<n;j++)

if(c2[i]>c2[j])//将c2数组从小到大排序

{

t=c2[i];

c2[i]=c2[j];

c2[j]=t;

}

}

void FCFS() //先来先服务算法

{

int i;

printf("先来先服务 FCFS\n");

printf("被访问的下一个磁道\t\t\t磁道号移动距离\n");

int su=start;//初始磁道数赋值给变量su

sum=0;

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

{

if(su<s[i])

s1[i]=s[i]-su;//s1数组记录每次移动磁道数，输出在界面上

else

s1[i]=su-s[i];

su=s[i];

sum+=s1[i];//记录移动磁道数

}

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

{

printf("\t%d\t\t\t\t\t%d\t\t\n",s[i],s1[i]);

}

printf("平均寻道长度：%.2f\n",sum\*1.0/num);

}

void SSTF() //最短寻道

{

printf("最短寻道 SSTF:\n");

printf("被访问的下一个磁道\t\t\t磁道号移动距离\n");

int s2[100],i;

sum=0;

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

s2[i]=s[i];

int min,temp,st=start,j,tag;

for(j=0;j<num;j++){

min=10000;

for(i=0;i<num;i++) //每次选出离当前磁道最近的磁道

{

temp=abs(s2[i]-st);

if(temp<min){

min=temp;

tag=i;

}

}

st=s2[tag];

printf("\t%d\t\t\t\t\t%d\t\t\n",s2[tag],min);

sum=sum+min; //记录每次的移动数

s2[tag]=10000;

}

printf("平均寻道长度:%.2f\n",sum\*1.0/num);

}

void SCAN()//扫描算法

{

int i;

printf("扫描算法 SCAN:\n");

printf("被访问的下一个磁道：\t\t\t磁道号移动距离：\n");

int su=start;

int s2[100];

sum=0;

for(i=0;i<n;i++) //先自内而外地进行移动磁头

s2[i]=c2[i];

for(i=0;i<m;i++) //c1c2数组是create函数已经分化好的排序数组

s2[i+n]=c1[i];

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

{

if(su<s2[i])

s1[i]=s2[i]-su;

else

s1[i]=su-s2[i];

su=s2[i];

sum+=s1[i];

}

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

{

printf("\t%d\t\t\t\t\t%d\t\t\n",s2[i],s1[i]);

}

printf("平均寻道长度：%.2f\n",sum\*1.0/num);

}

void CSAN()

{

int i;

printf("循环扫描 CSAN:\n");

printf("被访问的下一个磁道：\t\t\t磁道号移动距离：\n");

int su=start;

int j=0;

int s2[100];

sum=0;

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

s2[i]=c2[i];

for(i=m-1;i>=0;j++,i--)

s2[j+n]=c1[i];

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

{

if(su<s2[i])

s1[i]=s2[i]-su;

else

s1[i]=su-s2[i];

su=s2[i];

sum+=s1[i];

}

//sum=sum/num;

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

{

printf("\t%d\t\t\t\t\t%d\t\t\n",s2[i],s1[i]);

}

printf("平均寻道长度:%.2f\n",sum\*1.0/num);

}

void MENU()

{

int x;

creat(); //初始化磁道

do{

printf("磁盘调度\n");

printf("|--------请选择操作------------|\n");

printf("|------1.创建磁道--------------|\n");

printf("|------2.先来先服务FCFS--------|\n");

printf("|------3.最短寻道SSTF----------|\n");

printf("|------4.扫描算法 SCAN---------|\n");

printf("|------5.循环扫描算法 CSCAN----|\n");

printf("|------6.EXIT------------------|\n");

scanf("%d",&x);

switch(x)

{

case 2: FCFS();

break;

case 3:SSTF();

break;

case 4:SCAN();

break;

case 5:CSAN();

break;

case 6:

printf("再见！");

break;

}

}while(x<6);

}

int main() //主调函数

{

MENU();

return 0;

}

# 四、调试分析和思考

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

（1）先来先服务算法（FCFS）是一种比较简单的磁盘调度算法。它根据进程请求访问磁盘的先后次序进行调度。此算法的优点是公平、简单，且每个进程的请求都能依次得到处理，不会出现某一进程的请求长期得不到满足的情况。此算法由于未对寻道进行优化，在对磁盘的访问请求比较多的情况下，此算法将降低设备服务的吞吐量，致使平均寻道时间可能较长，但各进程得到服务的响应时间的变化幅度较小。

（2）最短寻道时间优先算法（SSTF）该算法选择这样的进程，其要求访问的磁道与当前磁头所在的磁道距离最近，以使每次的寻道时间最短，该算法可以得到比较好的吞吐量，但却不能保证平均寻道时间最短。其缺点是对用户的服务请求的响应机会不是均等的，因而导致响应时间的变化幅度很大。在服务请求很多的情况下，对内外边缘磁道的请求将会无限期的被延迟，有些请求的响应时间将不可预期。

（3）扫描算法（SCAN）电梯调度扫描算法不仅考虑到欲访问的磁道与当前磁道的距离，更优先考虑的是磁头的当前移动方向。例如，当磁头正在自里向外移动时，扫描算法所选择的下一个访问对象应是其欲访问的磁道既在当前磁道之外，又是距离最近的。这样自里向外地访问，直到再无更外的磁道需要访问才将磁臂换向，自外向里移动。这时，同样也是每次选择这样的进程来调度，即其要访问的磁道，在当前磁道之内，从而避免了饥饿现象的出现。由于这种算法中磁头移动的规律颇似电梯的运行，故又称为电梯调度算法。此算法基本上克服了最短寻道时间优先算法的服务集中于中间磁道和响应时间变化比较大的缺点，而具有最短寻道时间优先算法的优点即吞吐量较大，平均响应时间较小，但由于是摆动式的扫描方法，两侧磁道被访问的频率仍低于中间磁道。

（4）循环扫描算法（C-SCAN） 在扫描算法的基础上规定磁头单向移动来提供服务，回返时直接快速移动至起始端而不服务任何请求。

# 五、测试数据与结果

【列出你的测试结果，明确输入和输出数据。测试数据应该完整和严格，可以直接贴结果图。】

# C:\Users\wuyu\Documents\Tencent Files\1124274751\FileRecv\MobileFile\Image\4UXH8HZF3DLWR6MB6@}XXF7.png







