

### 

**实验报告**

**课程名称:** **实用操作系统教程**

**设计课题: 磁盘调度算法**

**指导教师: 熊婷**

**专 业：** 计算机科学与技术

**班 级：** 1982066班

**学 号：** 198206606

**姓 名：** 陈文龙

**成 员：** 陈文龙、姜宏达

**二O 二 一 年 12 月 25 日**

**目 录**

1.实验目的 1

2.实验要求 1

3.实验内容及原理 1

4.算法描述 1

4.1最短寻道时间优先算法 1

4.2先来先服务算法 2

4.3循环扫描算法 2

5.程序流程图 2

6.源代码 3

7.结果过程及截图 7

7.1测试数据 7

7.2三种算法下的测试结果 7

8.总结 8

参考文献： 9

# 1.实验目的

通过实验使学生了解并掌握最短寻道时间优先算法的相关特性。

# 2.实验要求

实验结束后要求每位同学都应该掌握最短寻道时间优先算法的基本原理。

# 3.实验内容及原理

(1)给定一组测试数据，如：20，44，40，4，80，12，76；读写头起始位置：40。创建文件job.txt，并导入测试数据：

20，44，40，4，80，12，76

42 7

注：42表示起始位置，7表示要访问的磁道数。

(2)编译源代码

(3)观察并结合运行结果研究实例代码中最短寻道时间优先算法的特点，补充先来先服务算法的代码，运行并分析比较这两种算法的优缺点及区别。

# 4.算法描述

## **4.1最短寻道时间优先算法**

这种算法的基本出发点是以磁头移动距离的大小作为优先的因素。它从当前磁头位置出发，选择离磁头最近的磁道为其服务。最短寻道时间优先算法使那些靠近磁头当前位置的申请可及时得到服务，防止了磁头大幅度来回摆动，减少了磁道平均查找时间。该算法没考虑磁头移动的方向，也没有考虑进程在队列中等待的时间，从而可能使移动臂不断花时间改变方向，还可能使一些离磁头较远的申请者在较长时间内得不到服务。

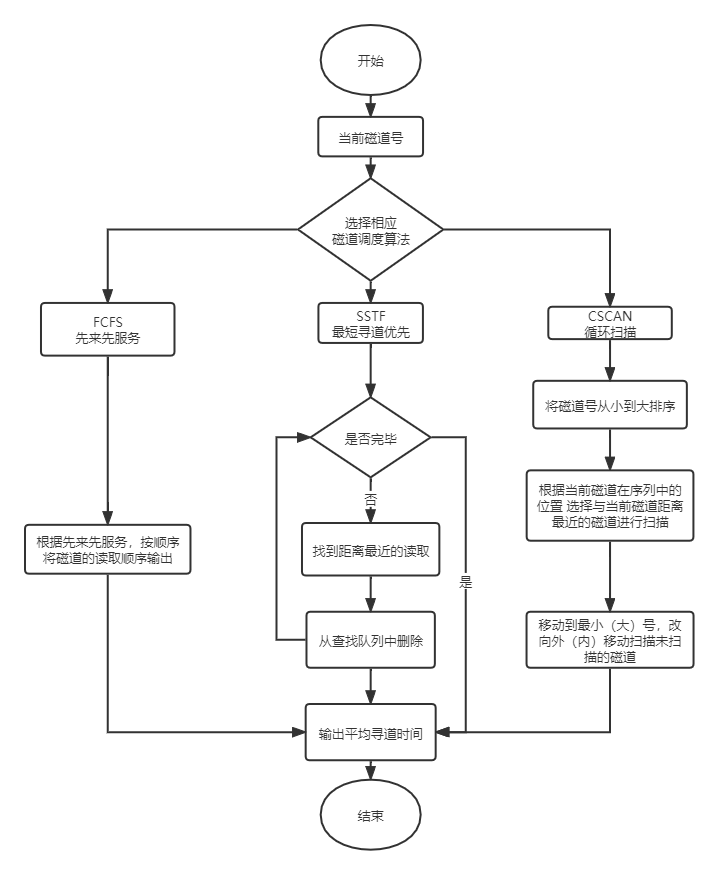
## **4.2先来先服务算法**

先来先服务算法根据访问请求的先后次序选择先提出访问请求的为之服务。先来先服务算法是磁盘调度的最简单的一种形式，它既容易实现，又公平合理。它的缺点是效率不高，相邻两次请求可能会造成最内到最外的柱面寻道，致使磁头反复移动，增加了服务时间，对机械的寿命也不利。

## **4.3循环扫描算法**

循环扫描算法是对扫描算法的改进。如果对磁道的访问请求时均匀分布的，当磁头到达磁盘的一端并反向运动时，落在磁头之后的访问请求相对较少。这是由于这些磁道刚被处理，而磁盘另一端的请求密度相当高，且这些访问请求等待的时间长。为了解决这种情况,循环扫描算法规定磁头单向移动。例如,只自里向外移动,当磁头移到最外的被访问磁道时,头立即返回到最里的欲访问磁道,即将最小磁道号紧接着最大磁道号构成循环,进行扫描。［1］

# 5.程序流程图



# 6.源代码

#include <iostream>

#include<algorithm>

#include<queue>

using namespace std;

int visit[100] = { 0 };//记录访问序列

int number;//记录访问序列数

int choose;//记录选择的算法

int position;//记录当前磁头位置

int  load[100] = { 0 };

int load1[100] = { 0 };

int load2[100] = { 0 };

int flag1[100] = { 0 };

void FCFS();

void SSTF();

void Le\_Rifind(int low, int high, double sum, int position);

void SCAN();

void CSCAN();

void FSCAN();

void print() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "输入1选择FCFS，输入2选择SSTF，输入3选择CSCAN" << endl;

    cout << "请选择磁盘调度算法:" << endl;

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

int main() {

    cout << "请输入磁盘访问序列数量：";

    cin >> number;

    cout << "磁盘访问序列：";

    for (int i = 0; i < number; i++) {

        cin >> visit[i];

    }

    for (int i = 0; i < number; i++) {

        load[i] = visit[i];

    }

    sort(load, load + number);

    cout << "当前磁头位置：";

    cin >> position;

    print();

    while (cin >> choose) {

        switch (choose) {

        case 1: FCFS();

            print();

            break;

        case 2:SSTF();

            print();

            break;

        case 3:CSCAN();

            print();

            break;

        default:return 0;

        }

    }

}

void FCFS() {//先来先服务

    int flag = position;

    double sum = 0;

    int cnt;

    int length;

    queue<int> q;

    for (int i = 0; i < number; i++) {

        q.push(visit[i]);

    }

    cout << "FCFS算法" << endl;

    cout << "下一个被访问的磁道" << "        " << "横跨的磁道数" << endl;

    for (int i = 0; i < number; i++) {

        cnt = q.front();

        q.pop();

        length = abs(flag - cnt);

        sum = sum + length;

        flag = cnt;

        cout << "    ";

        printf("%-22d", cnt);

        cout << length << endl;

    }

    cout << "总共寻道长度:" << sum << endl;

    sum = sum / number;

    cout << "平均寻道长度:" << sum << endl;

}

void SSTF() {//最短时间优先

    cout << "SSTF算法" << endl;

    int low = -1, high = -1;

    cout << "下一个被访问的磁道" << "        " << "横跨的磁道数" << endl;

    for (int i = 0; i < number; i++) {

        if (load[i] >= position) {

            high = i;

            low = high - 1;

            break;

        }

    }

    if (high == -1) {

        high = number;

        low = high - 1;

    }

    Le\_Rifind(low, high, 0, position);

}

void Le\_Rifind(int low, int high, double sum, int position) {

    int cnt, cnt1;

    if (low < 0 && high >= number) {

        cout << "总共寻道长度:" << sum << endl;

        cout << "平均寻道长度:" << sum / number << endl;

        return;

    }

    else if (low >= 0 && high >= number) {

        cnt = abs(load[low] - position);

        sum += cnt;

        cout << "    ";

        printf("%-22d", load[low]);

        cout << cnt << endl;

        Le\_Rifind(low - 1, high, sum, load[low]);

    }

    else if (low < 0 && high < number) {

        cnt = abs(load[high] - position);

        sum += cnt;

        cout << "    ";

        printf("%-22d", load[high]);

        cout << cnt << endl;

        Le\_Rifind(low, high + 1, sum, load[high]);

    }

    else if (low >= 0 && high < number) {

        cnt1 = abs(load[high] - position);

        cnt = abs(load[low] - position);

        if (cnt > cnt1) {

            sum += cnt1;

            cout << "    ";

            printf("%-22d", load[high]);

            cout << cnt1 << endl;

            Le\_Rifind(low, high + 1, sum, load[high]);

        }

        else {

            sum += cnt;

            cout << "    ";

            printf("%-22d", load[low]);

            cout << cnt << endl;

            Le\_Rifind(low - 1, high, sum, load[low]);

        }

    }

}

void CSCAN() {

    cout << "CSCAN算法" << endl;

    double sum = 0;

    int cnt;

    int k = 0;

    int flag = position;

    int low = -1, high = -1;

    for (int i = 0; i < number; i++) {

        if (load[i] >= position) {

            high = i;

            low = high - 1;

            break;

        }

    }

    if (high == -1) {

        high = number;

        low = high - 1;

    }

    cout << "下一个被访问的磁道" << "        " << "横跨的磁道数" << endl;

    for (int i = high; i < number; i++) {

        cnt = abs(load[i] - flag);

        sum += cnt;

        flag = load[i];

        cout << "    ";

        printf("%-22d", load[i]);

        cout << cnt << endl;

    }

    for (int i = flag; i < 200; i++) {

        if (flag1[i] == 0) {

            k++;

        }

    }

    for (int i = 0; i < load[0]; i++) {

        if (flag1[i] == 0) {

            k++;

        }

    }

    sum = sum + k + k;

    for (int i = 0; i <= low; i++) {

        cnt = abs(load[i] - flag);

        sum += cnt;

        flag = load[i];

        cout << "    ";

        printf("%-22d", load[i]);

        cout << cnt << endl;

    }

    cout << "总共寻道长度:" << sum << endl;

    cout << "平均寻道长度:" << sum / number << endl;

}

# 7.结果过程及截图

## **7.1测试数据**

7表示要访问的磁道数。

42表示磁头起始位置；

用于测试的磁盘访问序列为：20，44，40，4，80，12，76；



图1

## **7.2三种算法下的测试结果**

（1）FCFS-先来先服务算法

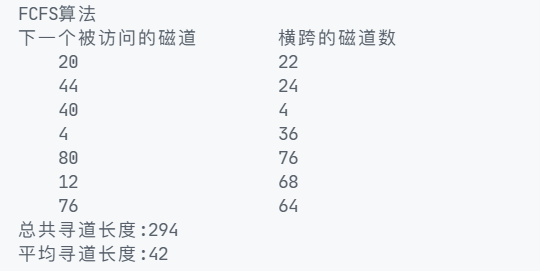


图2

（2）SSTF-最短寻道优先算法

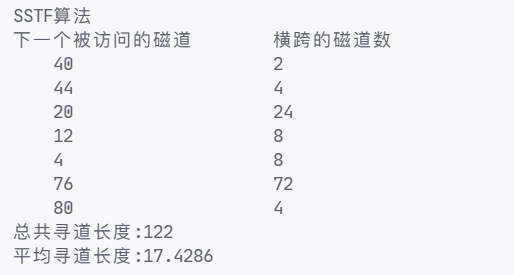


图3

（3）CSCAN-循环扫描算法

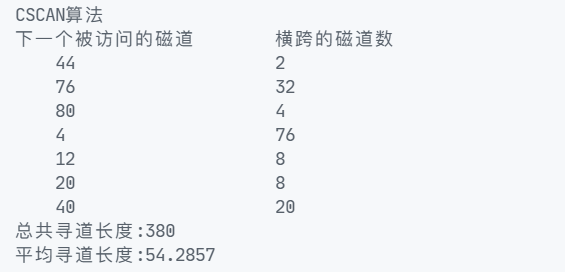


图4

# 8.总结

通过本次实验，我清楚的了解到磁盘调度的详细过程和三种主要的调度算法（先来先服务算法；最短寻道时间优先算法；循环扫描算法）以及三种调度算法之间的差异和共性，学习了解磁盘调度的工作原理及三种主要调度方法的工作原理，同时，也看到了经过优化的算法会带来的好处！并且在当中发现了自己的不足，对以前所学过的知识理解得不够深刻，掌握得不够牢固，看到了自己的实践经验还是比较缺乏，理论联系实际的能力还急需提高。

# 参考文献

［1］李建伟主编. 实用操作系统教程[M]. 北京：清华大学出版社，2016