

操作系统 -- 实验六实验报告

实验六：磁盘调度

一、实验题目

- (1) 观察和调整 Windows XP/7 的内存性能。
- (2) 了解和检测进程的虚拟内存空间。

二、实验目的

- (1) 了解磁盘结构以及磁盘上数据的组织方式。
- (2) 掌握磁盘访问时间的计算方式。
- (3) 掌握常用磁盘调度算法及其相关特性。

三、总体设计

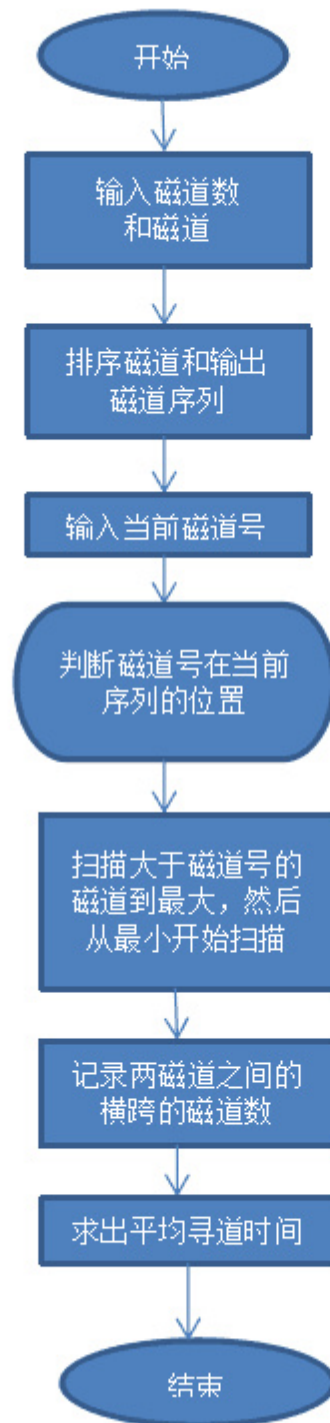
本实验通过编程模拟实现几种常见的磁盘调度算法。（1）测试数据：参见教材 P319-320，测试结果参见表 11.2。（2）使用 C 语言编程实现 FIFO、SSTF、SCAN、C-SCAN 算法（选做）。

程序主要包括：

- main() 主函数：控制整个程序流程；
- FFIO() 先进先出算法；
- SSTF() 最短服务时间算法；
- SCAN() 电梯扫描算法；
- CSCAN() 循环电梯算法。

四、详细设计

程序流程图：



程序代码：

```

/*
 *   Description: ---
 *   Author: Lynn
 *   Email: lgang219@gmail.com
 *   Create: 2018-07-05 21:08:39
 *   Last Modified: 2018-07-05 22:02:32
 */

```

```

#include "stdio.h"

```

```

#include "stdlib.h"
#define maxsize 1000 //定义最大数组域

//先进先出调度算法
void FIFO(int array[],int m)
{
    int sum=0,j,i,now;
    float avg;
    printf("\n 请输入当前的磁道号:");
    scanf("%d",&now);
    printf("\n FIFO 调度结果: ");
    printf("%d ",now);
    for(i=0; i<m; i++)
        printf("%d ",array[i]);
    sum=abs(now-array[0]);
    for(j=1; j<m; j++)
        sum+=abs(array[j]-array[j-1]); //累计总的移动距离
    avg=(float)sum/m; //计算平均寻道长度
    printf("\n 移动的总道数: %d \n",sum);
    printf(" 平均寻道长度: %f \n",avg);
}

//最短服务时间优先调度算法
void SSTF(int array[],int m)
{
    int temp;
    int k=1;
    int now,l,r;
    int i,j,sum=0;
    float avg;
    for(i=0; i<m; i++)
    {
        for(j=i+1; j<m; j++) //对磁道号进行从小到大排列
        {
            if(array[i]>array[j]) //两磁道号之间比较
            {
                temp=array[i];
                array[i]=array[j];
                array[j]=temp;
            }
        }
    }
    for( i=0; i<m; i++) //输出排序后的磁道号数组
        printf("%d ",array[i]);
    printf("\n 请输入当前的磁道号:");
    scanf("%d",&now);
    printf("\n SSTF 调度结果: ");
    if(array[m-1]<=now) //判断整个数组里的数是否都小于当前磁道号
    {
        for(i=m-1; i>=0; i--) //将数组磁道号从大到小输出
            printf("%d ",array[i]);
        sum=now-array[0]; //计算移动距离
    }
}

```

```

else if(array[0]>=now)//判断整个数组里的数是否都大于当前磁道号
{
    for(i=0; i<m; i++) //将磁道号从小到大输出
        printf("%d ",array[i]);
    sum=array[m-1]-now;//计算移动距离
}
else
{
    while(array[k]<now)//逐一比较以确定 k 值
    {
        k++;
    }

    l=k-1;
    r=k;
    //确定当前磁道在已排的序列中的位置
    while((l>=0)&&(r<m))
    {
        if((now-array[l])<=(array[r]-now))//判断最短距离
        {
            printf("%d ",array[l]);
            sum+=now-array[l];//计算移动距离
            now=array[l];
            l=l-1;
        }
        else
        {
            printf("%d ",array[r]);
            sum+=array[r]-now;//计算移动距离
            now=array[r];
            r=r+1;
        }
    }
    if(l== -1)
    {
        for(j=r; j<m; j++)
        {
            printf("%d ",array[j]);
        }
        sum+=array[m-1]-array[0];//计算移动距离
    }
    else
    {
        for(j=l; j>=0; j--)
        {
            printf("%d ",array[j]);
        }
        sum+=array[m-1]-array[0];//计算移动距离
    }
}
avg=(float)sum/m;
printf("\n 移动的总道数: %d \n",sum);

printf(" 平均寻道长度: %f \n",avg);

```

```

}

// 扫描算法
int SCAN(int array[], int m)
{
    int now=0;
    printf("SCAN 扫描算法");

    printf("\n 请输入当前的磁道号:");
    scanf("%d",&now);

    int temp=0;
    // 排序
    for(int i=0; i<m; i++)
    {
        for(int j=i+1; j<m; j++) //对磁道号进行从小到大排列
        {
            if(array[i]>array[j])//两磁道号之间比较
            {
                temp=array[i];
                array[i]=array[j];
                array[j]=temp;
            }
        }
    }

    // 输出结果
    int sum=0;
    float avg_steps=0;
    for(int i=0; i<m; i++)
    {
        if(array[i]>=now)
        {
            printf("\nSCAN 调度结果: %d ",now);
            for(int j=i; j<m; j++)
            {
                printf("%d ",array[j]);
                sum+=abs(array[j]-now);
                now=array[j];
            }

            if((i-1)!=0)
            {
                for(int j=i-1; j>=0; j--)
                {
                    printf("%d ",array[j]);
                    sum+=abs(array[j]-now);
                    now=array[j];
                }
            }
            avg_steps=float(sum/m);
            printf("\n移动的总道数: %d\n",sum);
            printf("平均寻道长度: %f\n\n",avg_steps);

            // 直接退出

```

```

        return 0;
    }
}
else
{
    if((i+1)==m)
    {
        printf("\nSCAN 调度结果: %d ", now);
        for(int j=i; j>=0; j--)
        {
            printf("%d ", array[j]);
            sum+=abs(array[j]-now);
            now=array[j];
        }
        avg_steps=float(sum/m);
        printf("\n移动的总道数: %d\n", sum);
        printf("平均寻道长度: %f\n\n", avg_steps);
    }
}

return 0;
}

// CSCAN 扫描算法
int CSCAN(int array[], int m)
{
    int now=0;
    printf("CSCAN 扫描算法");

    printf("\n 请输入当前的磁道号:");
    scanf("%d", &now);

    int temp=0;
    // 排序
    for(int i=0; i<m; i++)
    {
        for(int j=i+1; j<m; j++) //对磁道号进行从小到大排列
        {
            if(array[i]>array[j])//两磁道号之间比较
            {
                temp=array[i];
                array[i]=array[j];
                array[j]=temp;
            }
        }
    }

    // 输出结果
    for(int i=0; i<m; i++)
    {
        if(array[i]>=now)

```

```

    {
        printf("\nCSCAN 调度结果: %d ", now);
        for(int j=i; j<m; j++)
            printf("%d ", array[j]);
        for(int j=0; j<i; j++)
            printf("%d ", array[j]);
        // 直接退出
        return 0;
    }
    else
    {
        {
            if((i+1)==m)
            {
                printf("\nCSCAN 调度结果: %d ", now);
                for(int j=0; j<m; j++)
                    printf("%d ", array[j]);
            }
        }
    }
    return 0;
}

// 操作界面
int main()
{
    int c;
    int count;
    //int m=0;
    int cidao[maxsize]; //定义磁道号数组
    int i=0;
    int b;
    printf("\n ----- \n");
    printf("磁盘调度算法模拟");
    printf("\n ----- \n");
    printf("请先输入磁道数量: \n");
    scanf("%d", &b);
    printf("请先输入磁道序列: \n");

    for(i=0; i<b; i++)
    {
        scanf("%d", &cidao[i]);
    }
    printf("\n 磁道读取结果: \n");

    for(i=0; i<b; i++)
    {
        printf("%d ", cidao[i]); //输出读取的磁道的磁道号
    }

    count=b;
    printf("\n ");

```

```

while(1)
{
    printf("\n 算法选择:\n");
    printf(" 1、先进先出算法(FIFO)\n");
    printf(" 2、最短服务时间优先算法(SSTF)\n");
    printf(" 3、扫描算法(SCAN)\n");
    printf(" 4、循环扫描算法(C-SCAN)\n");
    printf(" 5. 退出\n");
    printf("\n");
    printf("请选择:");

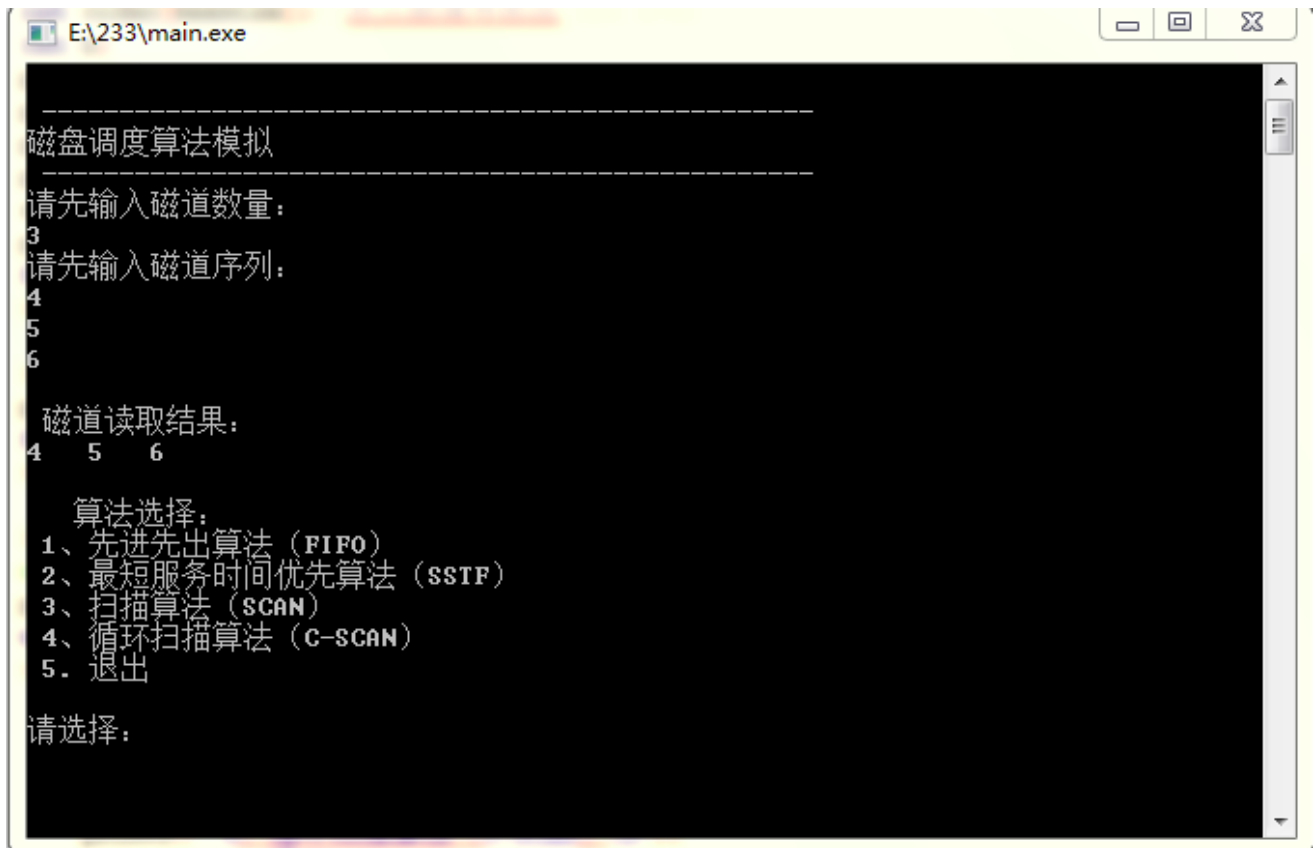
    scanf("%d",&c);
    if(c>5)
        break;

    switch(c)//算法选择
    {
        case 1:
            FIFO(cidao,count);//先进先出算法
            printf("\n");
            break;
        case 2:
            SSTF(cidao,count);//最短服务时间优先算法
            printf("\n");
            break;
        case 3:
            SCAN(cidao,count);//扫描算法,待补充!
            printf("\n");
            break;
        case 4:
            CSCAN(cidao,count);//循环扫描算法,待补充!
            printf("\n");
            break;
        case 5:
            exit(0);
            //
            //
    }
}
return 0;
}

```

五、实验结果与分析

程序运行结果：



```
-----  
磁盘调度算法模拟  
-----  
请先输入磁道数量:  
3  
请先输入磁道序列:  
4  
5  
6  
  
磁道读取结果:  
4 5 6  
  
算法选择:  
1、先进先出算法 (FIFO)  
2、最短服务时间优先算法 (SSTF)  
3、扫描算法 (SCAN)  
4、循环扫描算法 (C-SCAN)  
5. 退出  
  
请选择:
```

实验结果分析：

1. 关于FIFO算法：

- 1、算法思想：按访问请求到达的先后次序服务。
- 2、优点：简单，公平。
- 3、缺点：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，使磁头反复移动，增加了服务时间，对机械也不利。

2. 关于SSTF算法：

- 1、算法思想：优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先。
- 2、优点：改善了磁盘平均服务时间。
- 3、缺点：造成某些访问请求长期等待得不到服务。

3. 关于SCAN算法：

当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续扫描；否则改变移动方向，并为经过的访问请求服务，如此反复。

4. 关于循环扫描算法：

采用SCAN算法和C-SCAN算法时磁头总是严格地遵循从盘面的一端到另一端，显然，在实际使用时还可以改进，即磁头移动只需要到达最远端的一个请求即可返回，不需要到达磁盘端点。这种形式的SCAN算法和C-SCAN算法称为LOOK和C-LOOK调度。这是因为它们在朝一个给定方向移动前会查看是否有请求。注意，若无特别说明，也可以默认SCAN算法和C-SCAN算法为LOOK和C-LOOK调度。

六、小结与心得体会

- FIFO

FCFS算法根据进程请求访问磁盘的先后顺序进行调度，这是一种最简单的[调度算法](#)。该算法的优点是具有公平性。如果只有少量进程需要访问，且大部分请求都是访问簇聚的文件扇区，则有望达到较好的性能；但如果大量进程竞争使用磁盘，那么这种算法在性能上往往接近于随机调度。所以，实际磁盘调度中考虑一些更为复杂的调度算法。[1][](undefined)

- 1、算法思想：按访问请求到达的先后次序服务。
- 2、优点：简单，公平。
- 3、缺点：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，使磁头反复移动，增加了服务时间，对机械也不利。

- SSTF

SSTF算法选择调度处理的磁道是与当前磁头所在磁道距离最近的磁道，以使每次的寻找时间最短。当然，总是选择最小寻找时间并不能保证平均寻找时间最小，但是能提供比FCFS算法更好的性能。这种算法会产生“饥饿”现象。

- 1、算法思想：优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先。
- 2、优点：改善了磁盘平均服务时间。
- 3、缺点：造成某些访问请求长期等待得不到服务。

- SCAN 电梯算法

SCAN算法在磁头当前移动方向上选择与当前磁头所在磁道距离最近的请求作为下一次服务的对象。由于磁头移动规律与电梯运行相似，故又称为电梯调度算法。SCAN算法对最近扫描过的区域不公平，因此，它在访问局部性方面不如FCFS算法和SSTF算法好。

- CSCAN 循环电梯算法

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任何请求。由于SCAN算法偏向于处理那些接近最里或最外的磁道的访问请求，所以使用改进型的C-SCAN算法来避免这个问题。

采用SCAN算法和C-SCAN算法时磁头总是严格地遵循从盘面的一端到另一端，显然，在实际使用时还可以改进，即磁头移动只需要到达最远端的一个请求即可返回，不需要到达磁盘端点。这种形式的SCAN算法和C-SCAN算法称为LOOK和C-LOOK调度。这是因为它们在朝一个给定方向移动前会查看是否有请求。注意，若无特别说明，也可以默认SCAN算法和C-SCAN算法为LOOK和C-LOOK调度。