操作系统第五次上机报告

姓名：刘晨旭 学号：20232241110 班级：软件2306

仓库地址：https://github.com/AcidBarium/osHomework

实验目的

加深对操作系统设备管理技术的理解，掌握几种重要磁盘调度算法，提升编程能力和数据分析能力。

实验内容

示例程序展示了SSTF和SCAN两种磁盘调度算法。要求输入任意磁盘请求序列，显示响应过程并统计报告请求顺序和总寻道距离，以比较不同算法的优劣。程序采用C++语言，利用DiskArm类描述磁盘调度算法及其属性。

实验要求

编译运行示例程序，理解其代码结构；补充实现SCAN、C-SCAN、LOOK算法；使程序能随机生成磁盘柱面请求序列，动态观测各算法性能并进行比较分析。

详细代码

Fcfs

```
// 先来先服务算法
void DiskArm::Fcfs(void)
{
    int Current = CurrentCylinder;
    int Direction = SeekDirection;
    InitSpace("FCFS");
    cout << Current;
    for (int i = 0; i < RequestNumber; i++)
    {
        // 1 是向大头跑
        if (((Cylinder[i] >= Current) && !Direction) ||
            ((Cylinder[i] < Current) && Direction))
        {
            // 需要调头
            SeekChang++;             // 调头数加 1
            Direction = !Direction; // 改变方向标志
            // 报告当前响应的道号
            cout << endl
                 << Current << " -> " << Cylinder[i];
        }
        else // 不需调头，报告当前响应的道号
            cout << " -> " << Cylinder[i];
        // 累计寻道数，响应过的道号变为当前道号
        SeekNumber += abs(Current - Cylinder[i]);
        Current = Cylinder[i];
    }
    // 报告磁盘移臂调度的情况
    Report();
```

```
}
```

初始化当前磁道位置和寻道方向，然后依次处理每个请求。对于每个请求，首先判断是否需要调头：当请求磁道在当前磁道外侧且方向相反时，需要调头并增加调头计数器。无论是否调头，都会输出磁道移动路径，并累计移动距离到总寻道数中。最后更新当前磁道位置为刚处理的请求磁道位置。

处理完所有请求后，算法会输出调度报告。FCFS算法严格按照请求到达顺序处理，不考虑磁头移动优化，因此可能产生较多的寻道时间和调头次数，但实现简单且公平。

Sstf

```
// 最短寻道时间优先算法
void DiskArm::Sstf(void)
{
    int Shortest;
    int Distance = 999999;
    int Direction = SeekDirection;
    int Current = CurrentCylinder;
    InitSpace("SSTF");
    cout << Current;
    for (int i = 0; i < RequestNumber; i++)
    {
        // 查找当前最近道号
        for (int j = 0; j < RequestNumber; j++)
        {
            if (Cylinder[j] == -1)
                continue; //-1 表示已经响应过了

            if (Distance > abs(Current - Cylinder[j]))
            {
                // 到下一道号比当前距离近，下一道号为当前距离
                Distance = abs(Current - Cylinder[j]);
                Shortest = j;
            }
        }
        if (((Cylinder[Shortest] >= Current) && !Direction) ||
            ((Cylinder[Shortest] < CurrentCylinder) && Direction))
        {
            // 需要调头
            SeekChang++;             // 调头数加 1
            Direction = !Direction; // 改变方向标志
            // 报告当前响应的道号
            cout << endl
                 << Current << " -> " << Cylinder[Shortest];
        }
        else // 不需调头，报告当前响应的道号
            cout << " -> " << Cylinder[Shortest];
        // 累计寻道数，响应过的道号变为当前道号
        SeekNumber += abs(Current - Cylinder[Shortest]);
        Current = Cylinder[Shortest];
        // 恢复最近距离，销去响应过的道号
        Distance = 999999;
        Cylinder[Shortest] = -1;
```

```
    }
    Report();
}
```

程序每次从当前磁头所在道号出发，查找所有尚未响应的请求中与当前道号距离最近的一个，然后移动磁头到该道号，记录路径并累计寻道长度。如果最近的请求与当前磁头运动方向相反，则记录一次调头并改变方向，再继续处理请求。每处理完一个请求，就将该请求置为 -1 表示完成，循环直到所有请求完成，最后输出寻道总长度和调头次数等统计信息。

Scan

```
void DiskArm::Scan(void)
{
    int Current = CurrentCylinder;
    int Direction = SeekDirection;

    InitSpace("SCAN");
    int point = 0;
    for (int i = 0; i < RequestNumber; i++)
    {
        if (Cylinder[i] <= Current)
            point++;
    } // 标记
    sort(Cylinder, RequestNumber); // 升序排列
    cout << Current << " ";

    if (Direction == 0)
    {
        for (int i = point - 1; i >= 0; i--)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        if (Cylinder[0] != 0)
            cout << "->" << 0;
        SeekChang++;
        SeekNumber += abs(Current - 0);
        cout << endl;
        cout << 0;
        for (int i = point; i < RequestNumber; i++)
        {
            if (Cylinder[i] == 0)
                continue;
            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber += abs(Cylinder[RequestNumber - 1] - 0);
    }

    else if (Direction == 1)
    {
        for (int i = point; i < RequestNumber; i++)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        if (Cylinder[RequestNumber - 1] != MAXDISK)
            cout << "-> " << MAXDISK;
```

```
        SeekNumber += abs(MAXDISK - Current);
        SeekChang++;
        cout << endl;
        cout << MAXDISK;
        for (int i = point - 1; i >= 0; i--)
        {
            if (Cylinder[i] == MAXDISK)
                continue;
            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber += abs(MAXDISK - Cylinder[0]);
    }
    Report();
}
```

首先，它根据当前磁头所在柱面 `CurrentCylinder` 和移动方向 `SeekDirection`，将所有请求的柱面号 `Cylinder[]` 按升序排序。然后根据方向从当前位置出发，依次处理请求：

- **若方向为 0（向下）**：先从当前位置向 0 移动，依次访问所有小于等于当前位置的请求，若最小柱面不为 0，则访问 0，再掉头向上处理剩余请求。
- **若方向为 1（向上）**：先从当前位置向 `MAXDISK` 移动，依次访问所有大于等于当前位置的请求，若最大柱面不为 `MAXDISK`，则访问 `MAXDISK`，再掉头向下处理剩余请求。

过程中累计磁头移动的距离（`SeekNumber`）和变向次数（`SeekChang`），最后调用 `Report()` 输出调度结果。

CScan

```
void DiskArm::CScan(void)
{
    int Current = CurrentCylinder;
    int Direction = SeekDirection;
    InitSpace("CSCAN");
    int point = 0;
    for (int i = 0; i < RequestNumber; i++)
    {
        if (Cylinder[i] <= Current)
            point++;
    }
    sort(Cylinder, RequestNumber); // 升序排列
    cout << Current << " ";

    if (Direction == 0)
    {
        for (int i = point - 1; i >= 0; i--)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        if (Cylinder[0] != 0)
            cout << "-> " << 0; // 向左到0
        cout << endl;
```

```
        cout << 0;
        cout << "-> " << MAXDISK;
        SeekChang++;
        SeekNumber += abs(Current - 0); // 向左移动到0的距离
        SeekNumber += MAXDISK;          // 从0到MAXDISK
        cout << endl;
        SeekChang++;
        cout << MAXDISK;
        for (int i = RequestNumber - 1; i >= point; i--)
        {
            if (Cylinder[i] == MAXDISK)
                continue;

            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber +=
            abs(MAXDISK - Cylinder[point + 1]); // MAXDISK到最后一个访问点的距离
    }
    else if (Direction == 1)
    {
        for (int i = point; i < RequestNumber; i++)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        if (Cylinder[RequestNumber] != MAXDISK)
            cout << "-> " << MAXDISK;
        cout << endl;
        cout << MAXDISK << "-> " << 0 << endl;
        SeekNumber += abs(MAXDISK - Current) + MAXDISK;
        SeekChang++;
        SeekChang++;
        cout << 0;
        for (int i = 0; i <= point - 1; i++)
        {
            if (Cylinder[i] == 0)
                continue;

            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber += abs(Cylinder[point - 1] - 0);
    }
    Report();
}
```

1. 根据当前磁头位置 `CurrentCylinder` 和请求柱面号 `Cylinder[]`，将所有请求按升序排序。
2. 确定当前位置在排序数组中的插入点 `point`。
3. 根据 `SeekDirection` 判断当前磁头方向：
    - **方向 0（向左）**：从当前位置向 0 方向依次访问小于等于当前柱面的请求，若未到 0，则访问 0，再**跳到** `MAXDISK`，继续从最大柱面开始向下访问剩余请求。
    - **方向 1（向右）**：从当前位置向 `MAXDISK` 方向依次访问大于等于当前柱面的请求，若未到 `MAXDISK`，访问 `MAXDISK`，然后**跳到 0**，再从最小柱面向上访问剩余请求。

Look

```
void DiskArm::Look(void)
{
    int Current = CurrentCylinder;
    int Direction = SeekDirection;
    InitSpace("LOOK");
    int point = 0;
    for (int i = 0; i < RequestNumber; i++)
    {
        if (Cylinder[i] <= Current)
            point++;
    }
    sort(Cylinder, RequestNumber); // 升序排列
    cout << Current << " ";

    if (Direction == 0)
    {
        for (int i = point - 1; i >= 0; i--)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        SeekChang++;
        SeekNumber += abs(Current - Cylinder[0]);
        cout << endl;
        cout << Cylinder[0];
        for (int i = point; i < RequestNumber; i++)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber += abs(Cylinder[RequestNumber - 1] - Cylinder[0]);
    }

    else if (Direction == 1)
    {
        for (int i = point; i < RequestNumber; i++)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber += abs(Cylinder[RequestNumber - 1] - Current);
        SeekChang++;
        cout << endl;
        cout << Cylinder[RequestNumber - 1];
        for (int i = point - 1; i >= 0; i--)
        {
            cout << "-> " << Cylinder[i] << " ";
        }
        SeekNumber += abs(Cylinder[RequestNumber - 1] - Cylinder[0]);
    }
    Report();
}
```

- 先根据当前磁头位置 `CurrentCylinder` 和请求队列，对请求的柱面号数组 `Cylinder[]` 进行升序排序，并找到当前位置在请求中的分割点 `point`。

- 根据磁头移动方向 `SeekDirection`：
  - **方向为 0（向下）**时，磁头先向当前请求中小于等于当前位置的最大请求方向移动，访问这些请求，不用移动到磁盘起点 0（区别于 SCAN），然后再反向访问剩余较大的请求。
  - **方向为 1（向上）**时，磁头先向大于等于当前位置的请求移动，访问这些请求，然后反向访问较小的请求，同样不必移动到最大柱面 `MAXDISK`。
- 计算实际移动的磁头寻道长度 `SeekNumber` 和变向次数 `SeekChang`，最后调用 `Report()` 输出结果。

LOOK 算法在 SCAN 的基础上避免了磁头无谓地移动到磁盘边界，只扫到最远的请求位置后再反向移动，更加高效。

运行结果以及分析

```
生成的11个随机数: 86 63 96 77 194 92 139 39 159 17 2
Please input Current cylinder :Please input Current Direction (0/1) :Please input
Request Numbers :Please input Request cylinder string :
FCFS
148
148 -> 86 -> 63
63 -> 96
96 -> 77
77 -> 194
194 -> 92
92 -> 139
139 -> 39
39 -> 159
159 -> 17 -> 2
Seek Number: 780
Chang Direction: 9
AVG:70.9091

SSTF
148
148 -> 139
139 -> 159 -> 194
194 -> 96 -> 92 -> 86 -> 77 -> 63 -> 39 -> 17 -> 2
Seek Number: 256
Chang Direction: 3
AVG:23.2727

SCAN
148 -> 159 -> 194 -> 200
200-> 139 -> 96 -> 92 -> 86 -> 77 -> 63 -> 39 -> 17 -> 2
Seek Number: 250
Chang Direction: 1
AVG:22.7273

CSCAN
148 -> 159 -> 194 -> 200
200-> 0
0-> 2 -> 17 -> 39 -> 63 -> 77 -> 86 -> 92 -> 96 -> 139
Seek Number: 391
Chang Direction: 2
```

```
AVG:35.5455

LOOK
148 -> 159 -> 194
194-> 139 -> 96 -> 92 -> 86 -> 77 -> 63 -> 39 -> 17 -> 2
Seek Number: 238
Chang Direction: 1
AVG:21.6364
程序执行完成
```

- **FCFS（先来先服务）**：
  - 按请求顺序依次访问，路径较为随意，导致磁头频繁往返。
  - 变向次数较多（9次），寻道距离最长（780）。
- **SSTF（最短寻道时间优先）**：
  - 每次访问最近的请求，寻道路径更紧凑。
  - 变向次数减少到3次，寻道长度明显减少（256）。
- **SCAN（电梯算法）**：
  - 磁头单方向扫描到磁盘边界（200），再反向扫描。
  - 变向次数1次，寻道距离250，比SSTF略高，但变向更少，调度更有规律。
- **CSCAN（循环扫描）**：
  - 磁头扫描到边界200后，跳回0重新扫描。
  - 变向次数2次，寻道距离391，较SCAN略高，因跳跃回起点增加了距离。
- **LOOK（改进的SCAN）**：
  - 磁头只扫描到最远请求（194）后反向，不扫描到边界。
  - 变向次数1次，寻道距离最短（238），表现最好。

- **LOOK 算法整体表现最佳**，寻道距离最短，变向次数最低，平均寻道长度也最低，体现出高效性。
- **FCFS 效率最差**，因为没有优化磁头移动顺序。
- **SSTF 和 SCAN 表现也不错，适合不同场景使用。**
- **CSCAN 虽然保证公平扫描，但跳跃回起点增加寻道距离。**

结论

- 优化磁头访问顺序能大幅降低寻道距离和变向次数，提升性能。
- LOOK 算法在这组请求下效果最好，适合实际磁盘调度。
- FCFS 适合简单实现，但代价是较高的寻道开销。
- SCAN/CSCAN 平衡公平和效率，常用于实际操作系统。