COA2021-programming09

Good luck and have fun!

1 实验要求

1.1 Disk

在 Disk 类中，实现磁头寻道相关的两个方法。

```
public void seek(int start)
public void addPoint()
```

1.2 Scheduler

在 Scheduler 类中，实现三个磁盘调度算法：先来先服务算法、最短寻道时间优先算法、扫描算法，并计算平均寻道长度。

```
public double FCFS(int start, int[] request)
public double SSTF(int start, int[] request)
public double SCAN(int start, int[] request, boolean direction)
```

2 实验攻略

2.1 实验概述

本次实验的主体部分为外部存储器——磁盘的模拟。

在真实的计算机系统中，磁盘所占空间一般都比较大，因此在堆栈中开辟数组的方法来进行模拟并不现实。因此，我们选择使用二进制文件来模拟磁盘，将对磁盘的读写转化为对文件的读写。在这个磁盘中，我们已经实现好了基本的读写功能，大家的工作主要集中在模拟磁头结构上面。

由于磁盘的模拟比较简单，我们还加入了对磁盘调度算法的考察。但实际上，由于在这个简单的系统中（包括我们将来要实现的功能）一次只能运行一个任务，并不涉及到多任务的调度。因此，我们简化了这部分内容，你将会看到磁盘调度算法类与我们的模拟磁盘并没有太大关系。

接下来，我们将对 Disk 类的源码进行一个简单的解读。

2.2 代码导读

2.2.1 代码结构

```
.
│ pom.xml
│ README.md
│
└─src
  ├─main
  │ └─java
  │   ├─cpu
  │   │
  │   ├─memory
  │   │ │ Memory.java
  │   │ │
  │   │ ├─cache
  │   │ │
  │   │ └─disk
  │   │       Disk.java # 磁盘类，需要修改
  │   │       Scheduler.java # 磁盘调度算法类，需要修改
  │   │
  │   └─util
  │
  └─test
```

2.1.2 磁盘存储结构模拟

我们模拟了一个 128MB 大小的磁盘，规定该磁盘拥有 16 个磁头，每个盘面上有 256 个磁道，每个磁道上有 64 个扇区，每扇区包含 512 字节的数据。具体属性定义如下

```
public static int DISK_SIZE_B = 128 * 1024 * 1024;    // 磁盘大小 128 MB

public static final int DISK_HEAD_NUM = 16;    // 磁头数
public static final int TRACK_NUM = 256;      // 磁道数
public static final int SECTOR_PER_TRACK = 64;  // 每磁道扇区数
public static final int BYTE_PER_SECTOR = 512;  // 每扇区字节数
```

注：磁盘存储容量 = 磁头数 × 磁道(柱面)数 × 每道扇区数 × 每扇区字节数

同时，我们定义了一个文件作为虚拟磁盘。这样，我们就可以将磁盘里面的数据统统放到该文件中，也就可以将对磁盘的操作转化为文件的操作了。文件定义如下

```
private static File disk_device;   // 虚拟磁盘文件
```

与 cache 和 memory 类似，我们同样使用单例模式来构造 Disk 类。在 Disk 类的构造函数中，你可以看到我们对虚拟磁盘文件进行了创建和初始化。具体可自行查看源码。

为了方便操作，在这个磁盘中，我们的数据是线性存放的。具体来说，每个扇区的数据连续存放，扇区 0 占据了文件的前 512 个字节，扇区 1 占据了文件的第 512 到第 1023 个字节，以此类推，ID 区域、间隙、同步字节、CRC 等区域我们没有进行模拟。同理，磁道与磁道之间连续，盘面与盘面之间也连续。

2.1.3 磁盘读写功能模拟

我们提供了已实现好的磁盘读写方法如下

```
public char[] read(String addr, int len)
public void write(String addr, int len, char[] data)
```

这两个方法要干的事情十分简单，就是在我们创建的虚拟磁盘文件中读写数据，同时调整磁头的位置。大家可以自行查看源码。

需要额外说明的是，读写方法中接收的 addr 参数，是二进制表示的该数据起始位置在虚拟磁盘文件中的字节数。由于真实磁盘的地址表示已经超出了本课程的范畴，因此我们用这种方法简单地模拟磁盘中的“地址”。至于计算机内存的地址表示，我们将会在虚拟内存的章节学到。

2.1.4 磁头的模拟

在 Disk 类中，我们定义了一个磁头对象如下

```
private final DiskHead disk_head = new DiskHead(); // 磁头
```

DiskHead 类是 Disk 类中的一个内部类，它记录了自己当前所在的位置。核心数据结构如下

```
private static class DiskHead {
  int track = 0;  // 当前所在磁道号
  int sector = 0; // 当前所在扇区号
  int point = 0;  // 当前所在扇区内部的字节号
}
```

同时，我们在 DiskHead 类中提供了 seek 方法和 addpoint 方法，用于调整磁头的位置。其中，seek 方法表示每次数据读写之前将磁头移动到指定位置，addPoint 表示将磁头往后移动一个字节。这样，我们就成功模拟了一个十分简单的磁头。

2.3 实现指导

2.3.1 Disk

在 Disk 类的 read 和 write 方法中，我们会对 DiskHead 类的 seek 方法和 addPoint 方法进行调用。因此，你需要结合 read 和 write 方法的源码，理解 seek 方法和 addPoint 方法在其中起到什么作用，然后实现这两个方法。

注意，由于我们规定该磁盘有 16 个磁头即 16 个盘面，所以每个盘面的大小为 8MB。在不同盘面上，磁头的位置都是相同的（具体可以看 ppt 上的图）。因

此，在我们的模拟规则下，第 0 个字节、第 8MB 个字节、第 16MB 个字节（以此类推），它们的磁头位置都应该是相同的。

2.3.2 Scheduler

如 2.1 所述，Schedule 类的内容相对独立。具体来说，每个方法都会传入磁头初始磁道号与请求访问的磁道号数组，你需要计算出平均寻道长度并返回。

注意，在 SCAN 算法中，你将会用到磁道总数，这个数字需要与 Disk 类中的 TRACK_NUM 保持一致。