COA2020

1.实验要求

CRC

本次作业要求大家实现一个通用的CRC计算器

```
    /**
     * CRC计算器
     * @param data 数据流
     * @param polynomial 多项式
     * @return CheckCode
     */
    public static char[] Calculate(char[] data, String polynomial) {
        return null;
    }

    /**
     * CRC校验器
     * @param data 接收方接受的数据流
     * @param polynomial 多项式
     * @param CheckCode CheckCode
     * @return 余数
     */
    public static char[] Check(char[] data, String polynomial, char[]
CheckCode){
        return null;
    }
```

磁盘读写

本次作业要求大家实现磁盘的读写

```
    /**
     * 读磁盘
     * @param eip
     * @param len
     * @return
     */
    public char[] read(String eip, int len) {
        return null;
    }

    /**
```

```
     * 写磁盘（供后面的作业调用）
     * @param eip
     * @param len
     * @param data
     */
    public void write(String eip, int len, char[] data) {
    }

    /**
     * 写磁盘（地址为Integer型）
     * @param eip
     * @param len
     * @param data
     */
    public void write(int eip, int len, char[] data) {

    }
```

注意，在磁盘读写的实现中，需要加入CRC，并存在扇区中，因为校验和和数据在磁盘中都是字节形式存在，但是校验是在Bit级别上进行运算的，所以大家需要实现两个转换方法

```
    /**
     * 将Byte流转换成Bit流
     * @param data
     * @return
     */
    public static char[] ToBitStream(char[] data) {
        return null;
    }

    /**
     * 将Bit流转换为Byte流
     * @param data
     * @return
     */
    public static char[] ToByteStream(char[] data) {
        return null;
    }
```

2.相关资料

Time Machine

为了方便大家调试代码，我们建议大家使用`git`来管理自己的代码，seecoder上确实也是用的`git`，为了降低大家的学习成本，我直接将大家会用到的内容写在这里。在`git`中，默认是`master`分支，这点可以通过`git log`来确定，如果大家想尝试另一种实现，在现有的版本上进行修改的话，修改错误还需要通过查看`git log`来进行时光倒流，这非常的麻烦。

```
git checkout -b new_branch_name  # 创建一个新的分支，名字为new_branch_name
```

这条指令可以创建一个新的分支并且切换到这个分支，`git checkout branch_name`可以用来切换分支

```
git branch
```

这条指令可以查看当前项目中的所有分支

```
git branch -D branch_name
```

这条指令可以删除某个分支。最重要的指令是下面这条指令

```
git merge branch_name
```

当你在某个分支上修改完后，希望把代码合并到`master`分支上，步骤是这样的

```
#now suppose the branch is mmu
git checkout master # switch to master
git merge mmu # merge code from mmu
```

希望大家善用`git`，这会给大家的调试和试错提供非常大的帮助。

> 请使用master分支进行提交

Disk (这部分内容对作业毫无帮助)

磁盘是计算机系统中用来存储数据的区域。这部分内容任老师上课已经详细介绍过，在这里不做赘述。但是作为相关资料，我想给大家拓展一点文件系统的知识。虽然在我们操作主存的时候，Disk已经被我们抽象成了一个大型的数组，但是对于计算机的用户来说，这还远远不够。首先，磁盘是一个随机读写的设备，我们文件的不同字节是散落在数组的不同地方的，如果我们想要访问一个文件，我们就需要有一种组织文件的形式，这就是文件系统的出发点。当然，除了访问文件，我们很自然的希望我们可以支持修改文件，创造文件，删除文件等一系列操作。

在操作系统初始化（非正式描述）时，它会用将磁盘这个抽象用一种特定的格式初始化，在Windows上是FAT格式，在Linux上是EXT格式。这两种格式互有优劣。一般来说，一个完整的文件系统需要包含以下功能：

- 按名存取
- 目录的建立和维护
- 逻辑文件到物理文件的转换
- 存储空间的分配和管理
- 数据保密、保护和共享
- 提供一组用户使用的API

按名存取

我们有两种选择，一种是将名字和文件内容一起存放，另一种是放在另一片区域存放，但是名字和内容一起意味着我们读文件的时候需要跳过文件名，这样就会对文件操作带来不便。在EXT文件系统中使用INode来记录文件信息（创建日期等），在磁盘中专门有一块区域存放INode，我们可以将文件名也放入INode中。

```
class Inode{
  char filename[32];
  off_t offset;
}
```

分配与组织

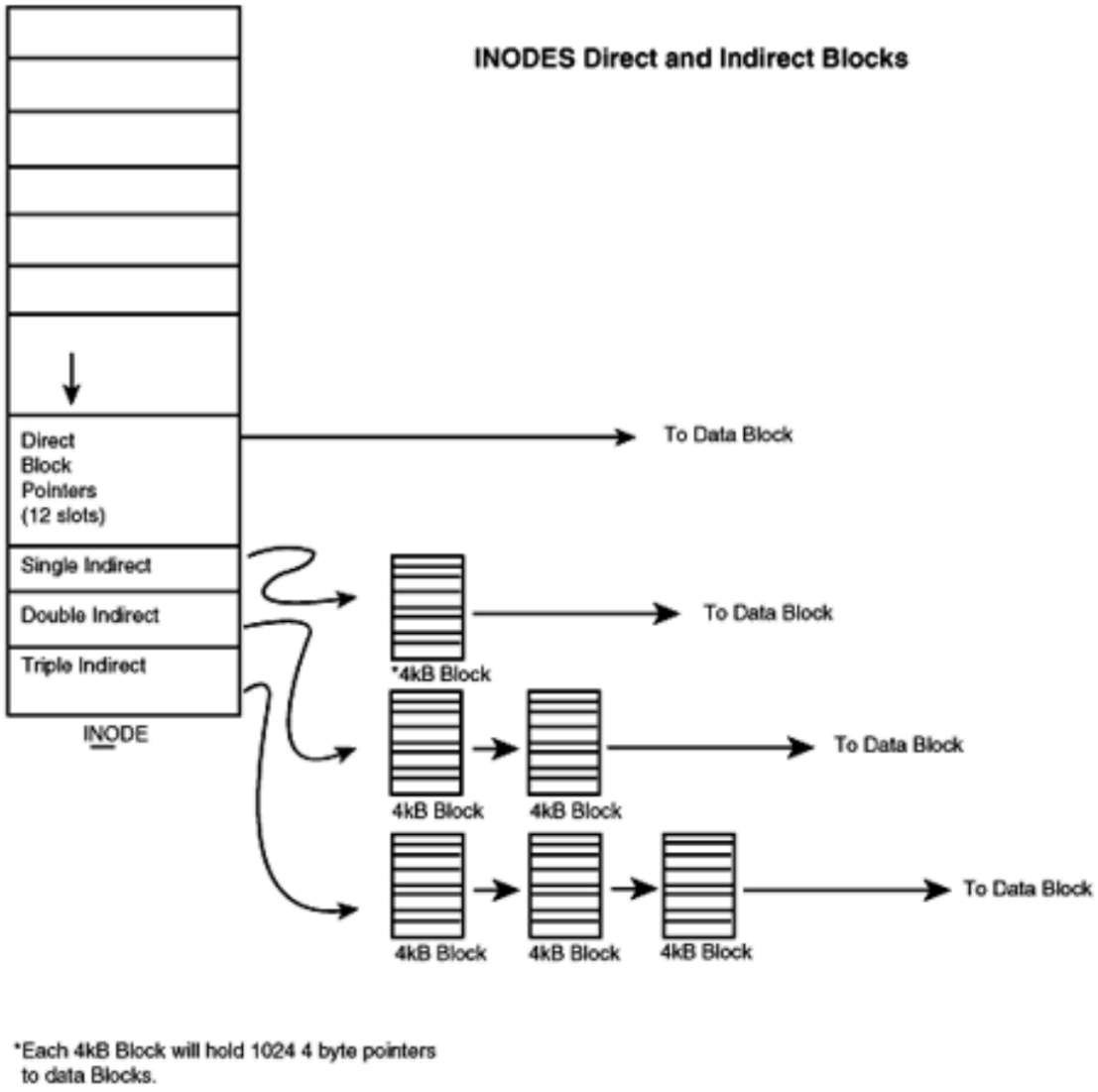
文件系统必须要能够为文件动态分配空间。因为文件的大小是不固定的。所以我们的INode中需要加入表示文件大小的属性

```
class Inode{
  char filename[32];
  off_t offset;
  size_t size;
}
```

顺序存放会遇到非常多的问题，就像大家在课上听任老师讲的关于内存碎片的内容相似，顺序存放也会导致磁盘出现很多磁盘碎片。它们占用的空间很难被再次利用。因此我们需要一种组织形式，类似于分页的想法，我们引入一个“块”的概念，每个文件都由若干个块组成，这样我们就不需要顺序存储了，完全可以使用类似页表的结构来组织文件。但是使用块来组织文件会对系统的性能有很大的影响。

- 需要支持文件的随机访问(访问一个文件的不同部分的效率都应该一样), 链表形式的组织方式显然不太合适, 因为访问的位置越靠后, 所花时间越长（这是FAT的实现）
- 需要方便地支持块的插入和删除, 这是因为文件的大小经常发生变化, 数组形式的组织方式显然不太合适, 因为向数组中插入元素可能需要申请更大的空间, 涉及到整个数组的拷贝, 频繁地进行这样的操作会严重影响效率, 磁盘的寿命也会有所下降
- 组织方式带来的额外开销不能太大, 一种需要花费大量磁盘空间来维护的组织方式显然是不被接受的

在EXT文件系统中，我们使用多级索引来组织块



对应的结构是这样

```
class Inode{
  char filename[32];
  size_t size;
  block_t index[15];
}
```

index[0]~index[11]是数据块的索引，它们指向的块中存储了文件数据

index[12]指向的块中包含了若干数据块的索引，因此index[12][0],...才是数据块的索引。index[13]是二级索引块的索引，index[14]是三级索引块的索引。

这种数据结构使得文件中任意数据块的访问, 插入和删除操作的时间复杂度都是O(1), 因为访问文件中任何位置的数据最多在访问5个磁盘块后达到; 另一方面, 这种数据结构的开销取决于索引块的数目, 小文件只需要用到前几个index就足够了, 不需要使用索引块; 而对于大型文件, 在块大小为4KB, 块索引号为32位整数的情况下, 用到的索引块的大小大约为文件大小的千分之一.

目录

大家可能听说过在Linux系统中，一切都是文件。不过不管在哪里，目录几乎都是被当成文件来对待的。它和文件没有区别，只不过它记录的信息是文件的位置。它是一张记录了当前目录文件信息的表，每一个表项都记录了该目录中文件的文件名和Inode索引。

```
class DirEntry{
  char filename[32];
  inode_t inode;
}
```

为了区分目录和普通文件（两种文件的逻辑组织形式不同），我们还需要在INode中添加一个文件类型的属性

```
class Inode{
  char filename[32];
  size_t size;
  block_t index[15];
  int type;
}
```

Cyclic Redundancy Check

CRC是一种在数据流传输过程中校验数据是否出错了的一种散列函数，数据像流一样输入函数后会产生一个校验码，将校验码附在数据的后面形成新的数据。在接收端，我们会将输入的数据重新输入到CRC中，如果最后输出的结果全为0，那表示我们的数据没有出错。和海明码、奇偶校验相同，它的错误检测能力是有范围的。我们先介绍它的原理

在二进制的世界中只存在0和1，我们考虑一个多项式系数的集合S，如果这个集合中只有0和1，我们称这个多项式系数的集合为有限域GF(2)（所有的系数会模2同余）的多项式环。给大家一个简单的例子体会一下

$$(x^3+x)+(x+1)=x^3+2x+1=x^3+1$$

同样的，上面这个加法可以看成是二进制相加

$$110+011=101$$

对于系数是这个集合中的多项式，我们也可以对他做除法

$$\frac{x^3+x^2+x}{x+1}=(x^2+1)-\frac{1}{x+1}\rightarrow(x^3+x^2+x)=(x^2+1)(x+1)-1$$

这样我们就得到了商x^2+1和余数-1。那么这些和上课讲的CRC有什么关系呢？我们可以将上述等式进一步转换成$(x^2+x+1)x=(x^2+1)(x+1)-1$，我们回忆一下CRC中将数据的后面填充0的操作，其实就是在一个多项式后面乘上x^n次方，n是除多项式$(x+1)$的最高次，这样我们就把CRC和多项式除法联系了起来。

一般的，我们可以将CRC写成如下形式

$$M(x) \cdot x^n = Q(x) \cdot K(x) - R(x)$$

$M(x)$是我们的数据对应的多项式，$K(x)$就是我们的除多项式，n是除多项式的最高次幂，$R(x)$是我们的校验和。对这些多项式，做除法的时候并不是相减而是异或。

3.实验攻略

本次实验难度一般，所以在这里只简单的介绍一下我们的数据结构

数据结构

磁盘抽象

```
/**
 * 600 Bytes/Sector
 */
private class Sector {
    char[] gap1 = new char[17];
    IDField idField = new IDField();
    char[] gap2 = new char[41];
    DataField dataField = new DataField();
    char[] gap3 = new char[20];
}

/**
 * 7 Bytes/IDField
 */
private class IDField {
    char SynchByte;
    char[] Track = new char[2];
    char Head;
    char sector;
    char[] CRC = new char[2];
}

/**
 * 515 Bytes/DataField
 */
private class DataField {
    char SynchByte;
    char[] Data = new char[512];
    char[] CRC = new char[2];
}

/**
 * 128 sectors pre track
 */
private class Track {
    Sector[] sectors = new Sector[SECTOR_PRE_TRACK];
```

```
        Track() {
            for (int i = 0; i < SECTOR_PRE_TRACK; i++) sectors[i] = new
Sector();
        }
    }


    /**
     * 16 tracks pre platter
     */
    private class Platter {
        Track[] tracks = new Track[TRACK_PRE_PLATTER];

        Platter() {
            for (int i = 0; i < TRACK_PRE_PLATTER; i++) tracks[i] = new
Track();
        }
    }

    /**
     * 8 platter pre Cylinder
     */
    private class Cylinder {
        Platter[] platters = new Platter[PLATTER_PRE_CYLINDER];

        Cylinder() {
            for (int i = 0; i < PLATTER_PRE_CYLINDER; i++) platters[i] = new
Platter();
        }
    }
```

为了降低磁盘寻址的复杂度，我们假设每个柱面的上的磁道是线性排列的。我们按照PPT给出了扇区与磁道的数据结构，使用了一种树形结构来组织柱面、扇区、磁道。大家可以尝试修改我们的数据结构，让它更加真实。

磁头

磁头记录了当前自己所在的位置

```
    /**
     * 磁头
     */
    private class disk_head {
        int cylinder;
        int platter;
        int track;
        int sector;
```

```
        int point;

        /**
         * 调整磁头的位置
         */
        public void adjust() {
            if (point == BYTE_PRE_SECTOR) {
                point = 0;
                sector++;
            }
            if (sector == SECTOR_PRE_TRACK) {
                sector = 0;
                track++;
            }
            if (track == TRACK_PRE_PLATTER) {
                track = 0;
                platter++;
            }
            if (platter == PLATTER_PRE_CYLINDER) {
                platter = 0;
                cylinder++;
            }
            if (cylinder == CYLINDER_NUM) {
                cylinder = 0;
            }
        }

        /**
         * 磁头回到起点
         */
        public void Init() {
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
            cylinder = 0;
            track = 0;
            sector = 0;
            point = 0;
            platter = 0;
        }

        /**
         * 将磁头移动到目标位置
         * @param start
         */
        public void Seek(int start) {
            try {
```

```
                Thread.sleep(0);
            } catch (Exception e) {
                e.printStackTrace();
            }
            for (int i = cylinder; i < CYLINDER_NUM; i++) {
                for (int t = platter; t < PLATTER_PRE_CYLINDER; t++) {
                    for (int j = track; j < TRACK_PRE_PLATTER; j++) {
                        for (int z = sector; z < SECTOR_PRE_TRACK; z++) {
                            for (int k = point; k < BYTE_PRE_SECTOR; k++) {
                                if ((i * PLATTER_PRE_CYLINDER *
TRACK_PRE_PLATTER * SECTOR_PRE_TRACK * BYTE_PRE_SECTOR + t * TRACK_PRE_PLATTER
* SECTOR_PRE_TRACK * BYTE_PRE_SECTOR + j * SECTOR_PRE_TRACK * BYTE_PRE_SECTOR +
z * BYTE_PRE_SECTOR + k) == start) {
                                    cylinder = i;
                                    track = j;
                                    sector = z;
                                    point = k;
                                    platter = t;
                                    return;
                                }
                            }
                        }
                    }
                }
            }
            Init();
            Seek(start);
        }
```

大家也可以自己实现一个移动磁头的算法。

真正的磁盘在这个地方

```
    private class RealDisk {
        Cylinder[] cylinders = new Cylinder[CYLINDER_NUM];

        public RealDisk() {
            for (int i = 0; i < CYLINDER_NUM; i++) cylinders[i] = new
Cylinder();
        }
    }
```

大家需要在类里面添加一些方法来修改扇区里面的数据。另外我们在Disk类中提供了一个校验磁盘的多项式`1100000000100001`，大家只能使用这个来校验磁盘中的数据。

对于字节流到比特流的转换来说，大家需要使用位运算来解决，这里需要注意的地方是Java的Char类型是2个字节，但是我们只使用了低八位，比如某个扇区中的第一个Byte是`00000111`，那么最后一个1可以这样取出来

```
(char) (((datum >> (0)) & (0b00000001)) + '0');
```

那么怎么把一个`1`写到一个Byte的最后一位呢？这个留给大家自己搞定。

测试

在CRC和Disk中为大家提供了测试接口

```
    /**
     * 这个方法仅用于测试，请勿修改
     * @param data
     * @param polynomial
     */
    public static void CalculateTest(char[] data, String polynomial){
        System.out.print(Calculate(data, polynomial));
    }
    /**
     * 这个方法仅用于测试，请勿修改
     * @param data
     * @param polynomial
     */
    public static void CheckTest(char[] data, String polynomial, char[]
CheckCode){
        System.out.print(Check(data, polynomial, CheckCode));
    }
```

```
    /**
     * 这个方法仅供测试，请勿修改
     * @param eip
     * @param len
     * @return
     */
    public char[] readTest(String eip, int len){
        char[] data = read(eip, len);
        System.out.print(data);
        return data;
    }
```

在测试的时候，我们会调用这几个方法。

4.代码框架

```
.
├── README.md
├── pom.xml
└── src
    └── main
```

```
└── java
    ├── cpu
    │   └── alu
    │       ├── ALU.java
    │       ├── FPU.java
    │       └── NBCDU.java
    ├── memory
    │   └── Disk.java           # you need write
    ├── transformer
    │   └── Transformer.java
    └── util
        ├── BinaryIntegers.java
        ├── CRC.java            # you need write
        └── IEEE754Float.java
```

5.未尽事宜

请写邮件给任老师