CSE Lab1 文件系统lab1

助教

- 姜尔玲 17307130291
- 王孟辉 19212010074
- 刘林方 19212010051

描述

我们经常使用计算机系统的存储功能，例如数据库系统和文件系统都发挥计算机系统的存储功能。在这个课程中，我们要实现的是一个文件系统。本次lab中需要构建一个有多数据副本（duplicate）和在文件/数据管理层面上分组（partition）的文件系统，为保证统一，我们将这个文件系统取名为SmartFileSystem。（duplicate和partition的具体内容在下面介绍）

注：大家请尽快构思SmartFileSystem的代码结构设计，关于lab中不确定的内容及时询问助教；代码结构的构思完成后，尽量和同学或者助教进行沟通，确保设计的正确性；同时在有自己思考和对问题的整理的基础上向助教提出疑问，避免浪费时间等待助教回复消息以及低效的沟通。

DDL

DDL：2020.10.25, 23:59

将所有提交内容打包为lab1-姓名-学好.zip上传至elearning

之后安排面试

SmartFileSystem的基础信息

1. SmartFileSystem分为"FileManager"和"BlockManager"两类服务。
2. SmartFileSystem用"File"来管理数据，使用"Block"来实现数据的物理存储。
3. FileManager管理File的集合，负责记录File的meta信息。
4. Block负责存储数据，Block中的数据分为BlockData和BlockMeta两类信息
    1. 前者以.data为结尾，存储文件数据
    2. 后者以.meta为结尾，存储Block的meta信息
5. Block有一定的size，单位是byte，每个Block中可以存储的内容不得大于该size。
6. Block应当设置为不可重写的，即在对文件内容进行修改时，应当新建一个Block进行写入，同时修改file的meta信息等。
7. BlockManager管理Block的集合，具备创建Block、索引和读取BlockData的能力。
8. FileManager和BlockManager支持分组，即不同的Manager只负责自己的FileMeta。
9. 读取File的data的过程是：通过File的meta信息获取File的内容在Block中的定位，通过对block的读取得到file的内容。
10. File需要支持随机读写，允许调整FileData的size；可以直接写入也可以使用Buffer。

11. File的写入要满足简单的一致性。成功的写入操作需要满足数据写入到block中且fileMeta修改成功。如果写入失败，则不能改变fileData，可以不需要保证FileMeta的不变性，也可以考虑保证。（请在面试的时候和文档中说清楚）。
12. 需要完成一个工具系列：
    1. smart-cat：获取File的File内容；能够从文件指定位置，读去指定长度的内容并且打印在控制台。
    2. smart-hex：读取block的data并用16禁止的形式打印到控制台。
    3. smart-write：将写入指针移动到指定位置后，开始读取用户数据，并且写入到文件中。
    4. smart-copy：复制File到另一个File。
13. 需要完成一个异常处理规范，处理异常是使用对应的内容，并且应该整理一个处理规范文档。

参考

BlockManager 和 FileManager 的 Partition 应该可以像下图一样：

```
BlockManager:
BM1 BM2 BM3
b1  b3  b5
b2  b4  b6
FileManager:
FM1 FM2
f1  f3
f2
下面是各个 File 使用的Block
f1:BM1.b1,BM3.b5
f2:BM2.b3,BM1.b2
f3:BM3.b6,BM2.b4
```

一次File读操作的流程：

1. 用户请求读取 FM1.f1 的数据；
2. File 先读取 BM1.b1 的数据，然后读取 BM3.b5 的数据。
3. 如果读取失败（FM不可用，BM不可用，Blk不可用，BIK校验失败等原因）则抛出异常给上层；
4. 如果读取成功，直接返回数据。

一次成功的写操作流程：

1. 用户请求写入数据到FM2.f3的第二个数据块上；
2. 随机选择一个BM,假设选择 BM1;
3. BM1 分配一个新的 Block 编号为 b7;
4. 写入数据到b7;
5. 改变FM2.f3的 FileMeta 为 BM3.b6,BM1.b7(不再引用BM2.b4了）；
6. 不应该从BM2中抹去b4的存在；

一次失败的写入操作：

1. 用户请求写入数据到FM1.f1 的第1个数据块和第2个数据块上；
2. BM3 为其分配b8作为新的第1个数据块，BM2为其分配b9作为新的第2个数据块；
3. 第一个数据块写入成功，第二个数据块写入失败；

4. 维持 FM1.f1 的 FileMeta 不变，然后抛出异常给上层表示写入失败；
5. 不需要删除新分配的b8和b9;
6. 这样就可以保证失败的写操作不会改变File,保证了简单一致性。(注：简单一致性没有定义 FileMeta 写入失败的处理过程，如果FileMeta写入失败，不需要恢复 FileMeta)

支持duplication

现实生活中，磁盘的损坏会使磁盘上的文件丢失，所以我们经常需要使用duplication来保证及时一部分磁盘损坏，也能读取到正确的文件内容。在这个lab中我们可以通过引入logic block来实现。当然如果你有别的方法也是OK的。

```
size: 200
block size: 32
logic block:
0:["bm-01",13]["bm-02",82]["bm-03",14]
1:["bm-05",31]
...
6:["bm-1",89]["bm-04",21]
```

可以看到，大小为200的文件，占据了六个logic block；且对于每个block而言，存在一定数量的duplicated block；这个“一定数量”既可以是一个固定的数字，也可以是一个在一定范围内的随机数字。

对于每一个logic block，可以首先随机选择一个block，如果block所属的manager存在且数据完整，则可以获取数据，进而一步步获取所有logic block的数据；对于数据信息错误的logic block，应该有相应的异常处理。

File的写操作

我们这里要实现两种file的写操作，其中一种是write，另一种是setSize。

- write(byte[] b)，将数组b的内容写入到file中
- setSize(int size)，将文件的大小直接进行修改
    - 如果size大于原来的file size，那么新增的字节应该全为0x00
    - 如果size小于原来的file size，注意修改file meta中对应的logic block，且被删除的数据不应该能够再次被访问
    - 主要实现方式合理即可

Block 实现建议

建议Block的data和meta分成两个文件进行存储

由于我们需要检验block是否被损坏，所以我们需要在meta中存储data内容对应的checksum，可以参考的meta存储内容格式为：/path/bm-xx/12.meta

```
size: 512
checksum: 12349192123491912921
```

校验码可以随便选择一种校验/哈希函数（例如MD5函数）

需要实现的接口

```
interface Id {
// empty interface
}

// Block write , immutatable
interface Block {
  Id getIndexId();
  BlockManager getBlockManager();
  byte[] read();
  int blockSize();
}

interface BlockManager {
  Block getBlock(Id indexId);
  Block newBlock(byte[] b);
  default Block newEmptyBlock(int blockSize) {
    return newBlock(new byte[blockSize]);
  }
}
public interface File {
  int MOVE_CURR = 0;
  int MOVE_HEAD = 1;
  int MOVE_TAIL = 2;
  Id getFileId();
  FileManager getFileManager();
  byte[] read(int length);
  void write(byte[] b);
  default long pos() {
    return move(0, MOVE_CURR);
  }
  long move(long offset, int where);

  //使用buffer的同学需要实现
  void close();
  long size();
  void setSize(long newSize);
}

public interface FileManager {
  File getFile(Id fileId);
  File newFile(Id fileId);
}
```

一点解释：

- File接口中的`MOVE_CURR`、`MOVE_HEAD`、`MOVE_TAIL`代表的是文件中光标的位置、文件开头和文

件结尾，其中光标的位置是File需要维护的一个指针，而用1和2给后两者赋值并无实际意义。
- `close`方法表示的是释放资源，在该lab中如果使用了buffer则需要释放资源，不使用buffer的情况下直接实现为空方法即可。

异常处理

我们的任务是完成一个文件系统，在文件的读取、写入等操作中，很容易引起异常：例如打开不存在的File、创建已经被创建过的文件、Block已经被损坏等。Error Code是一种很简单的异常处理方法，且大家需要整理一份异常处理的文档，解释清楚异常产生的原因即可。如下是一份参考实现。

```
public class ErrorCode extends RuntimeException {
  public static final int IO_EXCEPTION = 1;
  public static final int CHECKSUM_CHECK_FAILED = 2;
  // ... and more
  public static final int UNKNOWN = 1000;

  private static final Map<Integer, String> ErrorCodeMap = new HashMap<>();

  static {
    ErrorCodeMap.put(IO_EXCEPTION, "IO exception");
    ErrorCodeMap.put(CHECKSUM_CHECK_FAILED, "block checksum check failed");
    ErrorCodeMap.put(UNKNOWN, "unknown");
  }

  public static String getErrorText(int errorCode) {
    return ErrorCodeMap.getOrDefault(errorCode, "invalid");
  }

  private int errorCode;

  public ErrorCode(int errorCode) {
    super(String.format("error code '%d' \"%s\"", errorCode,
getErrorText(errorCode))) this.errorCode = errorCode;
  }

  public int getErrorCode() {
    return errorCode;
  }
}
```

工具参考实现

1. smart-cat：直接读取fileData
2. smart-hex：读取block的data并用16禁止的形式打印到控制台
3. smart-write：将写入指针移动到指定位置后，开始读取用户数据，并且写入到文件中
4. smart-copy：
    1. 读取已有的file的fileData，写入到新File中

2. 直接复制File的FileMeta，这个方法的正确性以来于Block是不可重写的，建议使用这个方法实现

bonus举例

- buffer实现

bonus是5-10分（额外分数），加上bonus之后总分不超过100分；除了基本的要求，其余是否算做bonus视情况而定；在面试之前整理明确好自己的思路。

评分标准根据本文档内容设定，涉及file、block的各项操作。

加油