CSE Lab 1

Assigned: Oct. 2

Due: Tue., Oct. 16, 23:59

TAs: 袁莉萍 (15302010051@fudan.edu.cn), 欧承祖 (oucz14@fudan.edu.cn)

Introduction

In this course, we will get to build a simple distributed file system. Though simple as it may seems to be, our Simple Distributed File System(SDFS) does cover some of the main design principles of computer system. By following the instructions of each lab and actually implementing the system, you will gain a much deeper understanding of the principles behind.

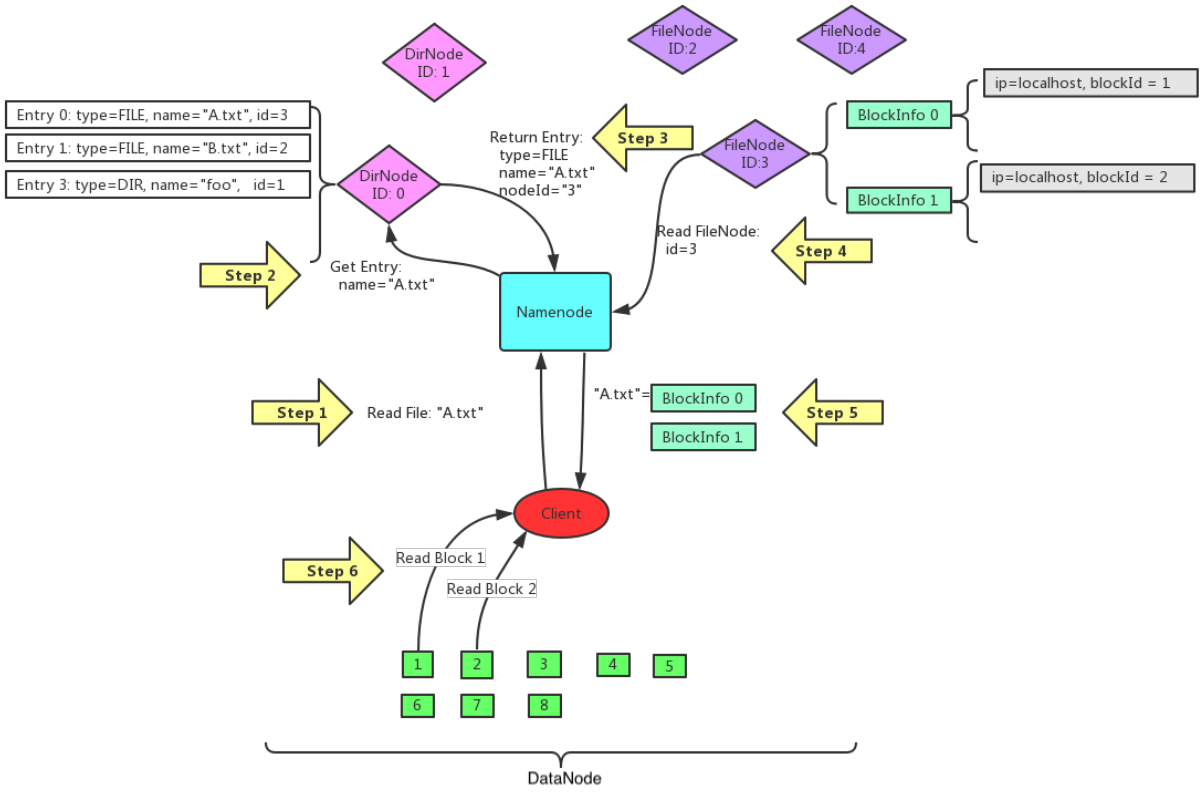
In this lab, you are asked to implement the basic function of a file system that runs on a single machine.

Logistics

This lab should be done by yourself alone. Any plagiarism or copy of another's code or words without proper attribution or credit is strictly prohibited.

Architecture

The full design document of SDFS is provided along with the lab. You should read it top-to-bottom before proceeding to the requirements of this lab. The design of SDFS may seem a little bit complicated to you. But fortunately, you do not need to implement all of them in this lab. In fact, the file system you are about to build today is not even distributed! However, for the purpose of forward compatibility, you should divide the system into three parts just like SDFS does.



Client

The client of the file system should have three functions:

1. `open` a file in SDFS, return a `SDFSFileChannel`.
2. `create` a file in SDFS, return a `SDFSFileChannel`.
3. `mkdir`, create a directory in SDFS.

That `SDFSFileChannel` is the file channel to the file opened or created. It should implement `SeekableByteChannel` interface. Note that there are two kinds of `open` in SDFS. One is for read-only and another is for read-write. One file could be opened as read-only simultaneously as many times as possible but no more than one read-write file channel could be opened in the same time.

NameNode

The NameNode is responsible for the metadata of the file system. Basically, it manages the entire file tree and the block information of each file. You should implement five functions:

1. `open` a file, return a `SDFSFileChannel`.
2. `create` a file, return a `SDFSFileChannel`. Just as stated in SDFS Design Document, the file tree in SDFS is composed of `FileNode` and `DirNode` and `FileNode` contains `BlockInfo`s. You should consider the process of the creation of a file carefully.

3. `mkdir`, create a directory. Directory is represented as `DirNode` in SDFS and each `DirNode` has entries in it.
4. `close` an opened file. Changes of the metadata will persist or reveal only after the file is properly closed.
5. `addBlocks`, add new blocks to a file, return the information of the newly created blocks. Since there is only one DataNode in this lab, block information contains only the number(ID) of the block.
6. `removeLastBlocks`, remove the last blocks of the file. It is used when truncating a file to free the last blocks for future use.

DataNode

The DataNode should handle `read` and `write` request from clients:

1. `read` data from the specified block.
2. `write` data to the specified block.

Persistence

In NameNode, the file tree should be persist on the disk whenever a change to it happens. Each node should be stored in a separate file with the name `[Node_ID].node`. Root node should always be `0.node`. You should design a format for `FileNode` and `DirNode` and implement your own serializing and deserializing function by yourself. For example, for `DirNode`, you just need to save the entries and for each entry, the first byte is the type of the node (0 for `file` and 1 for `dir`) and then the following three bytes records the length of the filename in bytes, then follows the filename and the last four bytes records the ID of the node. Simply using default (de)serializing function in java is not allowed. All the files could be stored under the same directory.

In DataNode, each block should be stored in a separate file with the name `[BLOCK_NUMBER].block`. All the files could be stored under the same directory.

Communication

Since we are building a pseudo-distributed file system, you can safely assume that both NameNode and DataNode are singletons and you can simply get the instance and directly call the methods of the NameNode and DataNode from the client.

Implementation

We provide the skeleton code and unit tests. Your implementation should pass all the tests. You are allowed to modify the structure of the project and even some of the tests based on your understanding of the system. But be sure to explain it in detail in the document. Or it will be graded based on provided tests. A hint, read the comments in the provided skeleton code carefully.

Documentation

We believe that a genius implementation is useless without a rich documentation. Your document should include the following content:

- The architecture of the system
- Similarities and differences between your file system and UNIX file system described in the text book
- How you solve the only-one-writer problem
- How you solve the changes-apply-after-close problem
- Format of the persistent file tree in your design
- Extra work if any, for bonus
- Difficulties you met if any
- Changes to the provided tests if any

Handin

Please pack all the code, document, and possible supplementing materials in a `zip` file with name `[STUDENT_ID].zip` and upload it to the `WORK_UPLOAD/lab1/` folder on FTP server.

Grading

Document: 45%. A clear and detailed document is expected. Both Chinese and English will do. Implementation: 45%. Any failed test will cause a loss of 20%. Code Style: 10%. Well-commented and maintainable code is expected. Bonus: 20%. Hard to get all of them but easy to take a sip. Be sure to mention extra work in the document.