Lab #1: Basic Services

Schedule: **Week #1 – Week #2**

Deadline: **Oct 7, 2025**

Points: **5/40**

Prerequisite

- gRPC
- Concurrency: threading and locking mechanisms (related to implementation technology you have selected – Java, Python, .NET, C++, Go)

Goals

- Create basic implementation of all services.

Task #1.1: Basic Lock Service Implementation

- Create Lock Service server application implementing gRPC interface defined in `LockService` protobuf.
- Application should start the server and listen on given port. Port is passed as input parameter of the application.
- Method `acquire()` should block the multiple calls from the clients for each specific `lockId` string. It means when one client acquires the lock *A* and the *A* is not locked by any other clients, the method should record the lock and return `true`. If the *A* is already locked by any other client, the method is blocked and continues only after the lock *A* is released. (Note: Return value `false` is not used in this basic implementation. Parameters `ownerId` and `sequence` are also not used in this basic implementation.)
- Method `release()` should release the lock and unblock all waiting `acquire()` calls for the same lock. (Note: Parameter `ownerId` is not used in this basic implementation.)
- Method `stop()` should stop the running server.

Task #1.2: Basic Extent Service Implementation

- Create Extent Service server application implementing gRPC interface defined in `ExtentService` protobuf.
- Application should start the server and listen on given port and store extent on given extent root path. Port and extent root path are passed as inut parameters of the application. The extent root path is a directory of the in real file system of the server. (Example: Let extent root path be `/home/dfs-extent-root`. The file `/dir1/file1` in our DFS is stored in real file system of the server in `/home/dfs-`

`extent-root/dir1/file1` file.)

- Method `get()` reads the content of the file/directory from the disk and returns it as an array of bytes. Distinction between file and directory is the file name postfix (directory name ends with character */*). Content of the directory could be any representation of the list of names of included files and directory names should be postfixed by character */*. Parameter `fileName` is always full path in our DFS (not path in real file system of the server). When the reading of the file/directory ends with an error, return `null` (in protobuf structure value `null` does not exist; return `null` means the optional parameter in response message is missing).
- Method `put()` writes or deletes the file/directory on the disk. Distinction between file and directory is similar as above. To put directory means to create new directory. If the parameter `fileData` is `null` (in protobuf structure value `null` does not exist; parameter is `null` means the optional parameter in request message is missing) it means either file or directory should be deleted (only empty directory can be deleted). Parameter `fileName` is always full path in our DFS (not path in real file system of the server). When writing/deleting of the file or creating/deleting directory is successful, return `true`, otherwise return `false`.
- Method `stop()` should stop the running server.

Task #1.3: Basic DFS Service Implementation

- Create DFS Service server application implementing gRPC interface defined in `DfsService` protobuf.
- Application should start the server and listen on given port and connect to Extent Service server and Lock Service server. Port and addresses of Extent Service server and Lock Service server are passed as input parmeters of the application. Address should use `hostname:port` or `IP:port` format.
- Method `dir()` uses remote object of Extent Service instance to get the content of the directory. The directory name should be the full path in DFS (not in the real file system of extent server) and ends with character */*. The content should be returned as an array/list of strings. If the directory is empty the empty list is returned. If there is an error to get the content of the directory, return `null` (see above about `null` value in protobuf).
- Method `mkdir()` uses remote object of Extent Service instance to create new directory. The directory name should be the full path in DFS (not in the real file system of extent server) and ends with character */*. The content of the directory could be anything except `null` (see above about `null` value in protobuf). If the directory creation is successful, return `true`, otherwise return `false`.
- Method `rmdir()` uses remote object of Extent Service instance to delete the empty directory. The directory name should be the full path in DFS (not in the real file system of extent server) and ends with character */*. The content of the directory must be `null` (see above about `null` value in protobuf). If the directory deletion is successful, return `true`, otherwise return `false`.
- Method `get()` uses remote object of Extent Service instance to get the content of

the file. The file name should be the full path in DFS (not in the real file system of extent server) and does not end with character /. If the file read is successful, return content as an array of bytes, otherwise return `null` (see above about `null` value in protobuf).

- Method `put()` uses remote object of Extent Service instance to write/create the file. The file name should be the full path in DFS (not in the real file system of extent server) and does not end with character /. The content of the file cannot be `null` (see above about `null` value in protobuf). If the file write/creation is successful, return `true`, otherwise return `false`.
- Method `delete()` uses remote object of Extent Service instance to delete the file. The file name should be the full path in DFS (not in the real file system of extent server) and does not end with character /. The content of the file must be `null` (see above about `null` value in protobuf). If the file deletion is successful, return `true`, otherwise return `false`.
- Method `stop()` should disconnect from Extent Service server and Lock Service server and stop the running server.

What to lock?

You must choose what the locks refer to. At one extreme, you could have a single lock for the whole file system, so that operations never proceed in parallel. At the other extreme you could lock each entry in a directory. Neither of these is a good idea! A single global lock prevents concurrency that would have been fine, for example `mkdir()`/`put()` operations in different directories. Fine-grained locks have high overhead and make deadlock likely, since you often need to hold more than one fine-grained lock.

Your best bet is to associate one lock with each file handle. Use the file or directory's full path in our DFS as the name of the lock (i.e. pass the path to `acquire()` and `release()` calls). The convention should be that any DFS Service operation should acquire the lock on the file or directory it uses, perform the operation, finish updating the Extent Service (if the operation has side-effects), and then release the lock on the path name. You must be careful about releasing the locks in all circumstances upon return from DFS Service operation. You may find adding these `acquire()` and `release()` calls a bit tedious. It is OK to do things the tedious way, but feel free to look for ways to make things a bit easier on you.

Notes

- When running services using shell scripts, start each service and client in a separate terminal.
- When running tests, all your services and clients must be stopped.