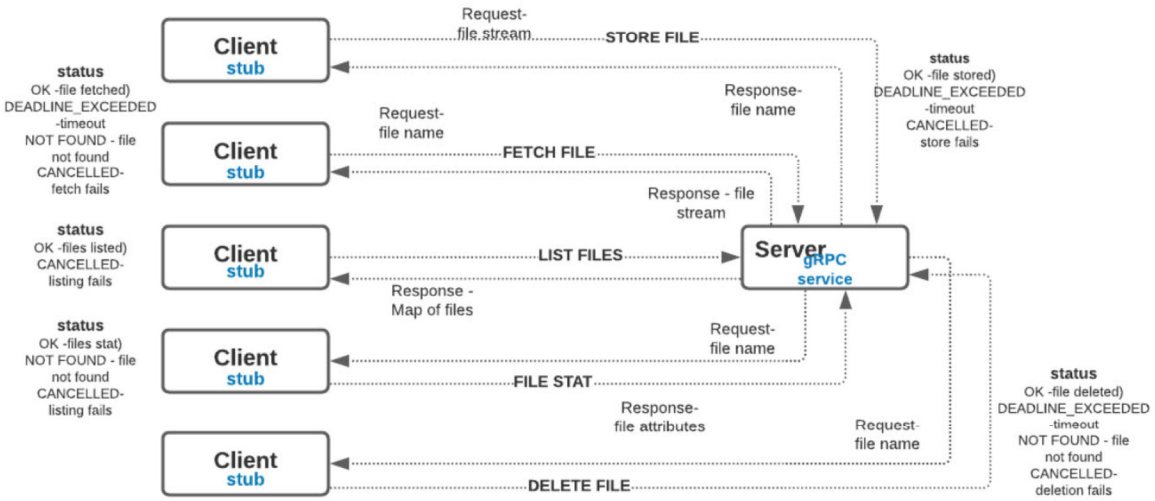
README

Introduction

This is GIOS Project III Spring 2024 Readme file from course CS6200. It has two sections.

1. RPC Protocol Service
2. Distributed File System

Part I: RPC Protocol Service



In this part of the project, we create RPC calls using the gRPC framework to store, fetch, list, and get attributes of a file in a remote server. Each RPC method is declared in a proto file with data structured as message types. The gRPC framework generates the necessary services for the client to communicate with the remote server. The server implements the RPC service methods, while the client creates a stub with the same methods. To avoid service latency and resource exhaustion, each RPC call has a deadline timeout.

For fetching a file, the RPC method is defined with string request and streaming response message types. The file name is set to the request object, and the fetchFile service method is invoked using the service stub. The ServerWriter API writes the file data from the server to the response,

and the ClientReader API reads the data. The server returns appropriate status codes for various scenarios.

To store a file, the storeFile service method is invoked with the context and response objects, returning the ClientWriter. The ClientWriter API streams the file content from the client to the server, and the ServerReader API reads the content and saves it.

To delete a file, the file name is passed to the service method via the stub. The gRPC service checks for file existence and removes the file, returning appropriate status codes.

To list files, the listFiles RPC method is defined with request and response message types containing a repeated field called Items. The server adds the attributes of each file to the response, which the client iterates to get each file attribute and insert into a map.

To get file attributes, the gRPC service checks for file availability and performs a stat system call to get the file properties, setting the modified and creation times to the response. Appropriate status codes are returned for various scenarios.

Testing:

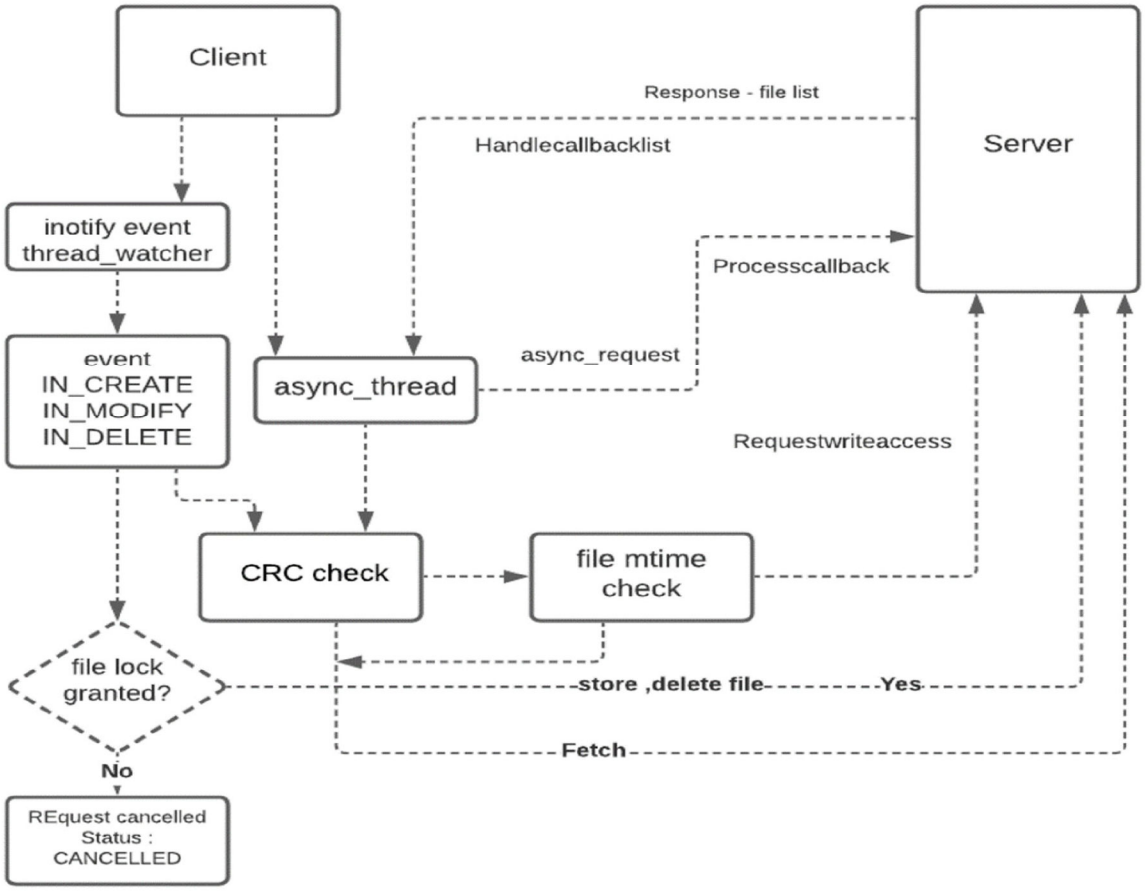
```
#!/bin/bash

bin/dfs-server-p1 -d 3 -m mnt/server/sample-files
```

```
#!/bin/bash
set -e
TXT_FILE_NAME='parin.txt'
BIN_FILE_NAME='gt-klaus.jpg'

for i in "list" "store" "fetch" "stat" "delete" ; do
if [[ $i == "list" ]] then
    bin/dfs-client-p1 -d 3 -m /mnt/client list
elif [[ $i == "stat" ]] then
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME}
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${TXT_FILE_NAME}
elif [[ $i == "delete" ]] then
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} && ls -la
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${TXT_FILE_NAME} && ls -la
else
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} && \
        diff -s mnt/server/${BIN_FILE_NAME} /mnt/client/${BIN_FILE_NAME}
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} && \
        diff -s mnt/server/${TXT_FILE_NAME} /mnt/client/${TXT_FILE_NAME}
fi
done
```

Part II: RPC Protocol Service



In this part of the project, a distributed file system was implemented where a client connects to a remote server via gRPC framework. The client sends asynchronous requests to the server and updates its local file system accordingly. A file watcher thread monitors local file changes and updates the server.

Each asynchronous request to the server returns a list of files and their attributes. If a file is unavailable in the client mount path, it is fetched from the server. Otherwise, the modified time is checked, and the file is either stored from client to server or retrieved from server to client, depending on which version is more recent.

The file watcher thread uses inotify to monitor file creation, modification, and deletion. When a file is created (IN_CREATE), modified (IN_MODIFY),

or deleted (IN_DELETE), the corresponding RPC call is made to store or delete the file on the server.

Before making the RPC for file storage or deletion, the client requests write access from the server, which maintains a map of filename and client ID, allowing one creator/writer per file. If the mapping is successful, the server returns OK status. If the file is mapped to another client, RESOURCE_EXHAUSTED status is returned, and if a deadline timeout occurs, DEADLINE_EXCEEDED status is returned. After a successful RPC, the client stores or deletes the file, and the lock is revoked and assigned to a new client.

To prevent race conditions between the async thread and the file watcher thread, mutex locks are used during the inotify callback and when iterating through the list of files from the server. Similarly, the filename to client ID map on the server is protected with mutex locks during insertion and erasure operations.

Testing:

```
#!/bin/bash

bin/dfs-server-p1 -d 3 -m mnt/server/sample-files
```

```
#!/bin/bash
set -e
TXT_FILE_NAME='parin.txt'
BIN_FILE_NAME='gt-klaus_LARGE.jpg'

for i in "list" "store" "fetch" "stat" "delete" ; do
if [[ $i == "list" ]] then
    bin/dfs-client-p1 -d 3 -m /mnt/client list
elif [[ $i == "stat" ]] then
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME}
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${TXT_FILE_NAME}
elif [[ $i == "delete" ]] then
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} && ls -la
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${TXT_FILE_NAME} && ls -la
else
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} && \
        diff -s mnt/server/${BIN_FILE_NAME} /mnt/client/${BIN_FILE_NAME}
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} && \
        diff -s mnt/server/${TXT_FILE_NAME} /mnt/client/${TXT_FILE_NAME}
fi
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME} &
    bin/dfs-client-p1 -d 3 -m /mnt/client $i ${BIN_FILE_NAME}  || echo "FILE IS LOCKED!!!!"
done
```

References:

"C++ - Sending Image (JPEG) through Socket in C Linux." https://stackoverflow.com/questions/15445207/sending-image-jpeg-through-socket-in-c-linux.

[gRPC C++ Reference]

https://grpc.github.io/grpc/cpp/index.html

[C++14 cppreference]

https://en.cppreference.com/w/cpp/14

[CPlusPlus](https://www.cplusplus.com/

"POSIX Threads Programming."

https://hpc-tutorials.llnl.gov/posix/#PassingArguments

[Protocol Buffers 3 Language Guide]

https://developers.google.com/protocol-buffers/docs/proto3

Referred the below Github file to understand the process control flow

https://github.com/xericyang97/GIOS/tree/main/Project4

Photo from the below Github file control flow

https://github.com/ JianchengGuo/GIOS6200/ /tree/master/

[gRPC C++ Examples]

https://github.com/grpc/grpc/tree/master/examples/cpp

GeeksforGeeks. 2020. *Flexible Array Members In A Structure In C - Geeksforgeeks*.