PROJECT 4 – Remote Procedure Calls

Part 1 – Building the RPC protocol Service

The files that I submitted for this part are as follows.

dfs-service.proto

- This is the file in which the functions and the message types used in the functions was declared.
- As per the project requirement, I added the functions for store, fetch, delete, list and stat.

- As described in the readme, store is the function in which we save the file in server. This is sent from the client. So, as a result, for the function store I have passed stream of data. This data will be saved and the updated filedetails will be returned. This filedetails will consist of very basic information like the filename and the modified time.

- The next function is fetch, which is the opposite of store function. In this a file is retuned from the server to the client. So, I made the return type of this function as stream and the function will take in the parameter filename.

- The next function is delete. This is function will delete the file from the server. Since we have to know the filename, I passed filename as the function parameter. As mentioned in the readme, after deletion, we just have to return the details of the file like the name and modified time. Hence, the return type will be FileDetails.

- The next function list will list all the files present in the directory along with its name and modified time. To do this I used the standard system library function stat. Since we are not finding status of any particular file, I haven't passed any argument to the function. The return type will be ListOfFiles which in turn will hold repeated Files that will contain details like the file name and modified time.

- The last method is the GetFileStatus which will return the status information of a particular file. Since we are dealing with a single file, I added filename as the function parameter. The return value is the FileStatus which includes information asked in the readme. The values include the filename, creation time, modified time and file size.

Dfslib-clientnode-p1.cpp

This is the client-side implementation of the above-mentioned functions.

Store

- As this function stores a file in server, we need to pass the data that needs to be stored from the client. I am sending the data in chunks.

- First, I am setting the deadline_timeout based on the timeout value provided in the headerfile. This is used in the server to check if the response was not returned before the timeout.

- Next, we also need a way to tell the server in what name the file should be saved. To do this, I am adding the filename as one of the client metadata. The "filename" is the key and the value will be the filename. The server can extract this metadata and save the file in that name.

- Next step is to call the service stub function store. I used the route guide program as reference while calling this function. Since the input to the function will be stream, I am using the ClientWriter.

- Next step is to open the file in the client and start sending the contents. Since I am sending the data in chunks, I declared a chunksize that can be used in both in client and server. I set this value to 3000.

- The basic idea is to transfer the data from file to a buffer which will be chunksize. Once the data has been read into the buffer, I am calling the ClientWriter's write function to write the buffer contents. This process is repeated until the entire file contents has been written.

- If there is any error during the process, I am returning the StatusCode CANCELLED.

Fetch

- In this function, the client requests for a file from the server. Once returned, client reads the file hence I am using the ClIentReader for this function.

- Similar to the store function I am first setting the timeout value.

- The next step is to call the service_Stub function fetch and this will return the response which will be the actual file contents of the requested file sent from the server.

- Next step is to just read the contents received. To do this, I am using the ClIentReader's Read function.

- Since the server sends data in chunks, I am reading it in chunks as well.

- First, I am reading a chunk data, opening the file in client and writing the data into the file. I am doing this until the Read function is returning data.

Delete

- First, similar to the store function I am first setting the timeout value.
- This function deletes the file in the server. So, the client-side function will not have implementation of the actual file delete. The logic will be in the server. Hence here, I will just call the service stub delete function and pass the file name.

List

- First, similar to the store function I am first setting the timeout value.
- This function will list the details of all files in the server.
- Since the readme states that we have to fill in the values in file_map, after getting the results back from the server, I am iterating over the results and using insert to put the details into the map. The key of the map will be the filename and the value will be the modified time.

Stat

- This function returns the status of the requested file in the server.
- First, similar to the store function I am first setting the timeout value.
- Then I am calling the GetFileStatus and passing the filename.
- As the readme suggests, the result should be assigned to file_status. Hence I am assigning the response to the file_status.
- The actual implementation of this function will be in the server.

Dfslib-servernode-p1.cpp

Store

- This function has to write the contents sent from the client in the server. To do this, the server has to first read the contents sent by client. Hence, I am using ServerReader for this function.

- First step is to extract the filename from the client metadata. Once we know the filename, I am opening it and writing data into it. To do this, I am using the ServerReader ‘s Read function.

- Until there is data to read, I keep reading chunk by chunk and writing it to the file. Iam opening file in ios::trunc mode so that the data previously available will be wiped down and the file will be written with the new data. This loop will exit once there is no more data returned from the Read function. I used the route guide program as a reference for this.

- Another thing to do here is to check if the timeout occurred before all the data was written. To do this, I am performing the check in the beginning of the while loop. So, this check will be performed after each chunk of data has been read.

- After the data has been written, I am setting the response. The response has to be the filename and the modified time. To get this, I am using the stat function. If the stat function returns a non-zero value, I am setting the status as NOT FOUND, else the status will be OK.

Fetch

- This function has to return the requested file back to client. Hence, I am using ServerWriter.

- The first obvious thing that has to be done is to check if the requested file is present in the server or not. To do this, I am using stat function. If the stat function returns a non-zero value, I am setting the status as NOT FOUND.

- If the file is present, I am getting the filesize. This info is needed for us to know if the transfer was successful or not.

- Next step is to open the requested file and start reading contents into the buffer. (This buffer’s size will be that of chunksize). After reading it into the buffer, I am calling ServerWriter’s Write function to write the contents to the client. This process is performed until the EOF is reached or the bytessent is equal to the size of the file.

- If there was an error while transferring, I am returning CANCELLED.

- Another thing to do here is to check if the timeout occurred before all the data was written. To do this, I am performing the check in the beginning of the while loop. So, this check will be performed after each chunk of data has been written.

Delete

- This function deletes a file from the server so the basic check that needs to be performed first if the file is present or not. To do this, I am using stat function. If the stat function returns a non-zero value, I am setting the status as NOT FOUND.

- Next thing to do here is to check if the timeout occurred before the delete operation was performed.

- Next the actual deletion is done by calling the remove function. Once the delete has been performed, the response has to be set.

- The filename and the modified time is set. The modified time is obtained from the stat function that was previously called.

List

- The list function will list all files present in the server.

- Similar to the previous functions, I am first checking if a timeout has occurred.

- Next, I am using the opendir to open the directory in the server. Then using readdir, I am getting the list of all files.

- Once I get the files, I am ignoring any files that start with ".". This means I am only listing the files in the directory.

- For every file obtained, I am getting the details using the stat function. The result is then added to the response. This step is performed for all the files present in the directory.

Stat

- Similar to the list, we first check for timeout and then call stat and pass the filename.

- Once we have the size, modified and creation time, we set these values in the response.

Testing Part 1

- To test the store function, I tried different file formats like image, text file etc.
- I also tested with small size text files and large size images.
- I tried to pass a filename that was not present in the client file directory to see if the NOT_FOUND status was returned.

- To test the fetch function, I tried different file formats like image, text file etc.
- I also tested with small size text files and large size images.
- I tried to pass a filename that was not present in the server file directory to see if the NOT_FOUND status was returned.

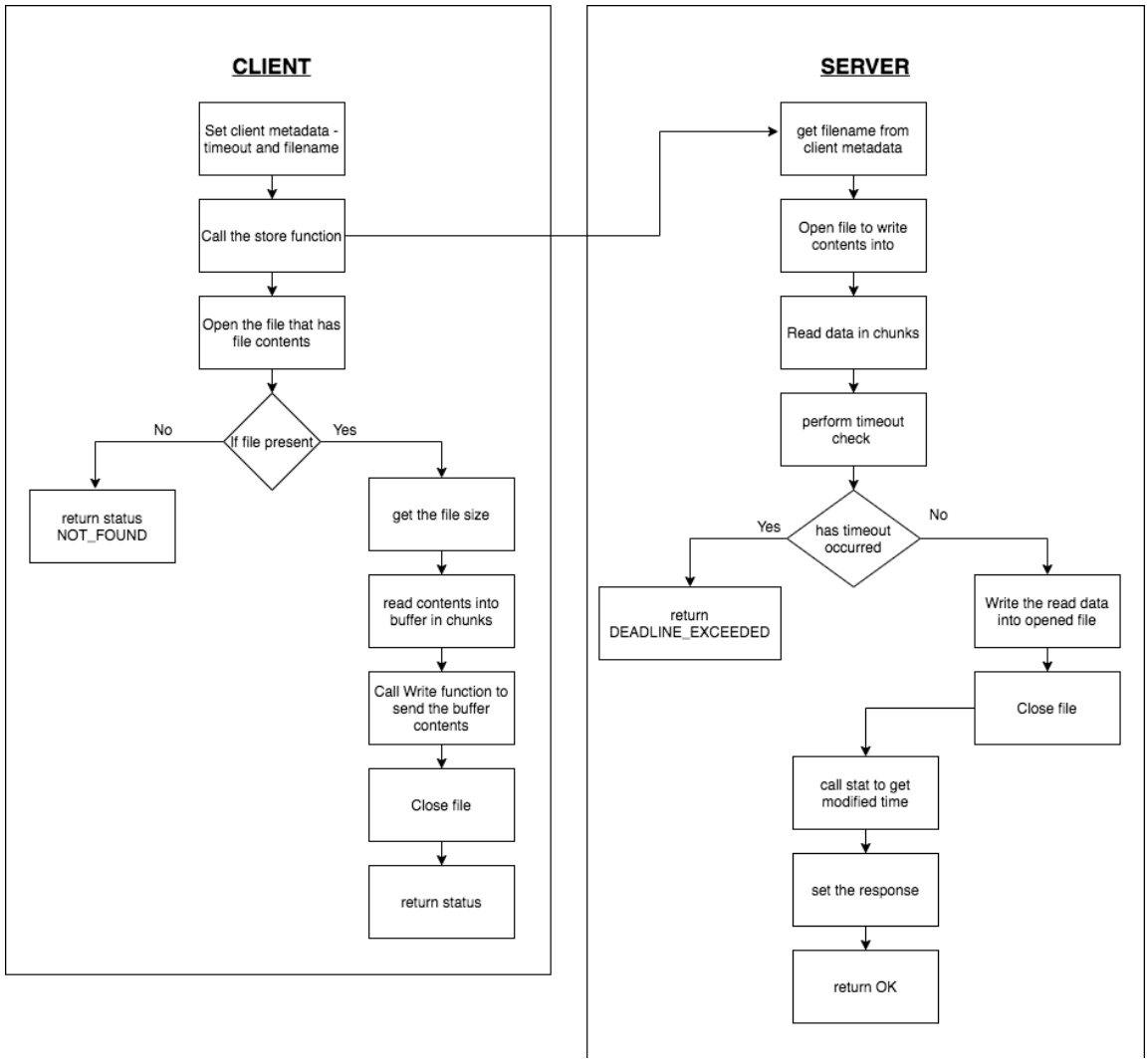
- To test delete, I passed filename which was not present in the server.
- I also tried to delete files with different formats and sizes ranging from small to large.

- To test list, I checked if the function is considering only files and ignoring files that start with ".".
- I tried to modify a file and see if calling list again displayed the new modified time.

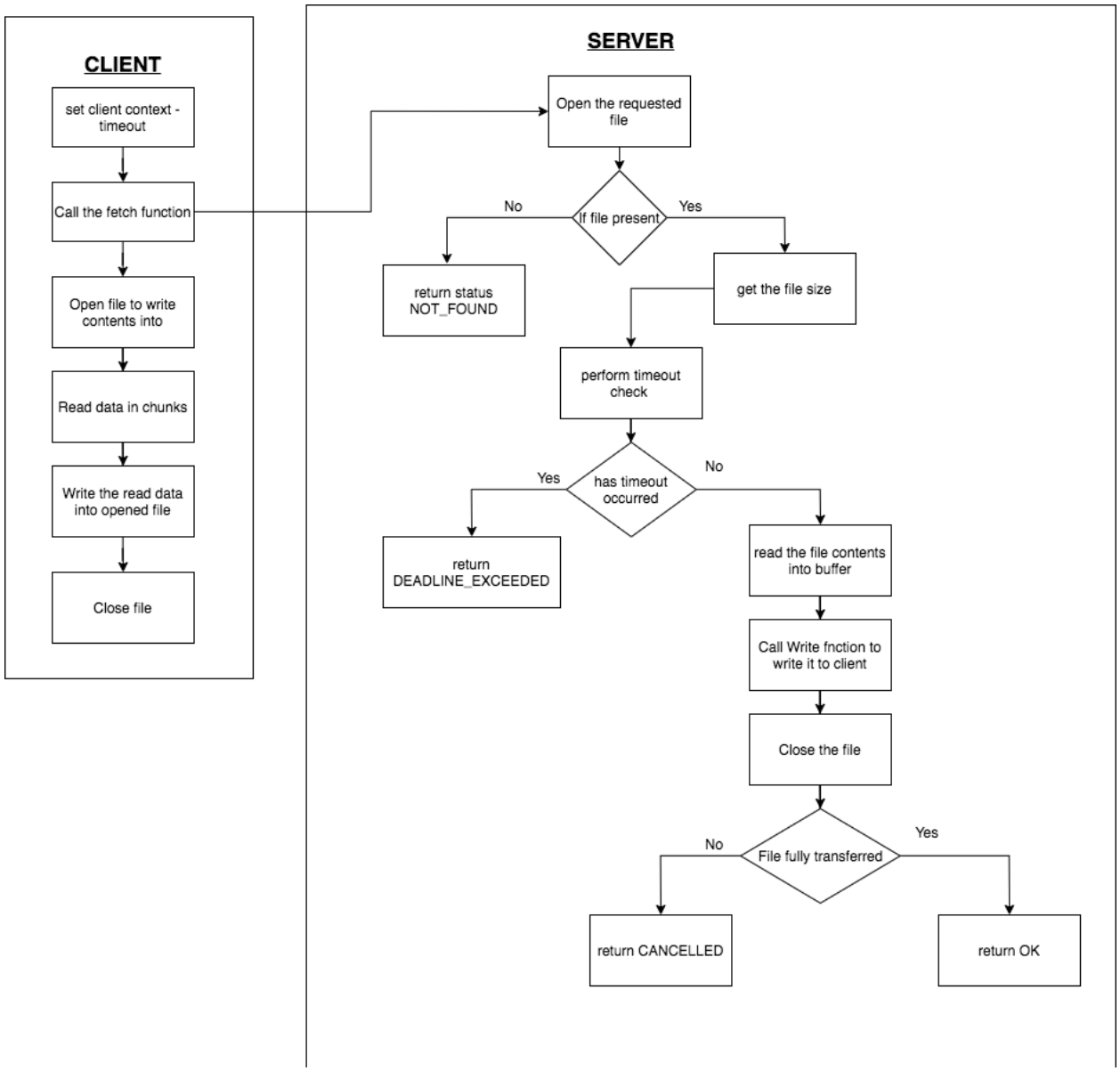
- To test stat, I tried to pass a filename that was not present in the server file directory to see if the NOT_FOUND status was returned.
  I tried to get status of files of different formats and sizes.

Program Flow - Part 1

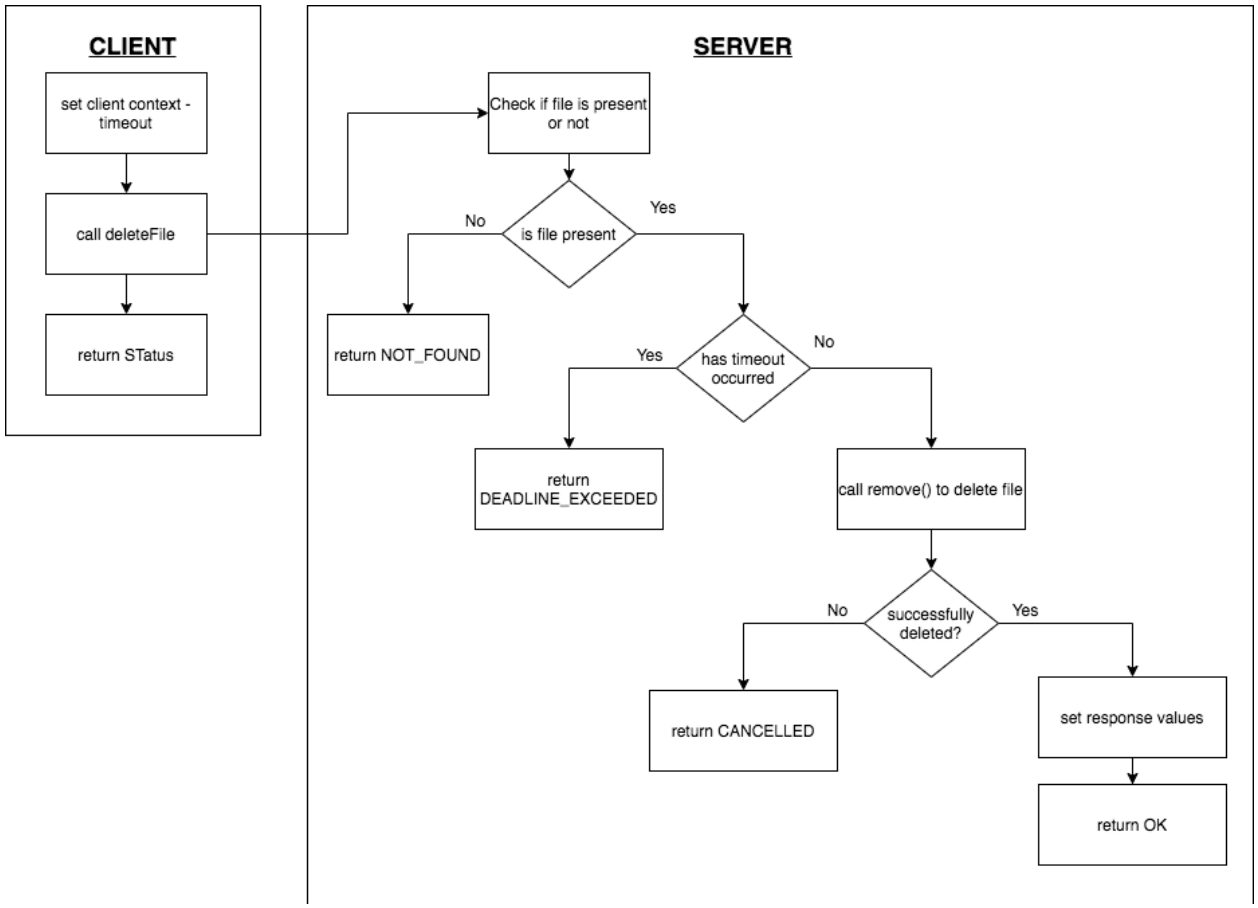
Store



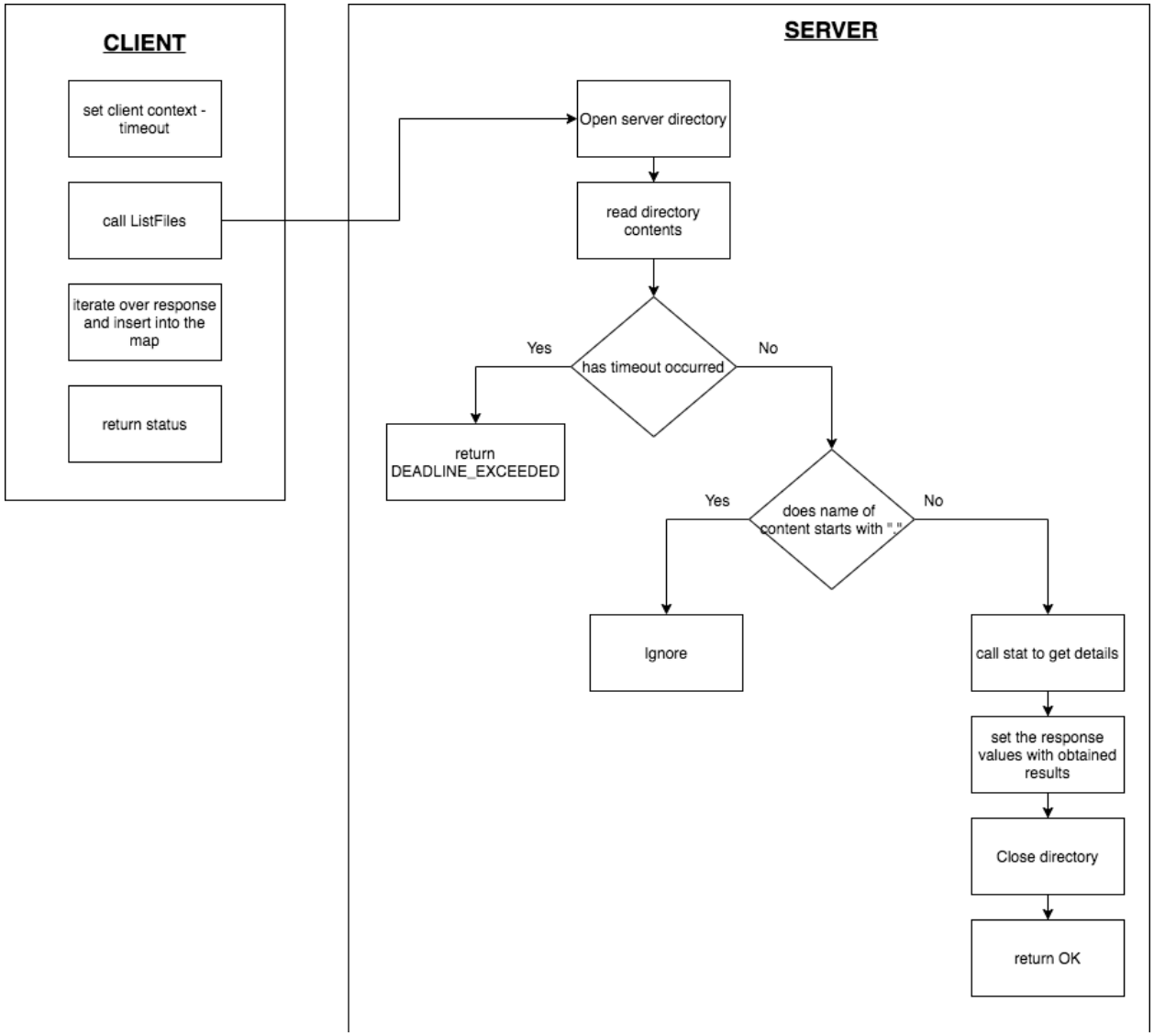
Fetch


CLIENT
set client context - timeout
Call the fetch function
Open file to write contents into
Read data in chunks
Write the read data into opened file
Close file
SERVER
Open the requested file
If file present
No
Yes
return status NOT_FOUND
get the file size
perform timeout check
has timeout occurred
Yes
No
return DEADLINE_EXCEEDED
read the file contents into buffer
Call Write fnction to write it to client
Close the file
File fully transferred
No
Yes
return CANCELLED
return OK

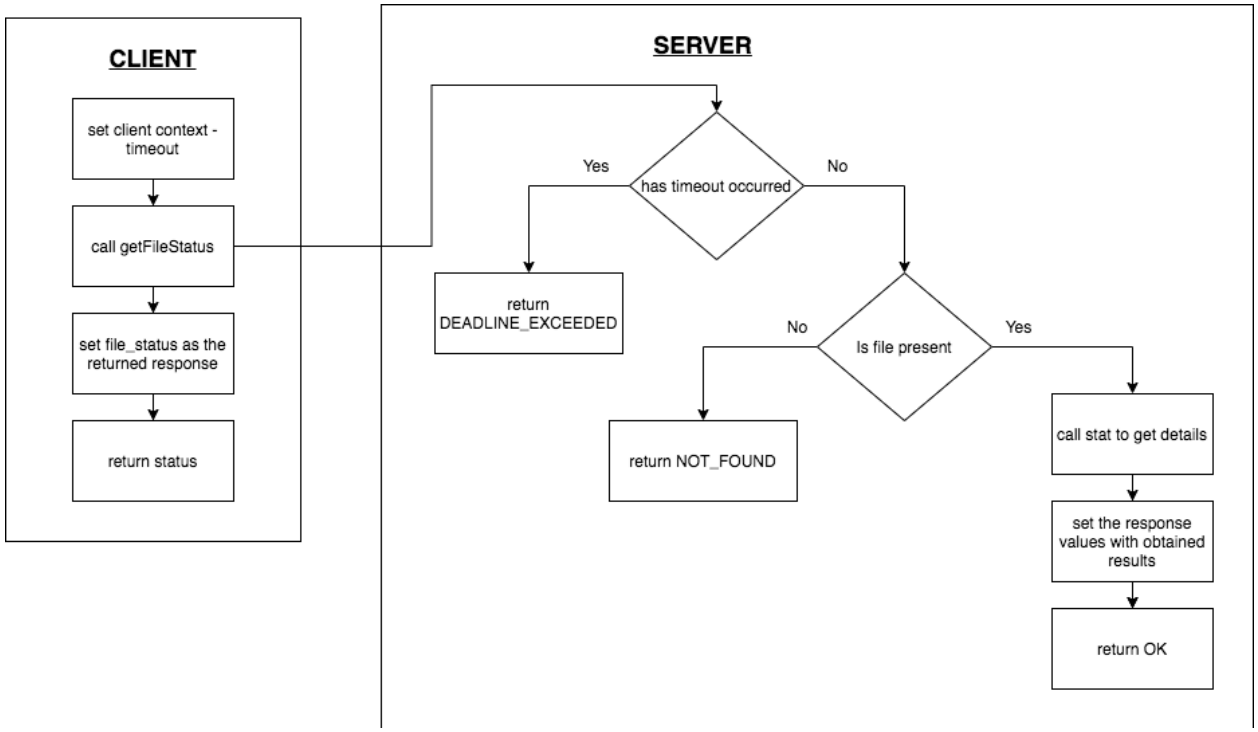
Delete


CLIENT
set client context - timeout
call deleteFile
return STatus
SERVER
Check if file is present or not
is file present
No
Yes
return NOT_FOUND
has timeout occurred
Yes
No
return DEADLINE_EXCEEDED
call remove() to delete file
successfully deleted?
No
Yes
return CANCELLED
set response values
return OK

List


CLIENT
set client context - timeout
call ListFiles
iterate over response and insert into the map
return status
SERVER
Open server directory
read directory contents
has timeout occurred
Yes
No
return DEADLINE_EXCEEDED
does name of content starts with "."
Yes
No
Ignore
call stat to get details
set the response values with obtained results
Close directory
return OK

Stat



Part 2 – Completing the Distributed File System

The files I have edited for this part of the project is mentioned below.

dfs-service.proto

- As mentioned in detail in the previous section, this file contains the function declarations and the message types. In this file I have added 2 additional functions -> callbacklist and writelock.

- As the name suggests, this writelock function is called when the client needs to obtain a lock on a file to perform certain operations.

- The callbacklist function will return list of files from the server to the client. Based on the list, the client makes changes to its files in its directory in order to sync with the server.

dfslib-clientnode-p2.cpp

- This file is almost the same as the part1. But I have made few changes to accommodate the project requirements. The changes are:
- In the new function RequestWriteAccess, I have invoked the GetWriteLock function which I defined in proto file. This function is defined in the server. To grant lock, we obviously need the ClientID. So, I have added that metadata when sending to server.

- The next function is Store. To do this, first I am requesting for the lock. Later, I am adding all metadata necessary and sending it to server and finally calling the store function. The remaining part of the function is same as explained in part 1. The new feature here is the checksum. This is used to check if the file contents in server and client are the same.

- The next function is Fetch. This does not need a lock as we are not doing anything to the server directory. Similar to above function, I am adding the metadata info and calling the fetch function. The remaining part of the function is same as explained in part 1.

- Delete function. Since this involves changes to server directory, the client should first request for a lock. To grant lock, we obviously need the ClientID. So, I have added that metadata when sending to server. Then the delete function is called.

- List function is very similar to part1. I am just calling the List function of server and setting up the response in the form of a map – file_map.

- Stat function is also same as part 1. I am calling the server's GetFileStatus.

- The next function is the InotifyWatcherCallback. This function will be called whenever there is a change in the filesystem. As suggested in the student instruction, I have called the callback() function and surrounded it with a lock to ensure proper synchronization.

- The next function is the HandleCallbackList. This function works with the list of files returned from the server. Based on the returned list, the client performs certain changes in its directory to make sure it is in sync with the server's directory.
  - First, I am iterating over all the files returned from the server.
  - If the file Is not present in the client, I am calling Fetch to get the file from the server.
  - If the file is present in the server, I am checking the modified times.
  - If the server's modified time is more recent, I am again just fetching from the server.
  - If both server and client have same file, then modifying the modified time of the file in the client.
  - Next, If the client has more recent version, then again Store is called to store this recent version in the server.
  - If a file is present in client but not in server, then I am calling remove to delete it from client. To get a list of deleted files from server, I created a list in server and added all the deleted files to it. In the ListFiles response, I am also adding this deleted file list and setting

the size parameter to 0. In the client, I am checking if size of a file is 0, if yes then I know that it was deleted from the server and hence I delete it from the client as well.

dfslib-servernode-p2.cpp

- The core parts of the functions are similar to that of part 1. The main changes that are made include acquiring the lock and checking the checksum of the server and client files.

- First function is the processCallBack. This function should return the list of files in the server so that the client can make changes to its directory. So I am calling the callbackList() here which in-turn calls the ListFiles() function defined in the server. I am also clearing the listofdeleted files as this has to be different for each async round.

- The next function I have written is to check the checksum of server and client files. This function is used to detect if the file contents are same on both ends.
  - I am first getting the checksum metadata from the client context.
  - For the server checksum, I am using the dfs_file_Checksum function as mentioned in the student's instruction.
  - Next step is to simply compare these 2 values.
  - If it is same, it means the file already exists. Hence return that status.

- The next function I have added is the GetWriteLock function.
  - The idea behind the lock is to maintain a map. This map will contain the filename as the key and the value will be the ID of the client which holds the lock.
  - So first I am getting the client ID from client which was added as metadata to client context.
  - Next, I am checking if this ClientID is present in the list or not. If not, then we know that the client doesn't have the lock. So, in this case we have to give the lock. This is just adding the entry to the map.
  - I am using a mutex lock to sync the addition and deletions to this map.
  - If the ClientID is present in the map, next I am checking if that client has lock to the file. If yes, then just returning status OK.
  - If not, then RESOURCE_EXHAUSTED is returned as the lock for the file is currently held by another client.

- If there is a function to grant lock, there needs to be a function to release the lock as well. So, the next function I added is releaseLock. All that this function does is remove the entry of the file from the lock map.

- The next function is Store.
  - First, I am getting all the metadata we passed through the client.
  - Next obvious step is to check if the client calling this store function has the lock.

- If the client has the lock, next step is to check the checksum. If the file is new, this is not required. This is required only if the file already exists in the server.
- Next if it is same, I am modifying the modified time.
- The remaining part of the function is similar to part 1 Store function.
- In the end I am calling releaseLock function to release the lock to the file so that any other client can get the lock.

- The next function is Fetch
- I have skipped the request lock part for this function as it is not altering the server directory.
- The first step is to check the checksum and do changes to modified time as mentioned earlier.
- The reminder part is same as part 1 Fetch.

- The next function is Delete.
- First step is to request for the lock. This part of the code is similar to Store's lock request.
- Once lock is present, we delete the file using the remove() function. This part of the code is similar to part 1 Delete.
- In the end, after a file has been deleted, I am adding the filename to the listOfDeletedFiles list.

- The next function is List.
- This function is same as Part 1.
- The additional code I have added is to send the listOfDeletedFiles to the client.
- I am doing this by adding the filename to the response and setting the size of this deleted file to 0.

- The last function Stat is similar to the part 1 Stat function.

Testing Part 2

- I first did the basic tests of store, fetch, delete without mounting to see if the functions work as expected. To do this, I used the test cases from part 1.

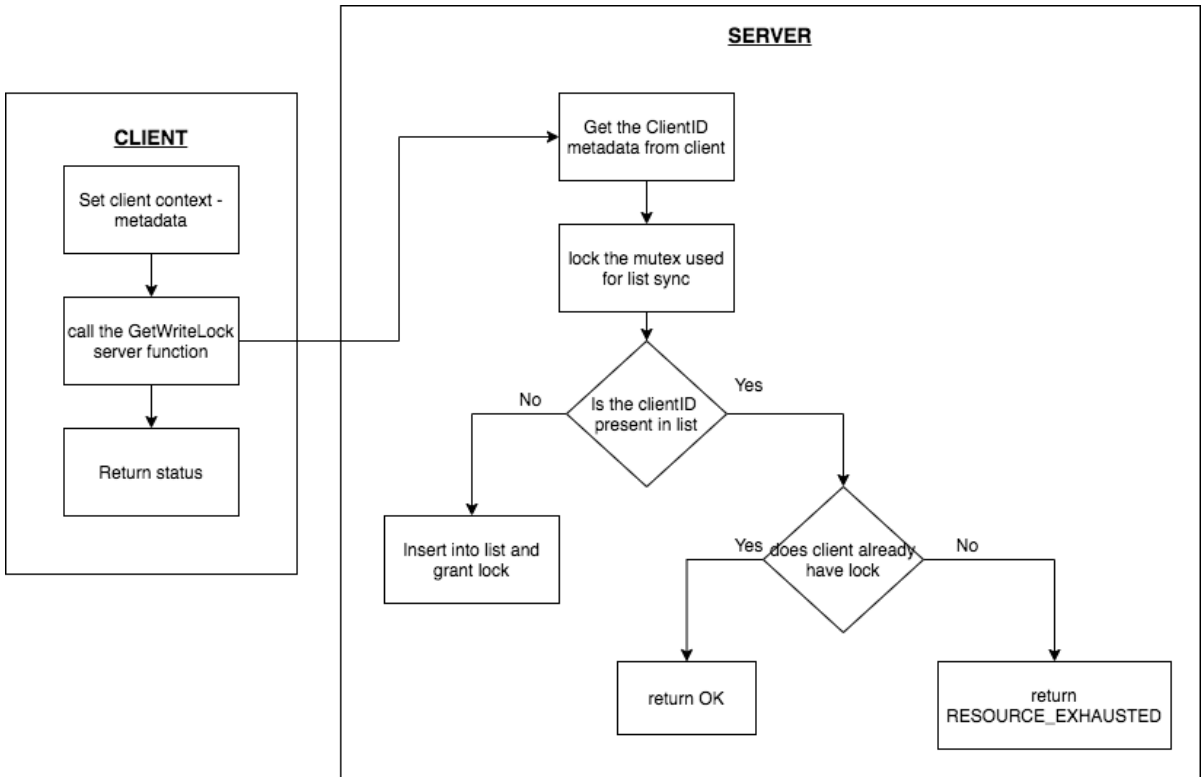
- Once all these cases were working, I was assured that my lock functionality was correctly implemented.

- Next, I checked the modified time part to check if the checksum code was correct. To do this, I had same files with same contents in both server and client but with different modified times.

- Next, I called the mount function from the client and I checked if the client and server directories where syncing.

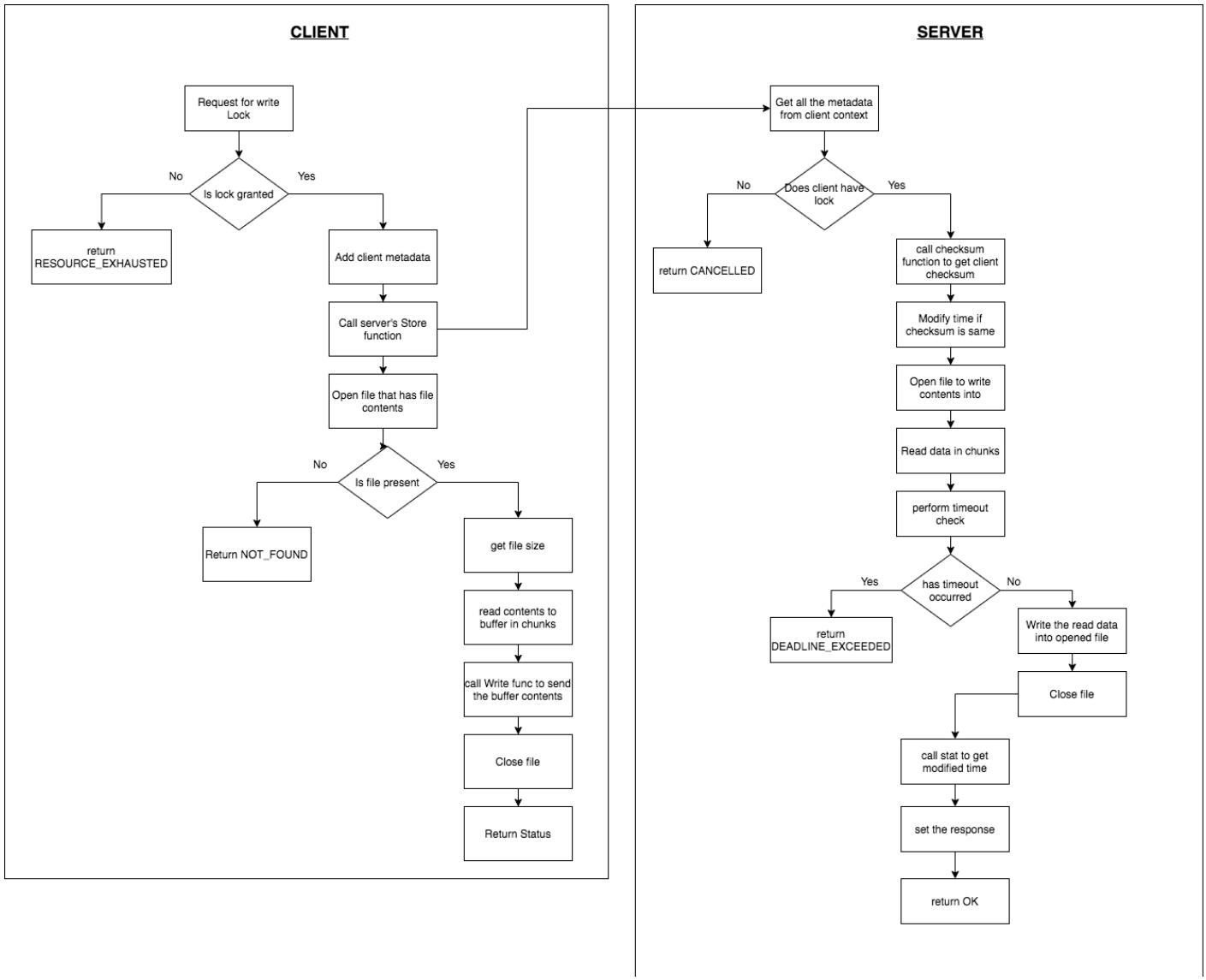
Program Flow Part 2

Getwritelock

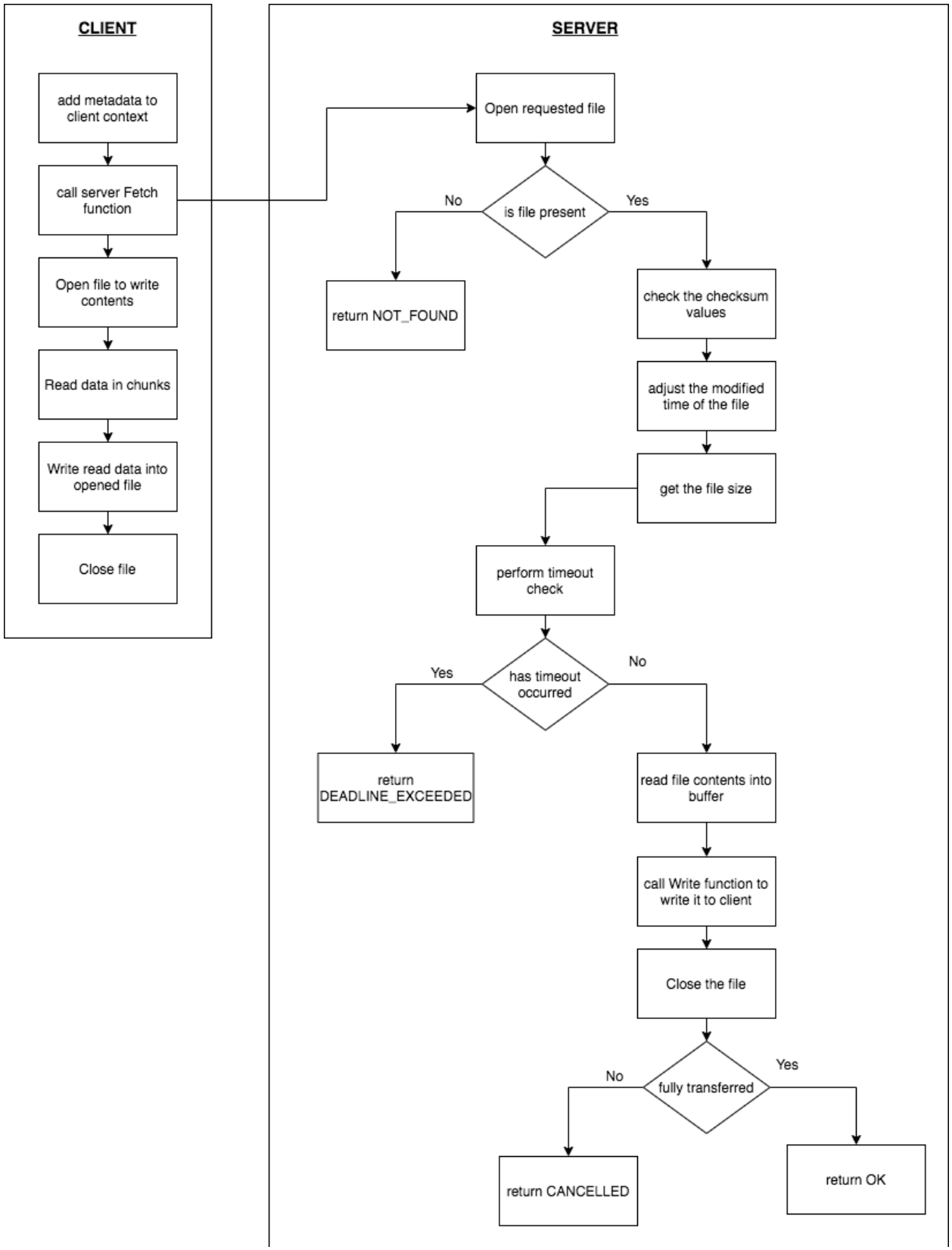


dvenkatesh7

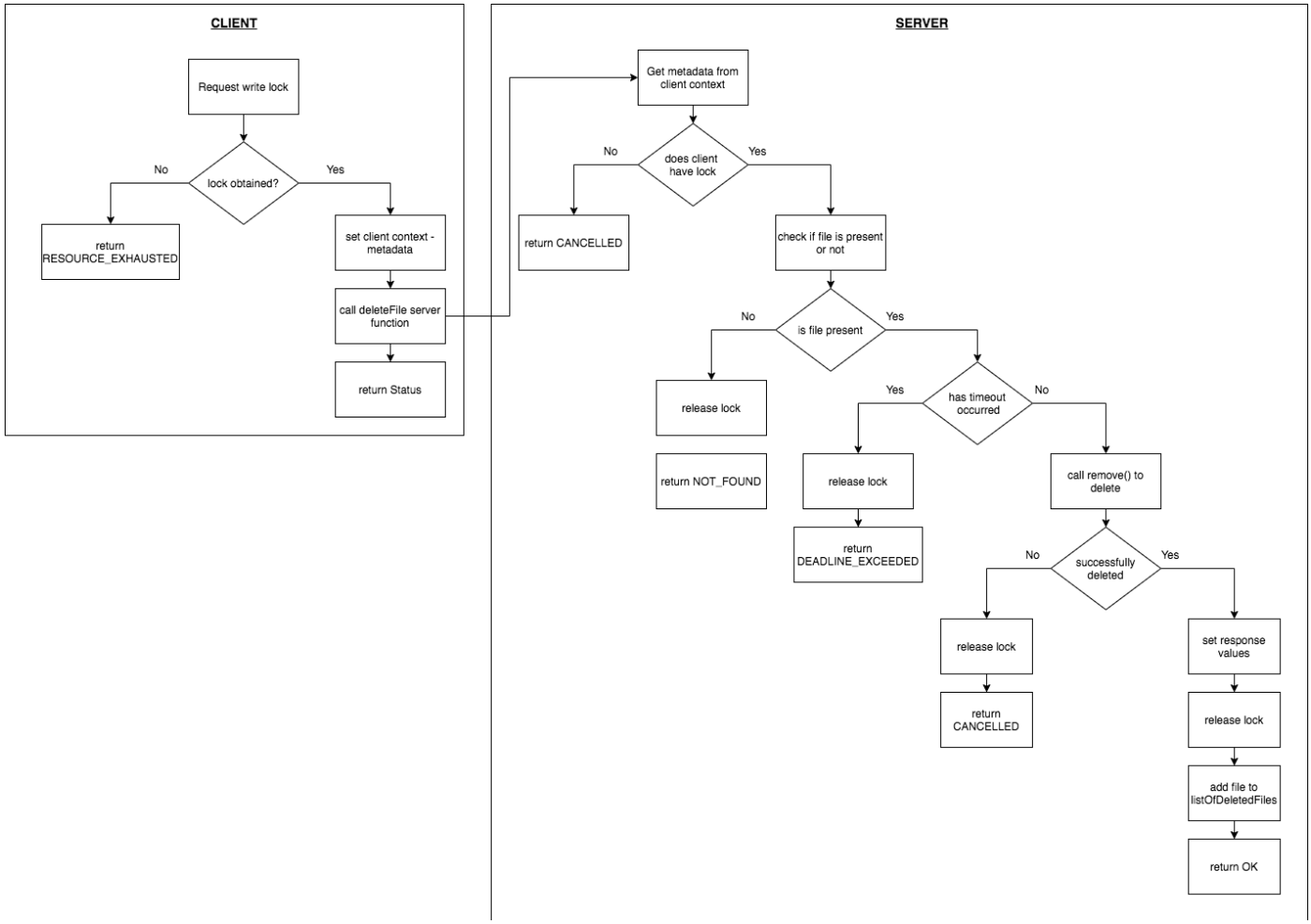
Store



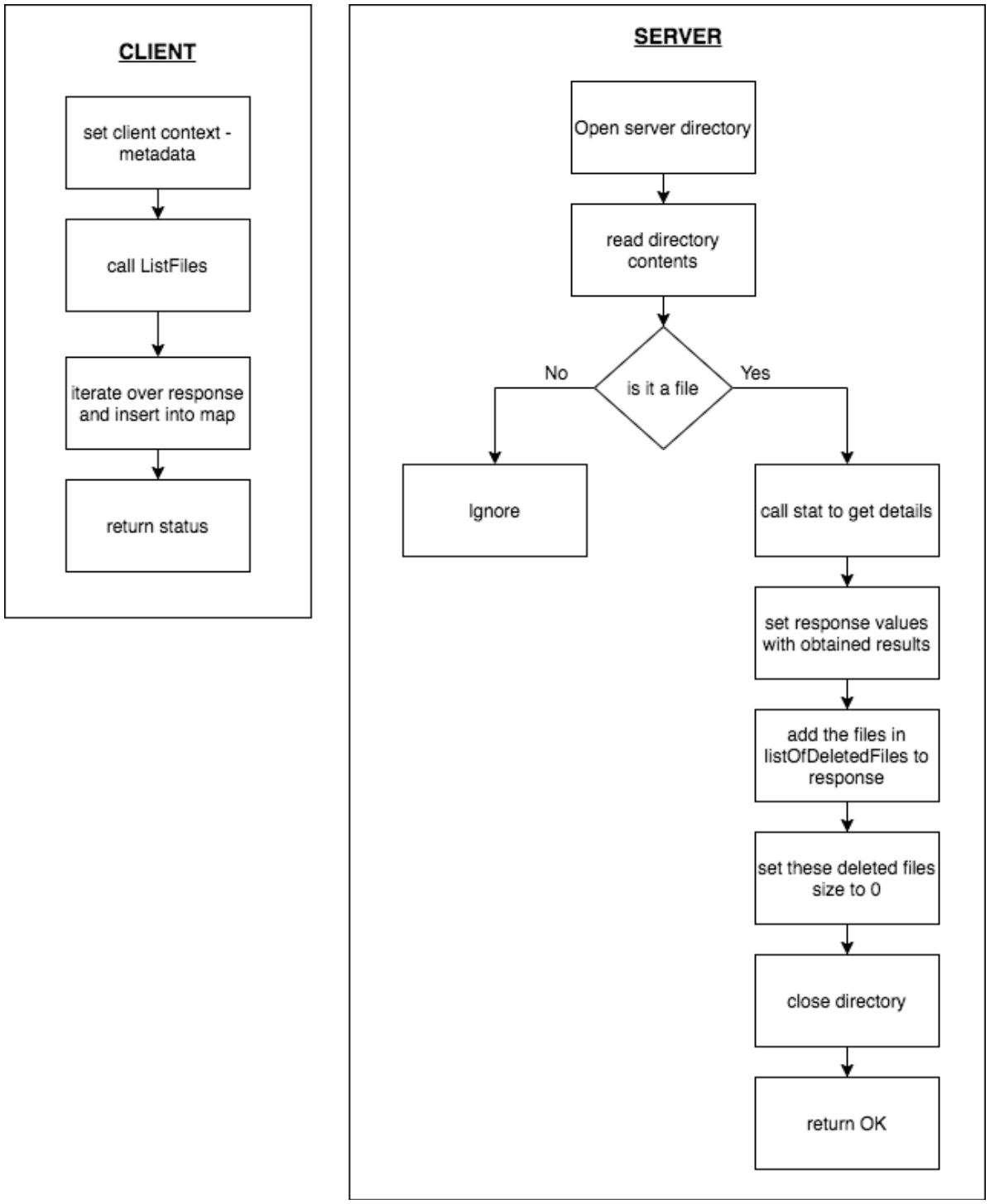
Fetch


CLIENT
add metadata to client context
call server Fetch function
Open file to write contents
Read data in chunks
Write read data into opened file
Close file
SERVER
Open requested file
is file present
No
Yes
return NOT_FOUND
check the checksum values
adjust the modified time of the file
get the file size
perform timeout check
has timeout occurred
Yes
No
return DEADLINE_EXCEEDED
read file contents into buffer
call Write function to write it to client
Close the file
fully transferred
No
Yes
return CANCELLED
return OK

Delete


CLIENT
Request write lock
lock obtained?
No
Yes
return RESOURCE_EXHAUSTED
set client context - metadata
call deleteFile server function
return Status
SERVER
Get metadata from client context
does client have lock
No
Yes
return CANCELLED
check if file is present or not
is file present
No
Yes
release lock
return NOT_FOUND
has timeout occurred
Yes
No
release lock
return DEADLINE_EXCEEDED
call remove() to delete
successfully deleted
No
Yes
release lock
return CANCELLED
set response values
release lock
add file to listOfDeletedFiles
return OK

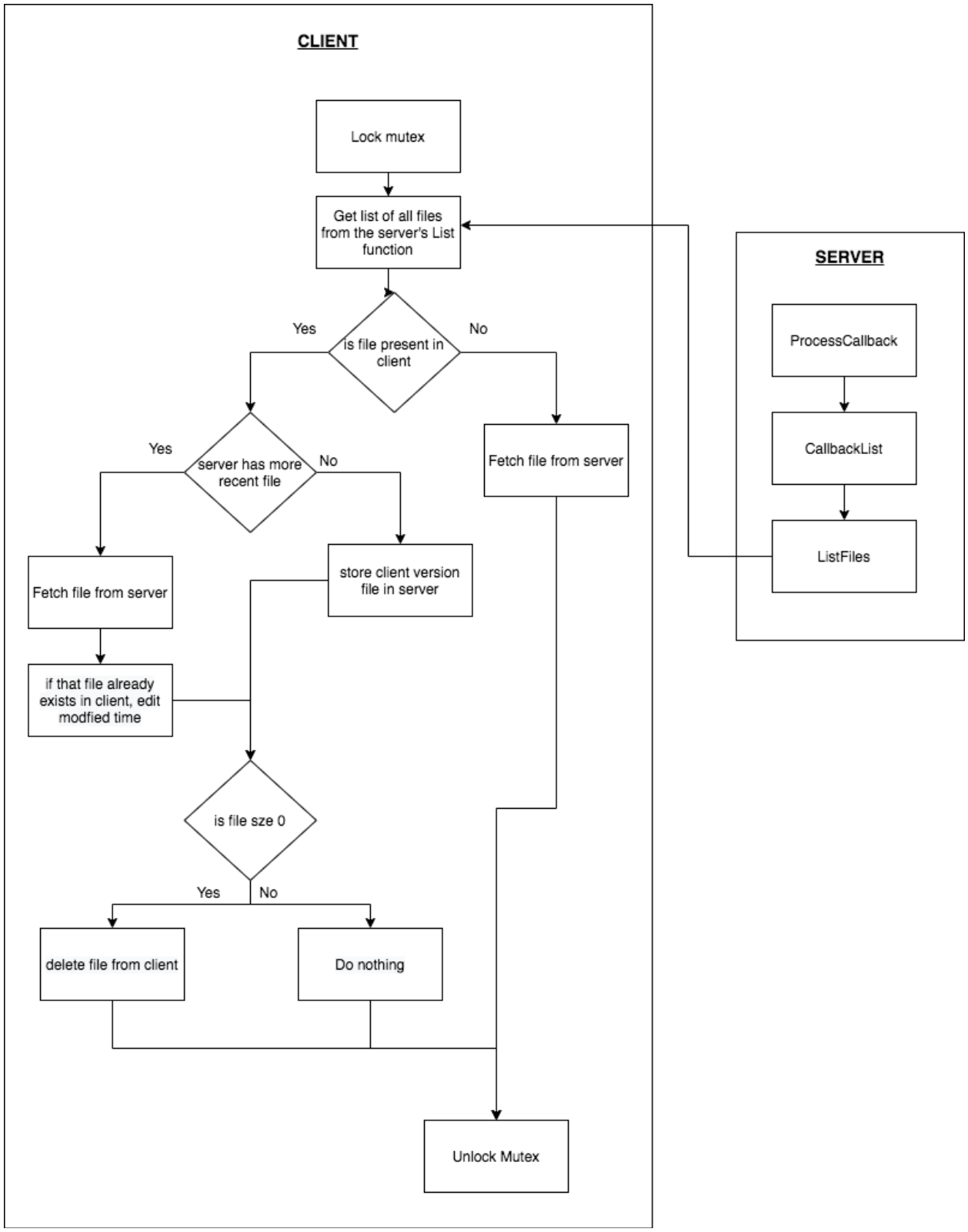
List



Stat is same as part 1.

dvenkatesh7

Callback process



References

https://www.youtube.com/watch?v=Yw4rkaTc0f8
https://www.youtube.com/watch?v=KPoBaBpr2XI
https://stackoverflow.com/questions/44468734/import-timestamp-in-proto-file-of-protobuf-for-grpc
https://grpc.io/blog/deadlines/
https://developers.google.com/protocol-buffers/docs/proto3
https://grpc.io/blog/deadlines/
https://developers.google.com/protocol-buffers/docs/reference/cpp/google.protobuf.util.time_util
https://codeforwin.org/2018/03/c-program-find-file-properties-using-stat-function.html
https://stackoverflow.com/questions/306533/how-do-i-get-a-list-of-files-in-a-directory-in
http://www.cplusplus.com/reference/cstdio/remove/
https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm
https://www.cplusplus.com/doc/tutorial/files/
https://grpc.github.io/grpc/cpp/classgrpc_1_1_client_context.html
https://grpc.github.io/grpc/cpp/classgrpc_1_1_server_context.html#a1424dee498921ed06e890c7134e88957
https://groups.google.com/g/grpc-io/c/tNlNfYgWv7k?pli=1
https://www.cplusplus.com/doc/tutorial/files/
https://www.tutorialspoint.com/How-can-I-get-the-list-of-files-in-a-directory-using-C-Cplusplus
https://en.cppreference.com/w/cpp/thread/mutex
https://community.particle.io/t/how-to-convert-string-to-uint32-t/36362
https://stackoverflow.com/questions/2185338/how-to-set-the-modification-time-of-a-file-programmatically