Problem Statement 3
[Online File Editor with Collaboration]

In this assignment, you will design an online file editor with online collaboration features. There will be a single server, which can handle multiple clients at the same time (max. 5 concurrent client connections).

Connection Phase:

The server creates a socket and listens for client connections on a specific port. The client should send a request for connection to the server, and an appropriate reply should be returned based on success or failure.

- ***Successful Connection:*** If successful, the server generates a unique 5-digit ID w.r.t. client socket id, and stores the client details in a *client records* file. A welcome message is returned to the client.
- ***Unsuccessful Connection:*** If unsuccessful, either because more than 5 connections are currently active or some other reason, an error message is returned to the client.

Server Functionalities:

After successful connection, a client can now access different functionalities by sending queries to the server.

– – ***Viewing active clients***

A client can request the server about the details of all active (currently connected) clients. The server returns a list of client IDs.

– – ***Uploading a file***

A client can upload a local file to the server. The server should create its own copy of the same file, and return success / failure messages.

The client successfully uploading a file is called the **owner** of the given file. The server should maintain a track of all files and owners in a *permission records* file. Apart from owners, there can be collaborators (see below).

– – ***Providing access to another client***

A client can allow access to other clients to *any file it owns.* If a client (say C1) asks the server to provide access to another client (say C2) to a particular file (say *f)*, the server dispatches an invite to C2. If C2 accepts the invitation, C2 becomes a **collaborator** of *f*. This invite (and subsequently on acceptance, the collaborator C2 itself) can be of two types, based on the type of access C1 wishes to give to C2:

- Viewer (V): Read-only privilege, C2 will not be able to modify *f*
- Editor (E): Read+write privilege, C2 can modify *f*

  – – – ***Accepting invites from other clients***

  As discussed above, if C1 invites C2 as a collaborator for *f,* an invite message should automatically be sent by the server to C2 with all details. If C2 accepts the invite, the server should perform the necessary actions as described above.

- ***Successful Invite:*** If the invite is successful, the server should send success messages to both C1 and C2, and maintain track of collaborators (and their access privilege V/E) along with owners in the *permission records* file.
- ***Unsuccessful Invite:*** An invite can fail in many scenarios (file does not exist, C1 is not the owner, C2 declines the invite, etc.). The server should send appropriate failure messages to C1 (and depending on the situation, to C2 also).

– – ***Reading lines from a file***

A client can request to read lines from a file by sending a query to the server. A client may be allowed to access the file (if it exists) *only if it is the owner or a collaborator (either viewer or editor) of the requested file*.

The client must also indicate in its query the range of line numbers it wishes to read, by specifying a starting and ending index. These indices may be in the range [-N, N) where N is the total no. of lines in the file (similar to what was discussed in Assignment 1). If only a single index is specified, only that single line should be returned. If no index is specified, the entire file should be read (start=0, end=-1 or end=N-1).

- ***Successful Read:*** If the read is successful, the requested line(s) are returned to the client.
- ***Unsuccessful Read:*** Appropriate error message to be returned to client based on situation (file does not exist / client does not have access / invalid line numbers)

– – ***Inserting lines to a file***

A client can request to insert line(s) into a file by sending a query to the server. A client may be allowed to modify the file (if it exists) *only if it is the owner or a collaborator (editor only) of the requested file*.

The client must indicate in its query the message (unlike Assignment 1, this message can contain newline characters) it wants to insert, and the line number it wishes to insert at, by specifying an index as above. If no index is specified, the contents of the message should be added at the end of the file.

- ***Successful Insert:*** Return the entire contents of the modified file
- ***Unsuccessful Insert:*** Appropriate error message to be returned to client based on situation (file does not exist / client does not have access / invalid line number)

– – ***Deleting lines from a file***

A client can request to delete line(s) into a file by sending a query to the server. A client may be allowed to modify the file (if it exists) *only if it is the owner or a collaborator (editor only) of the requested file*.

The client must also indicate in its query the range of line numbers it wishes to delete, by specifying a starting and ending index. These indices may be in the range [-N, N) where N is the total no. of lines in the file. If only a single index is specified, only that single line should be deleted. If no index is specified, the entire file contents (not the file itself) should be deleted (start=0, end=-1 or end=N-1).

- ***Successful Delete:*** Return the entire contents of the modified file
- ***Unsuccessful Insert:*** Appropriate error message to be returned to client based on situation (file does not exist / client does not have access / invalid line number)

Server Side:

- Create a socket and wait for client connections; once a client connects, generate a unique 5-digit client ID and store *client record* details
- Accommodate upto 5 concurrent client connections at the same time (refuse further connections if limit is reached, until some client leaves)
- Service multiple clients at the same time using select()
- If new file is uploaded / access privilege is granted for existing file, update the *permission record* details

Client Side:

- Connect to server
- Take command input from user, parse the command, and send appropriate query message to the server
- Retrieve server outputs and display to user

User Commands: To be entered at client-side input, parsed by client and appropriate communication (passing queries and collecting outputs) with server

1. `/users`: View all active clients
2. `/files`: View all files that have been uploaded to the server, along with all details (owners, collaborators, permissions)
3. `/upload <filename>`: Upload the local file *filename* to the server
4. `/download <filename>`: Download the server file *filename* to the client, if given client has permission to access that file
5. `/invite <filename> <client_id> <permission>`: Invite client *client_id* to join as collaborator for file *filename* (should already be present on server). *permission* can be either of V/E signifying viewer/editor.
6. `/read <filename> <start_idx> <end_idx>`: Read from file *filename* starting from line index *start_idx* to *end_idx*. If only one index is specified, read that line. If none are specified, read the entire file.
7. `/insert <filename> <idx> "<message>"`: Write *message* at the line number specified by *idx* into file *filename*. If *idx* is not specified, insert at the end. Quotes around the message to demarcate it from the other fields.
8. `/delete <filename> <start_idx> <end_idx>`: Delete lines starting from line index *start_idx* to *end_idx* from file filename. If only one index is specified, delete that line. If none are specified, delete the entire contents of the file.
9. `/exit`: Disconnects from the server, and then all files which this client owned should be deleted at the server, and update the permission file.

Submission Instructions:

- You should have one file for server **"server.c"** and one for the client, named as and **"client.c"**.
- Save this in a folder named in the format: **<Roll No.>_CL2_A3**. Compress this folder to tar.gz format, creating a compressed file **<Roll No.>_CL2_A3.tar.gz**. Upload this compressed file to moodle. *Example: If your roll no. is 21CS60R05, the folder should be* 20CS60R05_CL2_A2*, and the compressed file should be* 21CS60R05_CL2_A2.tar.gz.
- Not adhering to these instructions can incur a penalty.

Marking Scheme:

Setting up and maintaining server-client connections	**30**
Collaboration and Maintaining permission record	**25**
Uploading and downloading files	**25**
Read	**30**
Insert	**30**
Delete	**30**
Design Decisions and Error Handling	**30**