Project 2

In this project, you are going to implement the real file transfer function between the clients. The high level description of the requirements is as follows. Say, client A, is asked by the user for a file. As in the previous project, it first checks if it has the file and if it does not have the file, it will send a query to the server asking who has it. The server will choose one client, say, B, who has the file and tell A, along with a SHA-1 hash of the file. A will connect to B and B will accept the connection. After that B sends the file to A. Upon completion of the file transfer, A calls SHA-1 to get the hash of the file and compares it with the one server gave it. If they match, it sends a message to the server saying that "I got the file." Server will update the record for A and if next time a client asks for this file, server may ask A to upload the file.

In this assignment, the clients don't really store files in the disk. I will provide a function called "void generate_file(int file_id, int length, char *buf)" which takes the id of the file and a pointer to a char array (plus the length of the file) as parameter. After you call this function, the array pointed by buf will be filled with the content of the file. All files are of size 20,000 bytes. You can use "void find_file_hash(int file_length, char *file, unsigned char *hash)" to find the hash of a file. Parameter "hash" should be a pointer to an array of 20 unsigned char.

You need only modify the client and server code of the previous project for this project. For the client (just for simplicity, let's call it client A):

1. **Initialization.** Same as Project 1.
2. **Connect to the server**. Same as Project 1.
3. **Report to the server**. Same as Project 1.
4. **Open its own listening port.** Client A opens its listening port by calling the listen() function; it needs to do this because other clients may want to connect to it and download file from it.
5. **The while loop**. Client A enters an infinite loop:
    - If the user types "f", client A will ask which file the user wants and the user will input the file index. If the user types in, say, 10, client A first checks if it has file 10. If yes, it prints out a message like "I already have file 10." Otherwise, it sends a message to the server saying that "can you tell me who has file 10?'' The server will reply with a message with the information of a client, say, client B, who has file 10, including B's ID, B's IP address, and B's listening port number. The server also sends client A the SHA-1 hash of the file. Client A will print out the information it receives. Client A will then try to connect to client B by calling the connect() function. After client B accepts the connection, client A will send client B a message saying that "I need file 10." Client B will then send client A the file in 1000-byte blocks, where client B sleeps for 1 second after sending each block

(so the file transfer takes 20 seconds). After receiving the entire file, client A calls the SHA-1 function to get the hash of the file. Client A prints out the hash along with the hash server gave it, both as 20 unsigned integers. If they match, client A sends the server a message saying that "I have downloaded file 10 from client B." Client A then closes the connection with client B. If they do not match, client A asks client B to send again, which goes on until client A gets the file from client B.

- If the user types "q", client A quits: a) it first sends the server a message saying that it wishes to quit, b) waits until server closes the connection, c) terminate the program.
- Client A also monitors its listening port. If there is a client, say, client C, trying to connect to it, it will accept the request. Client A will then receive a message from client C containing the index of the file client C needs. It will then print out "Client C wants file (the file index), do you want to send the correct file?" If the user says yes, then client A goes ahead to send the file; otherwise, it will ask the user to input a key which is a single byte, which will be xored with every byte in the file to be sent to client C. After sending the file, client A prints out the SHA-1 hash of the file it sends to client C. If client C wants client A to send the file again, client A will ask the user again and repeat the process. Later, when client C closes the connection with client A, client A prints out "client C closed the connection with me!"

6. **Auxiliary functions**. Same as Project 1.
7. Note, that client A must use the select() function to get user inputs, read message from server, and read messages from the clients.

For the server:

1. **Initialization.** Same as Project 1.
2. **Accept requests.** Same Project 1.
3. **Answer client queries.** If a client, say, client A, sends the server a query for a file, the server first prints out client A's ID, IP address (the client's IP address can be found when calling the accept() function), and the file index client A is asking for. The server then checks its record of clients and finds one who has the file. In this project, the server replies with a client whose ID is (1) larger than A (2) is the smallest among such clients. That is, for example, if client A's ID is 5 and clients 0, 1, 3, 7, 9 are the ones who have the file, the server will give client 7 to client A. In the case that there is no client with ID greater than A has the file, the server "wraps around," that is, for example, if client A's ID is 5 and clients 0, 1, 3 are the ones who have the file, the server will give client 0 to client A. The server also computes a SHA-1 hash of the file and sends it to client A.

4. **Update client info.** After client A has downloaded a file, it will send a message to the server saying that "I have downloaded file what from whom." The sever will update the record for client A, i.e., add this file to its file vector. If a client leaves, the sever deletes its info.
5. **Respond to user command.** Same as Project 1.
6. **Respond to client quit message.** Same as Project 1.

Thread:

When a client A is asked to send a file, one possible implementation is that it only "focuses on" the file transmission, i.e., it sits in a loop and cares about nothing else. This might be fine if the total number of clients is not large such that client A will not be contacted again during this time. One much better implementation is to create a send thread, which will be responsible for sending files to all those who have requested for files. You probably want to use an array to remember the clients who have requested files, the file indices, the keys, and how much has already been sent for each client. If there are multiple clients, you would want to send file blocks to each client on a round-robin manner. **Extra 10% if you implement the file transfer using thread.**

The set of files for this project can be downloaded here, including common.cpp and SHA-1.c for generating the file and SHA-1 function. The server.cpp and client.cpp also implements relevant functions in Project 1. You are free to modify the code any way you want.