CSE322: July 2025
Assignment-1
Socket Programming

In this assignment, you will need to develop a file server system where clients can upload, download and request for files**. You have to use Java for the implementation.** The requirements for the system are as follows:

1. **Connecting to the Server:** Each client will have a unique username. A client can connect to the server by providing his/her username. If a client is already connected using a particular username, any further login attempts from the same username will be denied by the server, resulting in an immediate termination of the connection accompanied by an appropriate message to the client. Each client will be associated with a single username at any given time.
2. **Creating Directories:** When a client gets connected for the first time, the server creates a directory with his/her username as the directory name. This directory will serve as the storage location for all files uploaded by that client.
3. **Capabilities of a Client:** While connected to the server, a client will have the following capabilities:
    a. Look up the list of clients who were connected to the server at least once. The clients who are currently online should be distinguishable in the list.
    b. Look up his/her list of uploaded files, both private and public, and download any of these files. Private and public files should be distinguishable in the list.
    c. Look up only the public files of other clients and download any of these files.
    d. Make a file request consisting of a short file description, recipient and a request ID.
    e. View all his/her unread messages.
    f. Upload a file:
        i. The file access can only be private or public. It will be specified by the uploader at time of upload.
        ii. If the file is uploaded in response to a request, it must be uploaded as a public file of the uploader (automatically configured), and the uploader must provide the valid request ID.
        iii. When a requested file is uploaded, the server sends a message to the client who requested that file. Multiple clients can upload the same requested file and the server will send as many messages.
    g. View his/her upload and download history.

Implementation Details

1. The server incorporates some configurable parameters:
   1. `MAX_BUFFER_SIZE`,
   2. `MIN_CHUNK_SIZE`,
   3. `MAX_CHUNK_SIZE`.
2. While uploading a file, a client initiates the transmission by sending the file name and file size to the server. The server then checks the total size of all chunks currently stored in the buffer, plus the size of the new file. If the combined size exceeds the `MAX_BUFFER_SIZE`, the server does not allow the transmission. Otherwise, the server randomly generates a **chunk size** (between `MIN_CHUNK_SIZE` and `MAX_CHUNK_SIZE`) and sends a confirmation message to the client with the **chunk size** and a **fileID**. The **fileID** will be used as the file identifier for the rest of the file transmission. So, the server needs to maintain a link between the **fileID** and **file name**, along with other necessary information inside the server.
3. The client then splits the file into chunks based on the **chunk size**. For example, if the file size is 1040 KB and the server allows that client a maximum of 100 KB for each chunk, the client has to split the file into 11 chunks (the first 10 chunks with 100 KB, and the last chunk with 40 KB). These chunks are then sent sequentially to the server.
4. Upon receiving each chunk, the server sends an acknowledgment. The client sends the next chunk only after receiving the acknowledgment.
5. Once the client receives an acknowledgment for the last chunk, it sends a completion message to the server. The server then checks the file size by adding all the chunk sizes. If the size of all chunks matches the initial file size mentioned by the client, the upload is considered successful, and the server sends a success message to the client. Otherwise, the server sends an error message indicating failure and deletes all the chunks.
6. A client becomes offline if s/he logs out or gets disconnected from the server. If an uploader goes offline in the middle of a file transmission, the server should discard the incomplete files.
7. Unlike the upload process, the server does not closely monitor the download of a file. It sends the file with chunk sizes of `MAX_CHUNK_SIZE` and does not wait for the receiver's acknowledgment. The receiver gets a completion message from the server once the download is complete.
8. For the file request, the client will provide the short file description and the recipient, and the server will generate the request ID. File request can be either unicast or broadcast. If the client specifies a single recipient (by username), the server sends the request to that client. If the recipient is set to '`ALL`', the server broadcasts the request to all connected clients, both online and offline.
9. For the upload and download history, the server will maintain a log file for each client in that client's directory, containing the filename, date and time, upload or download action, and status (successful or failed).
10. Use separate threads where necessary (e.g., to upload or download multiple files in parallel, handle multiple clients, etc.).

Tentative Marks Distribution

Task	**Marks**
Connecting to the server	2
Looking up clients list	2
Looking up own files	3
Looking up others' public files	3
Uploading files	15
Downloading files	10
File requests handling	5
Message management	4
Upload-Download log management	4
Proper submission	2
Total	**50**

Submission Instructions and Deadline

Put your source codes in a folder named "xx05xxx", then zip it into "xx05xxx.zip" and upload it to the moodle. (Your submission must not contain any other file)

Deadline: 14th December 2025 (Sunday, 11:55 PM).

Please do not copy from any sources (friends, the internet, AI tools like ChatGPT, etc.), as doing so will result in a straight negative 100% marks.