**CSE 322 (A1 + B1)**

**Offline Assignment 1**

**Topic**: Client-server application development using Socket Programming

**Task**: Implementation of both client and server sides of a distributed file transfer application

1. **Requirement Specification:**

In this assignment, you are required to build a *distributed file transfer* application on client-server architecture. You are required to write two programs (in whatever language you prefer that supports socket programming): (i) the client and (ii) the server. The client program will enable remote users to share and transfer files among themselves in a distributed fashion where all the communication will be coordinated by the server. Specific requirements for the task are as follows.

* On startup, the client program will prompt the user for a username and password and then login to the server using the supplied username, password pair. We assume there is a list of already valid users; thus **you need not to add sign up feature.**
* Upon receipt of the login information from a client, the server will check the credentials against the valid <uname, password> list stored (in a simple text file) in the server and if a valid match is found, a **“login successful”** message will be sent to the client.
* Each client will have some sharable files (stored in some pre-specified folders, e.g. in a folder named ‘Shared’). After receiving the “login successful” message from server, client will send the list of sharable files to the server. The list will contain name and size of each file.
* The server will maintain a database of file sharing information. The database will contain the name of each file, its size and the list of clients who has a copy of the file. We assume any two files with the same content will have the same name and size and vice versa (i.e., any two files with the same name and size will have same content). When the server receives the list of sharable files from a client, it will update the database accordingly.
* Any client can get/read the file database (mentioned above) from the server by sending some command (such as ‘GetList’).
* Whenever a client wants download a file, it will ask the server (again using some command such as ‘GetFile’). The client will specify the filename and size.
* The server, upon receipt of a file download request from a client, will start to transfer the file in a distributed fashion to the requesting client from the sharing clients. To understand how the distributed file transfer will work, consider the following scenario.

Assume a file “abc.txt” is shared by four clients namely, C1, C2, C3 and C4. Let the size of the file is 2800 byte. Another client C5 wants to download the file “abc.txt” and requests the server accordingly. Let the client C4 is not available (logged off) at the time of this request. Hence the server will download the file from the remaining available three clients: C1, C2, and C3. It (server) will first request C1 for the first 2800/3=933 (i.e., file size/number of logged-on clients sharing the file) bytes. Next the server will request C2 for the next 933 byte and then C3 for the remaining 934 byte. That is, the server will distribute the load uniformly among all the clients sharing the file and **you must display the loads**. Thus the requested byte range can be tabulated as follows.

|  |  |
| --- | --- |
| **From Client** | **Requested byte range** |
| C1 | 0-932 |
| C2 | 933-1865 |
| C3 | 1866-2999 |
|  |  |

Note that, the server will not wait for the reply to come, rather will continue on sending the requests. The replies may arrive at any order independent of the order in which the requests were sent. Now, if any sharing client goes offline for some reason during the transfer try to adopt some way to complete the transfer. However, you don’t need to be optimal to handle this. Rather you can adopt some easy principles.

* A client when requested by server to send a chunk of a file specified by byte range, will read the file and send the requested range of bytes from the file accordingly. **Note that, the file data will be sent to server which will subsequently direct it to the requesting client.**
* Thus, a requesting client may receive different portions of the file at different points of time. Hence, it will be responsible for ordering the received bytes. When all the bytes are downloaded and ordered, it will be saved in the disk. Then the client will send a ‘download completion’ message to server.
* When a server gets a ‘download completion’ massage, it will update the file database by adding the name of the client to list of clients sharing the file.
* There should also be provision for log out. Whenever a user logs out, the server will make sure that no file request will be sent to this client.
* The server should keep provision for handling simultaneous connection from multiple clients. You must ensure that there is no inter-client dependency among client connections in server (such as blocking of one client by another client).

1. **Programming Issues**

* You must use socket programming in your implementation, both for client and server.
* You may use any programming language you wish (Java, C#, Python) as long as it supports socket abstraction to access OS’s native TCP service.
* Use object oriented programming. In that case, you will have at least two class files *client.class* and *server.class*. We will be running the client class multiple times in the same machine. Each client window will represent a separate user.
* You may take all the input and show output in console. Use of GUI is not a requirement.
* Take care to handle exceptions. Unwanted action by client should not crash the server.

1. **Ethical Issues**

Since all of you will be doing the same assignment, experience tells us that there is high chance of copying. **Let us warn you that any case of plagiarism (copying) will be handled severely with nearly zero tolerance and may even result in suspension from the course irrespective of whether you were the server (source of code) or the client (who copied the code).**

1. **Submission**

For A1 : Submit it by 13th March 2:00AM in Moodle

For B1 : Submit it by 14th March 2:00AM in Moodle

No delay will be accepted.