Programming Project for CSci 4211- Spring 2017

Objectives

In this project, you are going to implement a chatting platform. A chatting platform is basically an instant messaging platform where clients can connect to the server and chat with each other. Based on this platform, you are also going to implement several additional features such as peer-to-peer file sharing. The goal of this project is for you to gain experience in implementing both client-server model and peer-to-peer model socket programming. You are free to use any programming languages, for example C, C++, Java, etc., to implement the server application and the client application in this project as long as they can meet all the requirements of the project.

Design Overview

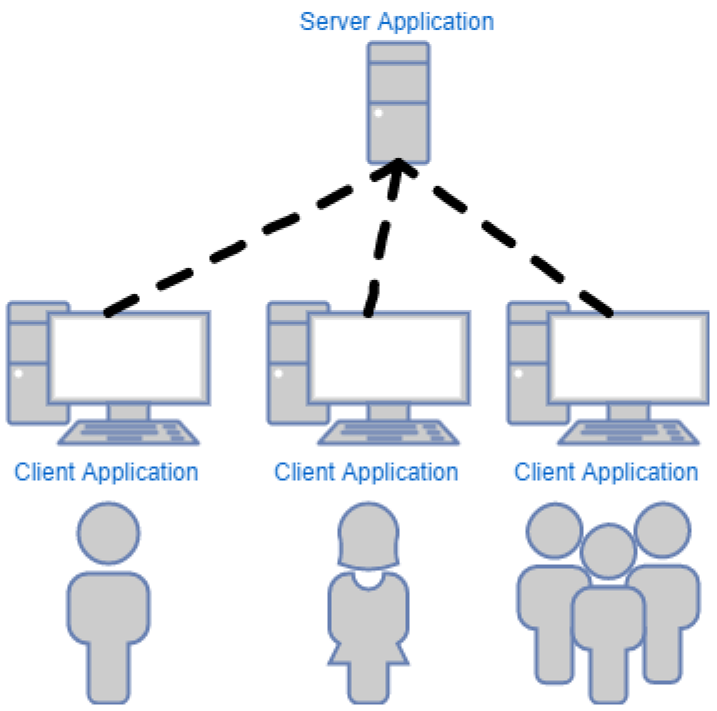

Figure 1 The client-server scenario of the chatting platform

Figure 1 shows the working scenario of the chatting platform. The users can issue commands to the client application, where the client application will process the commands and deliver the commands to the server application if necessary. In the following sections, we use server and client to refer to server application and client application respectively. There are two phases in this programming project. In **Phase_1**, you are going to implement the basic functionalities of the chatting platform, both the server and the client. In **Phase_2**, you are requested to implement a peer-to-peer file sharing environment based on the chatting platform of **Phase_1**. The detailed requirements of each phase will be discussed in the following sections.

The server of the chatting platform is responsible for receiving the socket connections from clients and processing the received messages from clients. When initialized, the server opens and listens to a socket, waiting for client connections. The server must be able to handle multiple client connections since this is a multi-user platform. When connected to the server, each client will join the same chatroom where they can see all the messages from other connected clients. This is the client-server model part of this project.

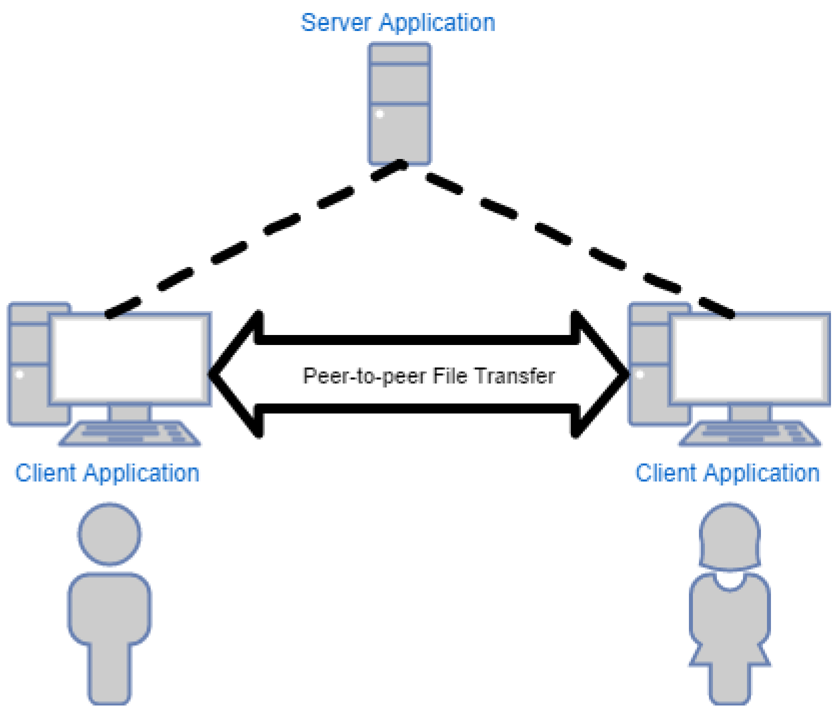

Figure 2 The Peer-To-Peer file transfer scenario in Phase_2

Figure 2 shows the peer-to-peer file transfer scenario. In the peer-to-peer model of file sharing, clients can inform the server the filenames and related connection information for sharing. Any connected client can ask the server for the list of available filenames and request a file with the file_ID. The server will respond with the corresponding connection information of that file (one of the clients who hold this file). The requesting client should establish a direct connection to the other client who holds the file for file transfer.

The details of server, client, and the communications in between will be described in the following sections.

Server

The server in this project is responsible for the following features:

- Handling client connections
- Client account management
- Client authentication
- Processing requests

Handling Client Connections

For handling client connections, the server must be able to handle multiple connections from different clients to create a multi-user chatting platform.

Client Account Management

For client account management, the server is responsible for the registration process of any new incoming clients and the import/export of a specific file for account information. The server should be initialized by executing the binary file with the filename as follows:

./server filename

The file contains the account information should be in the following format:

client_ID_0, password_0

client_ID_1, password_1

client_ID_2, password_2

The client can send a message to register for a new account. The message should be in the following format:

<REGISTER, client _ID, password>

The first column is the command_ID, REGISTER, indicates this is a command for registration. The detailed list of the command_ID will be shown in the following sections. The second column is the client_ID, and the last column is the password. After receiving the registration request, the server will check local database for the existence of such account. If the client_ID already exists, the server should return a corresponding error code. Otherwise, the server stores the client_ID with password and let the client enter the chatroom.

Client Authentication

When a registered client connects to the server, it can send a login request to the server in order to gain access to the chatroom. The message should be in the following format:

<LOGIN, Client ID, password>

The first column is the command_ID, LOGIN, indicates this is a command for login. The second column is the Client_ID, and the last column is the password.

The server receives the login request from the connecting client and check the database for client authentication. The client can enter the chatroom if the password matches the registered password of the corresponding Client_ID. If the Client_ID does not exist or the password is incorrect, a corresponding error code will be returned by the server.

Processing Client Requests

For processing client requests, the server has to provide several features for the clients. For examples, clients can use the CLIST request to get the list of connected clients from the server. The clients can also use the FLIST command to get the current file list available from the server for downloading. The detailed information about these additional Command_IDs will be explained in the following sections.

Client

In this project, a user uses a client to connect to a server and communicate with it to get other information in the client-server model. For peer-to-peer model, the client will have to act like a server to receive connections from other clients and process the requests. In this case, the client itself should be responsible for error handling, for example, handling the invalid format of a command. There will not be any error code for these situations.

Peer-To-Peer File Sharing

In **Phase_2** of this project, you are required to implement a peer-to-peer file sharing based on the platform of **Phase_1**. The clients connect to the server and provide the filename and their connection information that they want to share on the chatting platform by sending a FPUT command to the server. A client can send multiple FPUT commands to the server if it wants to share more than one file. The client will then open a socket and wait for the incoming requests from other peers for a file the client holds. Note that the client should also be able to receive multiple file sharing requests.

When receiving FPUT command from a client, the server should record the filename and the corresponding connection information for future requests of this file. If there are multiple clients sharing files with the same filename, the server should record all the connection information for getting the file, but will only return one filename when responding FLIST (there will not be multiple file_IDs for the same filename). If multiple clients request a file which has multiple providers (multiple clients have this file), the server should return the connection information of the file in a round-robin fashion instead of returning the same connection information of this file each time. Please note each time a client requests a file. The client needs to go through the server. Therefore, the server can assume the request will be satisfied and the requesting client now has a copy of the requested file.

Commands

The table of supported commands and their corresponding usage is as below. The client-server model commands need to be implemented in **Phase_1**. The other commands for peer-to-peer model file sharing are going to be implemented in **Phase_2**.

command_ID	Description	Format
	Client-Server Model Commands	
REGISTER	This command is for the client to issue a request to the server for account registration.	<REGISTER, client_ID, password>
LOGIN	This command is for the client to issue a login request to the server with client_ID and the corresponding password.	<LOGIN, client_ID, password>
DISCONNECT	This command is for the client to inform the server that it wants to disconnect from the server. The client can close the socket right after sending this command since there will not be response from the server for this command.	<DISCONNECT>
MSG	This command is for the client to send normal messages to the server. The server will broadcast the receiving messages to all the connected clients.	<MSG, message>

CLIST	This command is for the client to request the connected client list from the server. The server will respond with a list of client_IDs of all the connected clients, separated by commas.	<CLIST>
	Peer-To-Peer Model Commands	
FLIST	This command is for the client to request the available file list from the server. The server will respond with a list of filenames and corresponding file_ID for all the downloadable files provided by the connected clients, separated by commas.	<FLIST>
FPUT	This command is for the client to send the filename, IP address, and the port number for file sharing to the server indicating the available file for download from this client. The server will maintain a list of available filenames and assign a corresponding file_ID to the filename. The server will have to remove the filename from the list if the client disconnected from the server.	<FPUT, filename, IP_Address, port>
FGET	This command is for the client to download a file from the client which provides the file directly with file_ID. The server will return the corresponding connection information for the requested file, and then the client will connect to the other client for file downloading.	<FGET, file_ID>

Error Code

The table of error codes is shown as below. The first column of every response will be the error code, and the following columns may vary across different responses. Every successful request will have the error code 0x00 follows by the additional information of the response, except for the DISCONNECT command which the server will not respond to with any message.

Error Code	Description
0x00	Success
0x01	Access denied, wrong password or no such client_ID
0x02	Duplicate client_ID
0x03	Invalid file_ID
0x04	Invalid IP address and/or port
0xFF	Invalid format

The following table shows the possible error codes for each of the command.

command_ID	Possible Error Codes
REGISTER	0x00, 0x02, 0xFF
LOGIN	0x00, 0x01, 0xFF
DISCONNECT	N/A
MSG	0x00, 0xFF
CLIST	0x00
FLIST	0x00
FPUT	0x00, 0x04, 0xFF
FGET	0x00, 0x03, 0xFF

Examples for Command Interaction

- ***REGISTER***
  Success:
  Client_0: <REGISTER, client_0, 123456>
  Server: <0x00>
  Duplicate client_ID:
  Client_1: <REGISTER, client_0, 123456>
  Server: <0x02>
- ***LOGIN***
  Wrong Password:
  Client: <LOGIN, client_0, 654321>
  Server: <0x01>
  No such client_ID:
  Client: <LOGIN, client_999, 123456>
  Server: <0x01>
  Success:
  Client: < LOGIN, client_0, 123456>
  Server: <0x00>
- ***DISCONNECT***
  Client: <DISCONNECT>
  Server will have to remove the client_ID from the list.
  Client will have to terminate all the connected sockets for file transfer.
- ***MSG***
  Client: <MSG, “Hello world!”>
- ***CLIST***
  Client: <CLIST>
  Server: <0x00, client_0, client_1, …>
- ***FLIST***
  Client: <FLIST>
  Server: <0x00, file_ID0, filename_0, file_ID1, filename_1, …>
- ***FPUT***
  Client: <FPUT, filename, invalid_IP, invalid_port>
  Server: <0x04>
  Client: <FPUT, filename, ip_address, port>
  Server: <0x00>
- ***FGET***
  Client: <FGET, invalid_file_ID>
  Server: <0x03>
  Client: <FGET, file_ID>
  Server: <0x00, ip_address, port>