COMP/ELEC 416: Computer Networks

Project#1

Protocol Design and Development of a Networked Card Game

Group 7

Emir Atışay, 53590

Ege Berk Süzgen, 53714

Onur Şahin, 64232

23.02.2020

Table of Contents

Introduction:

This project is based on mainly three parts: interaction with the application layer, constructing a TCP protocol-based environment and FTP protocol for the file transfers. The first and the main part of the project is the TCP-based part. It involves client/server protocol, socket programming and multithreading. This part is where the WAR card game is played by the clients, with a TCP protocol-oriented communication. The TCP protocol allows server and client to communicate with each other in a message/command manner where they can only send and receive messages through a channel in a synchronized manner. This channel is implemented with socket programming. Server socket listens for the Client socket's connection and provides game, if there are two clients waiting. The second part, interaction with the application layer, aims provide the communication between the game-side of the project and the Application Programming Interface (API) of MongoDB. By achieving this connection, game-side becomes suitable for sending the game state to the database in order to save each state in the game. The final part of the project is the follower and master interactions. FTP protocol for the file transfers which is a TCP based protocol and capable of sending files through a channel. However, implementing a FTP protocol for the Master Follower communication in this game was not enough, because there are several reasons such as Follower's file request command that Master and Follower should send each other messages from a different channel but which is parallel to the FTP. So, Master and Follower communicates from two different channels. As a result, two different ports (one for each channel) are used in that part of the project.

The WAR Card Game and Server/Client TCP Connection:

The skeleton of this project is the Server/Client TCP Connection, since all the other parts are based on this connection's safety and utility. There are two different Main files to run in this project: one of them is the Main for the Client and the other one is the Main for the Server. When the Main for the server is executed, the server asks the user to determine whether the user is a Master or a Follower. If it applies that the upcoming user is Master, a server is created and three ports are assigned, 2 for follower connection and 1 for client connection. For the multiple port listening, a thread is observed. That new thread listens and accepts for the followers. However, in the body, Master deals with the client connections. If there are at least two client connected, Master creates a thread called GameThread. So, before opening the game thread, it waits until two Clients get into the same server and port.

When the Client Main is run, it connects to the Master server with the stated configuration settings which are IP address of the server and the port number for the socket. If the number of connected Client is two in the Master, it constructs a single Game Thread for them as it is mentioned. The Client object achieves this connection by the “ConnectionToServer” object that they have. This object is instantiated from the “ConnectionToServer” class. Client’s operations are declared in that class such as Connect, Disconnect, Send for Answer and Send for Init. Before those operations, it has constructor, which literally procures the connection of the client to the Server, by taking the server address and the server port as parameters. The Connect function connects the client to the server by creating three required components: socket, input stream and output stream. The Disconnect function does the reverse operations with connect function which closes three main components of the connection. The Send for Init function initializes each player’s (client’s) deck and sets required attributes. Basically, Sen For Init is the initialization of the game. In that function, client sends signal of ”0” and its name. As a reply, it takes signal of “1” and the deck for the game. The Send for Answer function is the function which guides client to play the game according to the game flow. It basically handles with the communication between the Client and the Master throughout the game. In general, it outputs signal of “2” and the card and inputs the result.

The other side of the game flow is in the Game Thread Class. It has a run function which the game is played on and some helper functions to supply game functionalities such as compare cards, initialize the deck and so on so forth. Except controlling the game flow, the Game Thread also deals with the Json and file write operations. There is custom class called Player which instantiates player objects with the required fields; player name, number of rounds player, remaining card of the player and the score of the player. Each player object is also created in the Game Thread and other stuff about player objects are handled in there. For instance, each game’s players (two clients) are written to the “player1 name-player 2 name.json” file to the intended directory. The intended directory will be explained in detail in the Follower part of the report. In addition, operations which are related to the database part of the project are mainly handled in the Game Thread, because Game Thread has the required information about players. For the client, thread coordinates the messages from both of the clients and operated game related functions with respect to those messages. Game related functions are the comparison of the players' card, updating the game state and provides the game state to the clients. An example message from Master is “3-0”, where it says that the round has won by the corresponding client. In general, the payloads are 2-3 bytes for the game. In case of quit from a client, game stops and choses the other client as a winner. And distributes the result of the game.

The FTP Protocol Master/Follower Connection:

Even though, the header only includes FTP protocol, this part also has its own TCP protocol which is stated as the command socket in the project manual. To clarify the FTP part, at first the FTP related Master will be explained. When the server (master) – side of the project is executed, the Follower Listener Thread is waits for Followers enters to the server. When a Follower enters to the server and the port of the follower part, the Follower Listener Thread opens a connection and creates a Follower Dealer Thread for each upcoming Follower. So, listener has the socket accept authorization, but the pipeline is through the dealer and the Follower. The Follower Dealer Thread literally deals with all Follower operations. That thread is constructed to have a parallel monitoring in different ports. The command which triggers the file sending operations are received from the Follower Dealer thread. At first, it makes a directory for each follower according to their entrance order. For instance, the first Follower which enters to the server is “Follower1”. Upon receiving the request from the Follower via the TCP connection in the command socket, it starts to perform necessary file transfer operations via the FTP communication. From the FTP based channel, it sends the file in a bitwise manner and from the command channel it sends the file's hashed value to the Follower. After that, it waits reaction from the follower side of the channel. In the Follower part, after receiving the hashed value of the file and the file itself, the Follower also calculates the hashed value of the file with the same hashing function. If the own calculated value and the received value matches, it sends a message to the Master that the file and the value passed the consistency check and as a result of this action the Master breaks the loop. However, if two values do not coincide with each other the Master performs the same operations in a loop until the Follower send the consistency check passed message. On the other hand, in order to handle both type of operations (file transfer and bilateral message communication), the Follower has its own buffered reader, print writer, file output stream, buffered output stream and input stream. The Follower class has similar implementation with the Client class. It includes its own Connect and Disconnect function also the file transfer flow functions such as Send for Answer and Accept File. In addition to this, as far as we can see, the follower's main aim is to handle with the distractions on the Master. So, a server socket is also constructed for the Follower server. However, as it is not necessary to deal with the corruptions in this project, that server socket is not instantiated.

The Application Layer; MongoDB Connection:

All operations about MongoDB are handled in the Master side of the project. The MongoDB server is initialized when the Master Thread is executed. It has its own default port which is 27017. From that port, Master connects to MongoDB. With that connection, Mongo Client object is observed. Here, Master is the user and has the client access to the MongoDB. So, the credential is created by the Master. After credential is created, database is got from the mongo client object. That database is sent to every Game Thread so that, each game thread can perform MongoDB operations on the master's database. Afterwards, a new collection is created when the Game Thread is started which takes the name of the game. Then for each state update, a MongoDB object called Document is added to the MongoDB. In addition, in the Game Thread, a timer is observed. With that timer object, MongoDB synchronized in a timely manner, which the interval of those updates is 30 seconds.

Conclusion:

At a glance, this project was dramatically comprehensive. It was an opportunity to apply what is taught to us in the course in a single project. Merging various concepts in a single application was one of the hardest and the most beneficial part of the project. All theoretical stuff that is seen in the lecture put into practice. In addition, working as a group, increases our individual working skills.

Task Distribution:

As suggested in the project manual, we examined the project in three different main parts. However, we did not completely separate the project and dealt some parts in group;

- Emir Atışay; Created General Project Structure and Architecture, Bug Fixes

Master-client interaction and the game logic
Master and MongoDB API interaction
Master-follower interaction

- Ege Berk Süzgen; Constructing parts in individual projects and Bug Fixes
  Master-client interaction and the game logic
  Master and MongoDB API interaction
  Master-follower interaction

- Onur Şahin; Testing projects individually and Running Trials
  Master and MongoDB API interaction
  Master-follower interaction

<u>References:</u>

- All the PS Codes and the tutorial links that is provided in the project description.