Colin Wlodkowski & Sam Hrncir

COMP 352

Homework 10

April 2019

# Overview of Application

The application we implemented was a tic-tac-toe game. It is a multi-user network application that supports multiple games of tic-tac-toe simultaneously. The application has a client-server style of operation. The server allows users to connect to it by running the TicTacToeClient application. The server manages pairing users to games and maintaining each game’s individual state. All of the game logic is housed on the server side. The server takes in a simple coordinate input representing a location on the board to mark with an “x” or an “o” (depending on which mark the player is using) from the client, checks if the location is valid, and then returns an updated board as a string to the client. The client side communicates with the server side by listening until the server specifies it to provide a line of input. While listening for this request, the client displays all strings sent to it by the server. This allows the user to know what is happening and when they are allowed to make a move. Thus, the game state, and graphics are all managed by the server and the client simply sends a coordinate input to the server and displays the "graphics" received from the server.

# Protocol Description

The server and client applications communicate using TCP protocol on the transport layer. Our TicTacToeServer is a TCP server because it has one socket, the welcomeSocket, that listens for new client connections and welcomes client contact. When a client connects to the welcomeSocket, the server creates a connection specific socket. This allows our server to talk with multiple clients. Clients connect to the server by creating a TCP socket and specifying the IP address and the port number of the server. The client socket is now bound to that specific server. We choose to use TCP for its services in providing reliable, in-order byte-stream transfer between client and server.

The way our server exchanges messages between participants is by telling clients when to talk, and when to listen. The server sends data to the clients by writing to a data output stream that is linked to the connection specific socket. The server tells the clients when they can send over data when the server sends the clients an END string. The server listens for exactly one line of input from the client using a buffered reader that is linked to an input stream reader that is linked to the connection specific socket. The client is set up to only allow the user to send over one line of input. The server then processes the client’s line of input via instances of the PlayerRequest class and TicTacToe class. It then replies to the client with a String that represents the updated game board. When the server replies to the client, the client listens to its buffered reader for data from the server, and upon receiving data, prints it to the users console. This is where TCP’s pipe abstraction is significant by the server sending over data that is uncorrupted and in order due to the client needing to show the user the data complete and in order for the applications to work successfully.

The server can handle multiple games at the same time with the usage of multithreading. Each time a client connects a new thread containing a new instance of the PlayerRequest class is ran. This object works in tandem with an instance of the TicTacToe game class to maintain the state of the board for the current game. Each players thread then can communicate with their specific game board without having to worry about users from another game interfering with their game state as the program ensures only two users can be assigned to a specific board at one time. This use of threading and checking the number of players allows the program to maintain the state of various games all running simultaneously. The server is also able to handle exceptions when players unexpectedly disconnect. When a player unexpectedly disconnects, the server sets the status of the game as false. On the server, the remaining PlayerRequest object checks the game’s status on its next turn, when it sees the status as false, the server notifies the client that the opponent disconnected and closes the connection.