**CS 421 – Computer Networks**  
**Programming Assignment II**

**Implementing a Tic-Tac-Toe Game over TCP**

**Due: May 24, 2023 at 11:59 PM**

**Introduction**

In this programming assignment, you are asked to write the Java or Python code for playing the famous Tic-Tac-Toe game using TCP. You will implement both the server and client sides of the program. The server side will maintain and keep track of the game, while the client side is used for sending the moves to the server and gathering information about the game from the server. You will use threads to accept multiple connections (players) on the server side. The goal of this assignment is to make you familiar with the internals of the TCP protocol, so using any (third party, core or non-core) API providing any level of abstraction specific to the TCP protocol is not allowed. You must implement your program using either the Java Socket API of the JDK or the socket module in the Python distribution. For Python, the use of any class/function from http or the requests modules is prohibited. If you have any doubts about what to use or not to use, please contact [**sarp.yenicesu@bilkent.edu.tr**](mailto:sarp.yenicesu@bilkent.edu.tr).

**Server Side**

Server side will represent the game master for our little 3x3 Tic-Tac-Toe game. It will accept connections from the client side and acknowledge them as separate players. Once the game has two connected players, it will decide the symbols (‘**O**’ or ‘**X**’) for each player, which player starts first, and assign an ID number for each of them. It will notify each client with a TCP message and with this notification the game starts. The game server should run with the following command:

java TicTacToeServer <port\_number> OR

python TicTacToeServer.py <port\_number>

where <port\_number> is the port that the server accepts connections.

When a client connects to the server, the server starts a new thread to receive messages from, and send messages to the new client, and continues to wait for another client to connect. For the specifications of the server side, it should do the followings:

* It should accept connections from clients and initiate a thread for each of them.
* Once 2 players are connected, it should notify the players with their symbols and their ID.
* It notifies which player starts the first and it starts the game.
* It should keep the state of the 3x3 TicTacToe board, and the turn information, and send these to the clients if requested.
* It should check the validity of the incoming client moves and send a message to the client if the move is invalid. Invalid moves are: Putting a symbol outside of the board, or at an already occupied location, or sending a move in other player’s turn.
* It should send each client the state of the board and the turn information at the beginning of each turn using the client ID. The client who has the current turn should receive an additional message emphasizing the turn information, i.e., “Your turn!”.
* It should check the winning conditions at each turn and notify the clients if there is a winner. Winning conditions are 3 same and successive symbols at a row, a column, or a diagonal. After all the grids are occupied, and there is no winner, Server should announce that the game is tied.

**Client Side**

The client first connects to the server and waits for its symbol and ID number. After they receive the game start notification, they make moves in their turn. Client side will start with the following command:

java TicTacToeClient <port\_number> OR

python TicTacToeClient.py <port\_number>

where <port\_number> is the server’s port number. In general, client side should satisfy the following specifications:

* It can request the board state and turn information at any moment throughout the game.
* Using the client ID, it should send a message to Server about their move.

**Threading**

In our game, the server will handle the requests coming from clients in seperate threads. So, your Server program should create at least 2 threads. This way, you will learn how real world applications handle thousands of connections in real time, but we will do it in a smaller scale.

Here, we outline a sample design pattern that you can use in your programs. According to this design, a separate thread is initialized in the main thread for each player:

|  |
| --- |
| function thread\_run( client){  while(game\_not\_over){ //  received = recv\_packet( client)  is\_valid = check\_if\_valid(received))  if is\_valid{  update\_board( received)  }  } }  // On main function  client1 = accept\_connection()  client2 = accept\_connection()  init\_thread( function=thread\_run, arguments=client1)  init\_thread( function=thread\_run, arguments=client2) |

With this implementation, you will get a **nonblocking** communication. For instance, if it is Player 0’s turn, however, Player 1 tries to make a move, you can immediately send an error message.

In this assignment, you are allowed to create your own designs. For more information about threading in Python and Java:

<https://www.pythontutorial.net/python-concurrency/python-threading>

<https://www.geeksforgeeks.org/multithreading-in-java/>

**Implementation Details & Report**

Write a **report** (PDF file) in which you explain the important parts of your code. We should be able to navigate the source code just from the report. The number of pages should not exceed 5; otherwise, you may **lose** points. The design and implementation details are up to you but do not forget to specify these details in your report clearly. Also, you should precisely indicate the format of the messages. Your program must be a console application (no graphical user interface (GUI) is required), but we expect some kind of representation to understand the board state clearly.

**Bonus Assignment**

As a **bonus assignment**, you can add additional features and mechanisms to your assignment, i.e., a timer for the turns. We will evaluate these features based on creativity and implementational complexity. The grading of this part is separate from the original grading scheme, and it is not mandatory. The bonus points that you will get from this part will be added to the original grade. If the addition is related to the network technologies, it will increase the contribution relatively higher than other bonus implementations.

**Example**

Suppose that we start the Server with the following:

java TicTacToeServer 6000 OR python TicTacToeServer.py 6000

And we start two clients with the following:

java TicTacToeClient 6000 OR python TicTacToeClient.py 6000

Let’s say that the Server assigns ‘X’ symbol to the client with ID=0 and ‘O’ symbol to the client with ID=1.

Let’s say Player 0 puts an X on the location (0,0), and Player 1 responds by putting an O on (1,1). Lastly, Player 0 puts an X at (1,4). Then, the command line will look like the following for the server:

|  |
| --- |
| Command-­‐line:    A client is connected, and it is assigned with the symbol X and ID=0.  A client is connected, and it is assigned with the symbol O and ID=1.  The game is started.  Waiting for Player 0’s move  Received X on (0,0). It is a legal move.  Waiting for Player 1’s move.  Received O on (1,1). It is a legal move.  Waiting for Player 0’s move  Received X on (1,4). It is an illegal move. (The move is out of the board.) |

The command line should look like the following for Client 0:

|  |
| --- |
| Command-­‐line:    Connected to the server.  Retrieved symbol X and ID=0.  Turn information: Your turn!  State of the board:  \_\_\_\_\_\_\_\_  \_\_|\_\_|\_\_  \_\_|\_\_|\_\_  \_\_|\_\_|\_\_  Put X to (0,0) (In this line, the client sends its move to the server)  Turn information: Player 1’s turn! (Wait for player 1’s move)  State of the board:  \_\_\_\_\_\_\_  X |\_\_|\_\_  \_\_|\_\_|\_\_  \_\_|\_\_|\_\_  Turn information: Your turn!  State of the board:  \_\_\_\_\_\_\_  X |\_\_|\_\_  \_\_| O|\_\_  \_\_|\_\_|\_\_  Put X to (1,4) (In this line, the client sends an illegal move to the server)  Server says: “This is an illegal move. Please change your move!” |

Client 1’s command line should be similar to Client 0’s.

Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do**

**not obey the submission rules below or your program does not run as described in the**

**assignment above.**

* The assignment should be uploaded to the Moodle in a zip file. Any other methods (Disk/CD/DVD) of submission will not be accepted.
* The name of the submission should start with [CS421\_PA2], and include your name and student ID. For example, the name must be

[CS421\_PA2]AliVelioglu20111222

if your name and ID are Ali Velioglu and 20111222. You are **not allowed** to work in groups.

* All the files must be submitted in a **zip** file whose name is described above. The file must be a .zip file, not a .rar file or any other compressed file.
* All of the files must be in the root of the zip file; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain:

– Any class files or other executables,

– Any third-party library archives (i.e. jar files),

– Any text files,

– Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.

* The standard rules for plagiarism and academic honesty apply; if in doubt refer to [Academic Integrity Guidelines for Students](https://w3.bilkent.edu.tr/web/provost/SAIC_Students.pdf) and [Academic Integrity, Plagiarism & Cheating](http://ascu.bilkent.edu.tr/Academic_Honesty.pdf).

**Important Notes**

**General Submission Rules:**

**1- )** Your submission should includesource code(s) (.java/.py).

**2- )** The format of the report should be **PDF**. (Do not upload .doc, .docx, or any other types). The number of pages should not exceed 5.

**4- )** Your submission should not contain any other files other than the source code(s) (.java/.py), report (.pdf) and **README**. No .txt files, no folders, and no IDE-related files should be included.

**5-)** Compress these files with the **.zip** format. (.rar, .7z, or any other compressing types will not be accepted.)

**6-)** Make sure to follow the rules in the “Submission Rules” section like the name of the zip file, method of the submission, etc.

**For Python Submissions:**

**1-)** Thecode should run with the “python3 ProxyDownloader.py <port>” command.

**2-)** Python version should be **3.6 or higher**. Other versions (like Python 2) are **not accepted**.

**For Java Submissions:**

**1-)** The code should run with the following commands:

Compile: “javac \*.java”

Run: “java ProxyDownloader <port>” command.

**2-)** Java version should be **8 or higher.**

**3-)** The JDK should be Oracle JDK (**not** OpenJDK)