CONCURRENT NETWORK APPLICATIONS ASSIGNMENT

Multithreaded networked application

Joe Pickering – P013373I

2019

Contents

[User Guide 2](#_Toc25785164)

[Design 3](#_Toc25785165)

[Functionality 3](#_Toc25785166)

[Concurrency Network Diagram 3](#_Toc25785167)

[Design of my system 3](#_Toc25785168)

[Explanation of strategy 3](#_Toc25785169)

[Bug handling 3](#_Toc25785170)

[Critical Evaluation 4](#_Toc25785171)

[What I learned: 4](#_Toc25785172)

[What went well/wrong 4](#_Toc25785173)

[Challenges Overcome 4](#_Toc25785174)

[What I would do differently 4](#_Toc25785175)

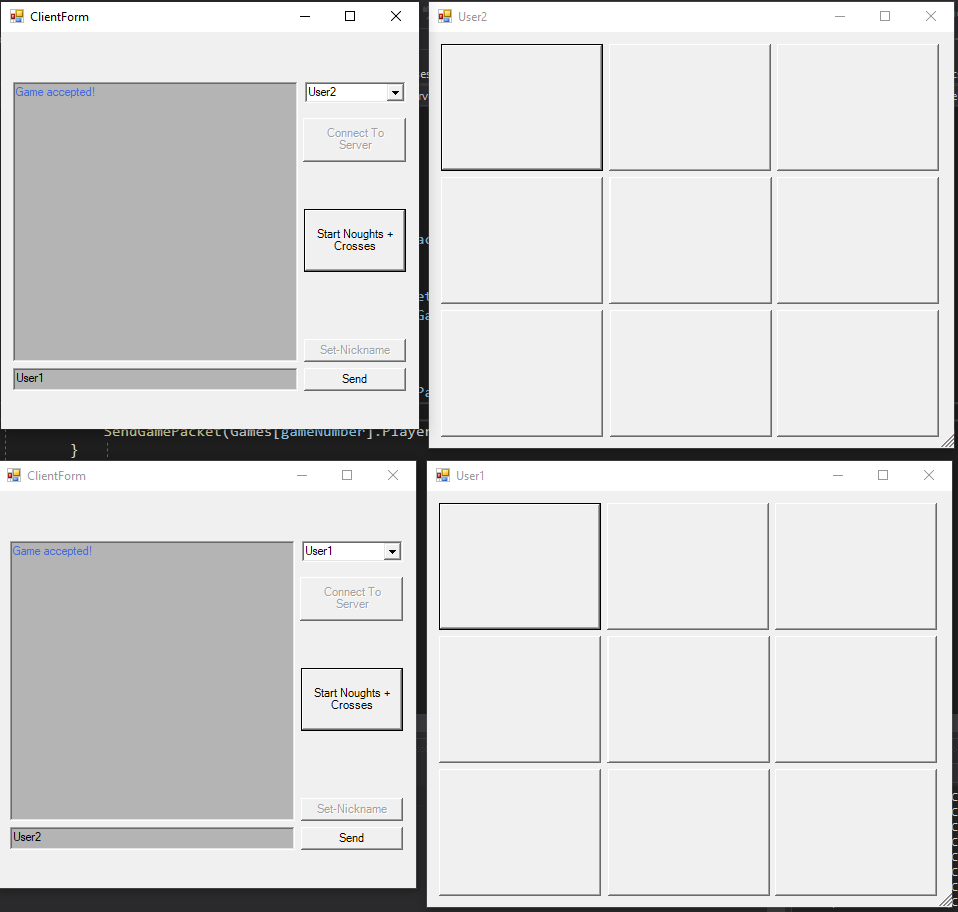
[References: 5](#_Toc25785176)

# User Guide

Upon loading, you will be presented with a form that shows two text boxes and some buttons, one of which allows you to set your nickname, the other, greyed out and unable to be used. Set your nickname by typing in anything you wish, and then click the “Create Nick Name” button. This will allow you to create a nickname. You can create a nick name and if you wish to, you can reset it by typing something new in and clicking the button again.

After this, the connect to server button will become available allowing you to connect to the server. If you click this button, you won’t be able to reset your nickname. After connecting, you will have your destination selection set to the server.

This is located in the drop down box on the top right. You can select whether or not to send messages to all users, via “server”, or to specific clients, via selecting them. If you try to send a message to yourself, then the server won’t process the message.

You may have noticed there is a button to start a “noughts and crosses game”. This button will only work provided the user hasn’t selected the server and has selected a user within the list. If this is selected, then the chosen user will receive a dialog box with an option to accept the game request or decline it. If they accept, then a new noughts and crosses dialog box is shown for both users, yet if the user chooses to decline, then nothing happens.

# Design

## Functionality

I have created multiple functions of this system such as:

* Setting client usernames
* Messaging the server to all clients (Server to all clients)
* Messaging to a specified client among the client list through the server (Client – server – client)
* Requesting a game to a specified client
* Starting a game of noughts and crosses with the clients.
* A packet sending system utilising serialisation and deserialization of data
* TCP and UDP connection and sending of packets

## Concurrency Network Diagram

(See my appendix for diagram)

## Design of my system

(See appendix for diagram)

# Explanation of strategy

## Bug handling

In various places for my project I had bug handling (Docs.microsoft.com, 2019). There was bug handling within the client form, in the simple client and in the server. Each bug handled was handled in a manner whereby I acknowledged program speed, threading and security.

First and foremost the client bugs that I fixed. The user could create a nickname and connect to the server then proceed to set another nickname, this was an issue since they connected as the form was created then they would select a nickname. I fixed this buy by forcing the user to create a nickname to start off with and then click the connect button. This then meant that the user was forced to connect with a nickname and afterwards the set nickname and connection boxes were set to disabled therefore users couldn’t create multiple nicknames nor could they connect twice.

Another bug that I fixed was to do with requesting games to other clients. First and foremost, users cannot request games with the server. This was a bug that started off yet I fixed this by forcing the user to request a game with another user other than the server. Furthermore, this works too for sending direct messages, it simply takes a value that isn’t the server’s in the list each client is represented with and then sends a message to that selected client.

One bug that was difficult to fix was that whenever a client requested a game with another client, it would create a dialog box for both client’s sender and the recipient. This was fixed by implementing a function that only send the dialog box to the recipient’s client rather than the senders as well.

A further bug that was fixed was to do with the windows forms appearing, this bug was to do with one client receiving multiple accept requests which forced it to create multiple forms for that client. I simply solved this by sending a game start packet to the users that would start the game, this would then set up the game.

## Race Condition handling

Race handling was used in my program in two different places, one for the client combo-box on their form and another for the chat box. Each of these items in the form were running their own threads and would sometimes have other objects invoke them so they could update them. For my chat box, when the client received a packet, if it corresponded to that of one that would update the chat box e.g a message or nickname, then the chatbox thread would be invoked and updated. This was done through a delegate (Docs.microsoft.com, 2019) for updating the chat window to ensure that if the thread had it’s data read, it was invoked first to give the update a chance.

Another place this was appropriate was for my combo-box. This system is multi-threaded since I required the combo-box to be set up similarly to how the chat window is, whereby I need a function to invoke it and update the contents. If the user receives a new user list packet, then the combo box needs to be updated with the new information for all users within the user list. This is done by once again invoking a different thread which is the chatbox and once available, updating the information to this new list.

## Mutual Exclusion

I did not have any events where the threads were deadlocked or required some code for them to be mutually excluded (Techopedia.com, 2019) yet I can consider some events which could follow in my code had something specific happen. For example, sometimes the user requires a game packet which is sent from one client to another. If the client required a calculation from the server, and the server was processing a function, then the tcp reader method thread could be deadlocked as the client is trying to access data from something whereby the server is trying to create, thus creating the deadlock.

# Critical Evaluation

## What I learned:

I learned multiple different aspects of networking as part of this project. I also learned many skills as part of debugging and refactoring code to keep my flow of data under good control and to keep it optimised. This enabled me to continue programming in the solution and implement my features such as sending game requests and starting the noughts and crosses game.

I also learned how to develop TCP and UDP connections by making use of another learned subject, packets for making use of serialisation and deserialization of data to enable consistent object sending. UDP proved to be difficult to start off with yet once I got it underway, I was fine.

## What went well/wrong

One thing that went wrong was the implementation for the game request and the showing of the noughts and crosses form, I was able to restructure the code here though as I implemented a game request packet enum list within the packet and sent that over, I’m proud of that since it was a pragmatic approach for optimisation and it prevents duplication.

Something that went well was my direct messaging for individual clients. I managed to send messages to other clients via a “combo-box” in windows forms, this allowed me to select an index for the combo-box and then use this index within the server code to directly send the message to a list of clients whereby the direction was the client located at that index.

Using Rolfe et a’s reflective model (Nicole Brown, 2019), when solving these parts is I would look at the different items whereby I would get stuck or would need to dive deeper into the code. I would ask myself, what is going on for my code, so what should happen and finally now what will I do.

## Challenges Overcome

One challenge that I overcame was through getting the client forms to finally show for the users, this proved very difficult as at first my primary option was to get the tcp and udp packets sending yet I was going to stop there. However, I overcame this by creating the request game feature and by also having the client forms for each user show up when they accepted the request game dialog. This was over-come by using Kolb’s learning style (Mcleod, 2019) whereby I firstly had an idea in my mind which was to have a game request packet with an enumerated list being the different states of the packet. I then reflected on this by implementing and debugging my idea by then considering if there was an easier approach and further then conceptualising whether or not this idea was a suitable one for my program or if I should have used a different one instead. This helped me out in several areas as while developing I utilised this learning method when developing the following: the combo box direct messaging selection, sending a game request, accepting and declining game requests, showing the windows forms and finally selecting the windows forms buttons.

## What I would do differently

Overall for this project I have gained a lot of great experience when working in this new programming paradigm as I wasn’t recently knowledgeable with threading, and had basic knowledge with file transfer protocols, I understood network logic so to have this working has finally provided an explanation as to many of my questions. Using the gibbs reflective cycle (The University of Edinburgh, 2019), I’d say that after creating the solution, I’m happier about my project yet during development I was very stressed in places as I had no initial carried direction for where to go. To evaluate my experience with the development, I’d say that it was incredibly stressful in places to get things properly working as for example when trying to send messages to every client, I wasn’t checking the specific client, and this was an issue I had made from not understanding what the code was supposed to do anyways. This has allowed me to understand that for development of a similar project in the future, I will look strictly into the system and development. I could’ve checked earlier on as to how the development process would carry and what would be included in the system since this would have definitely helped me out. If this project rose again, I would definitely soak in more information and development practices for multi-threaded network systems.

# References:

Docs.microsoft.com. (2019). *Delegates - C# Programming Guide*. [online] Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/ [Accessed 28 Nov. 2019].

Techopedia.com. (2019). *What is Mutual Exclusion (Mutex)? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/25629/mutual-exclusion-mutex [Accessed 28 Nov. 2019].

Docs.microsoft.com. (2019). *First look at the debugger - Visual Studio*. [online] Available at: https://docs.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2019 [Accessed 28 Nov. 2019].

Mcleod, S. (2019). *Kolb's Learning Styles and Experiential Learning Cycle | Simply Psychology*. [online] Simplypsychology.org. Available at: https://www.simplypsychology.org/learning-kolb.html [Accessed 28 Nov. 2019].

The University of Edinburgh. (2019). *Gibbs' reflective cycle*. [online] Available at: https://www.ed.ac.uk/reflection/reflectors-toolkit/reflecting-on-experience/gibbs-reflective-cycle [Accessed 28 Nov. 2019].

Nicole Brown. (2019). *Reflective model according to Rolfe et al. - Nicole Brown*. [online] Available at: http://www.nicole-brown.co.uk/reflective-model-according-to-rolfe/ [Accessed 28 Nov. 2019].