­­

Concurrent Network Applications report

ANTHONY STURDY

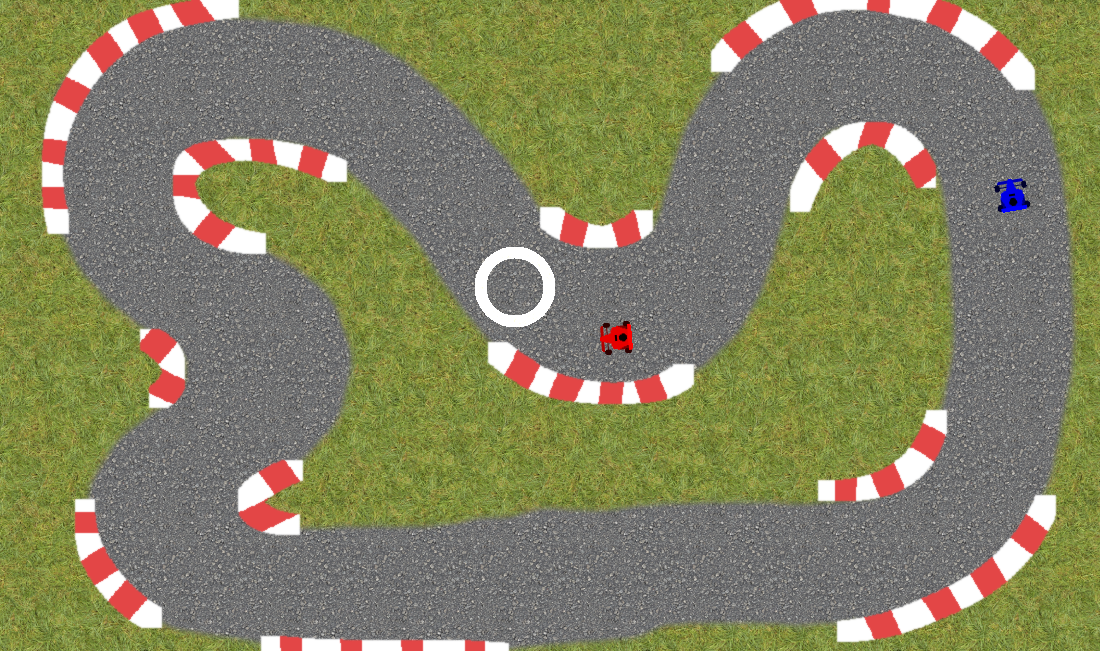


Table of Contents

[User Guide 3](#_Toc27407906)

[Design 3](#_Toc27407907)

[Functionality/Concepts Implemented 3](#_Toc27407908)

[Activity 4](#_Toc27407909)

[Class 4](#_Toc27407910)

[Explanation of Strategy 4](#_Toc27407911)

[Critical Evaluation 5](#_Toc27407912)

[Appendix 6](#_Toc27407913)

[Figure 1. Server Concurrency Diagram 6](#_Toc27407914)

[Figure 2. Client Concurrency Diagram 7](#_Toc27407915)

[Figure 3. Server UML Class Diagram 7](#_Toc27407916)

[Figure 4. Client UML Class Diagram 8](#_Toc27407917)

[Figure 5. Kolb’s Experimental Learning Cycle 8](#_Toc27407918)

[References 8](#_Toc27407919)

# User Guide

**Change Profile Picture**

Before connecting to a server, click the Profile Picture icon to the left of the Username input box. Select an image, then click Open. When you connect to the server, the profile picture will be applied.

**Connect to a server**

Enter server details in IP Address and Port input fields, ensure you have entered a username (custom profile picture not required), then press the Connect button.

**Send message**

Type your message into the large input field at the bottom of the application, then press the Send button.

**Send image**

Press the button with a file icon on it, next to the message input field. Select an image you want to send, then select Open.

**Send direct message**

Double click a use you would like to direct message from the User list on the left of the application, the username will appear in the message input field (e.g. @Anthony), type your message after this.

**Help command**

Type ‘/help’ into the message input field to get help from within the application.

**Challenge user to a racing game**

In the message input field, type ‘/game’ followed by the username of the use you would like to challenge (e.g. ‘/game Anthony’)

**How to play racing game**

Use the arrow keys to control your car, drive through the circle checkpoints, the first person to complete three laps of the circuit will win which is then output to the chat.

# Design

### Functionality/Concepts Implemented

* TCP Sending (Server and Client)
* UDP Sending (Server and Client)
* Nickname System (Stored on server)
* Customisable Server Details
* Profile Picture
* Clients List
* TCP Message Sending
* TCP Direct Messaging
* TCP Image Sending
* Command System (/help and /game)
* Two player 2D racing game using MonoGame

### Activity

**Server**

*(Figure 1).* This concurrency diagram of the server demonstrates how data flows through the application and how different threads are utilised to prevent the application from locking. Upon a client joining the server, a new thread is created which handles TCP packet reading for that client. The TCP read function uses a while loop which would lock up the whole server if it were on the main thread, having it on its own thread only locks that thread until another packet is received. The TCP thread is also responsible for creating another thread to handle UDP packet reading for the client.

**Client**

*(Figure 2).* This concurrency diagram of the client application demonstrates how data flows through and is processed by the client application. The application starts by first creating a windows form, which runs on its own thread. The windows form then calls a connect function upon button press, the connect function, similarly to the server application, starts a new thread to handle the TCP reading for the client. The function contains a while loop, which causes that thread to lock up when waiting for a packet but doesn’t lock up the entire application, due to it running its own thread. The TCP function is also responsible for creating the UDP reader thread upon receiving a UDP login packet. When a player receives a game start packet, a windows form is created which is run on a new thread. This is responsible for handling the MonoGame application and processing player information packets it receives, so each client knows where other clients are and what checkpoint they’re currently aiming for.

### Class

**Server**

*(Figure 3).* In this UML Class Diagram for the server application, you can there are 3 classes being used. The SimpleServer class is the main class, which then references both the Client and Game class. The Game class also references the SimpleServer class as it stores an instance of class to send packets. The Client class doesn’t reference either of the classes but is referenced by both, the client is responsible for handling the server’s connection to a client, for each client who joins the server, another Client is instantiated.

**Client**

*(Figure 4).* This UML Class Diagram for the client applications shows how the SimpleClient class references both the ClientForm and DrawTest class. The DrawTest class is what MonoGame uses to display the game. The ClientForm class is the form which is displayed for the chat room features. It includes the functionality for buttons and other GUI elements.

# Explanation of Strategy

To create a bug free implementation of features, I would test each feature to ensure it worked perfectly before moving onto the next feature. Once other features were added, I would also go over other features to ensure they have not been affected by adding more functionality. Testing the application at each stage of development allowed me to pinpoint any bugs quicker and meant I noticed anything which shouldn’t be able to happen which I may not have noticed if I did not test the application fully. For example, when the user connects to the server, they shouldn’t be able to attempt to connect to the server again until they have disconnected, so I disable the connect button and enable the disconnect button. Similarly, the profile picture and username cannot be changed while the user is connected, so I disabled the click event for the Image element and disabled the username text box so the user knows that they cannot change it while connected.

I did not have my problems with race conditions and using multiple threads, other than when I was implementing the 2D graphical game. I had a problem where the program would crash if the server sent another packet along with the player information packet, but the crash would not occur if a breakpoint was set and the code was stepped through slowly, then continued. I believe this was due to the stream of data becoming mixed and the BinaryFormatter was unable to deserialise the packet. To fix the problem, I merged the data with the player information packet which was being sent every frame anyway, which fixed my problem but resulted in a small amount more code being ran on the server to process each packet.

The definition of Mutual Exclusion is “a program object that prevents simultaneous access to a shared resource. This concept is used in concurrent programming with a critical section, a piece of code in which processes or threads access a shared resource.” (Techopedia.com, 2019). C# has a built in Mutex class which allows for easier mutual exclusion, I did not use this in my application as there are not multiple threads accessing a single resource. This might have been a fix for my problem with the binary formatter, as I believe multiple threads were writing to the MemoryStream which caused the BinaryFormatter to crash upon attempting deserialisation, but I solved the problem using another method so I did not need to use Mutex.

# Critical Evaluation

Throughout the development of this application, I have learned how data is sent across a network via a server, using TCP and UDP transport protocols, and how to implement the theory in an application using C# and Windows forms. Additionally, I have learned to implement and use MonoGame using MonoGame.Forms (<https://github.com/sqrMin1/MonoGame.Forms>) within my application to render a 2D graphical game and process a user’s input.

The Kolb Experimental Learning Cycle is “a four-stage learning cycle in which the learner 'touches all the bases'” (Mcleod, 2017). Each stage of learning and implementing the systems in my application can be compared to Kolb’s Learning Cycle *(Figure 5)*, for example learning the theory of sending data over a server in lectures would be the equivalent of “Learning from abstract conceptualization (thinking)” (Managingforimpact.org, 2012) Implementing features learnt during the lectures would be “Learning from active experimentation (doing)” (Managingforimpact.org, 2012), Completing tasks and having an understanding of the systems implementation would be “Learning from concrete experiences (feeling)” (Managingforimpact.org, 2012) and finally, the critical evaluation and reviewing my work would be “Learning from reflective observation (watching)” (Managingforimpact.org, 2012) which will allow me to further understand and learn from my experience which can be applied to future work, restarting the cycle.

The development of the chat room is the part of the development which went well for me. I didn’t come across many problems, was able to make progress fast and learned a lot about using Windows forms while doing it. I enjoyed creating small features and details to enhance the user experience, such as the profile picture system and optimising the profile picture system so only the necessary data is sent.

While creating the graphical game, I had a few problems with sending packets to players. Upon the server receiving game information a packet, I would send another packet to a client if they were inside the current checkpoint, but when the client would receive the packet, it would crash the BinaryFormatter. This only happened when running the game at real time, if I used a break point and stepped through the code it would work perfectly. To solve the issue, I put the players current checkpoint position in the packet along with the opponent’s position and rotation. This lead to some confusing code and more problems, but was eventually solved and the checkpoint system worked perfectly.

I have overcome a few personal challenges when developing the application. Before starting the application, I had no knowledge of how send data over a network, and minimal knowledge of how it works. I’m glad I was able to use the resources available to learn how to, and implement the networking aspects of my application. Another challenge I faced was using MonoGame, as I had never used a graphics framework with C# before except for using Unity. The MonoGame.Forms (<https://github.com/sqrMin1/MonoGame.Forms>) library streamlined the implementation of MonoGame with Windows forms, and was able to start rendering sprites and taking input in a short amount of time.

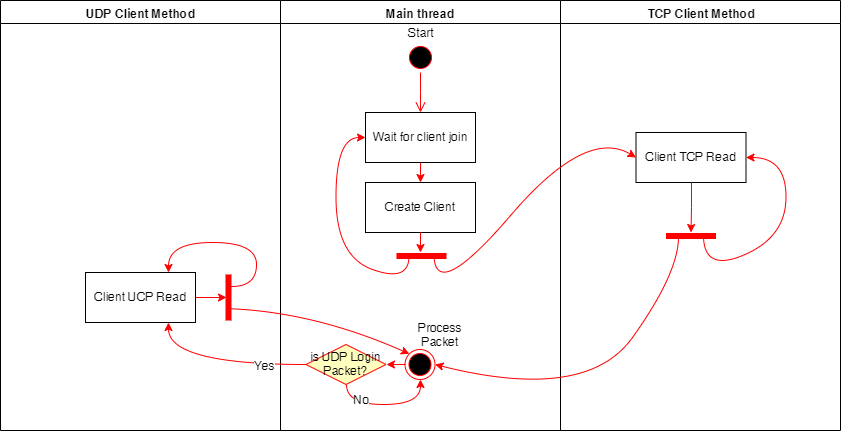
If I were to create the application again, I would do a few things differently. Firstly, I would like to create a GUI element in the client form to request games and view currently open non-full games which would allow anyone to join without having to directly challenge a user to a game. I believe this would be more polished than the command system, but I did not have time to implement the feature.

Another feature I’d like to have is bigger tracks to race around with support of more than two players. I think having 5-10 players would make the game much more fun to play, and having bigger maps to accommodate more players would make the game more interesting.

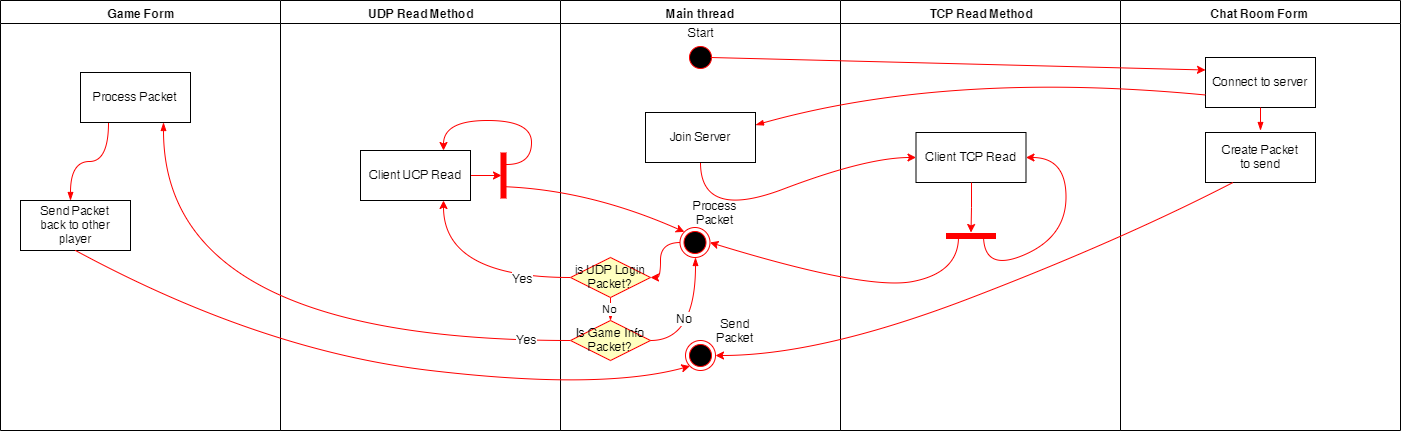
Lastly, I’d like to have implemented a voice chat room in my application. This would have been interesting to research and implement, and I think being able to voice call other users would make a great addition to the chat room, making it feel like more of a modern application.

# Appendix

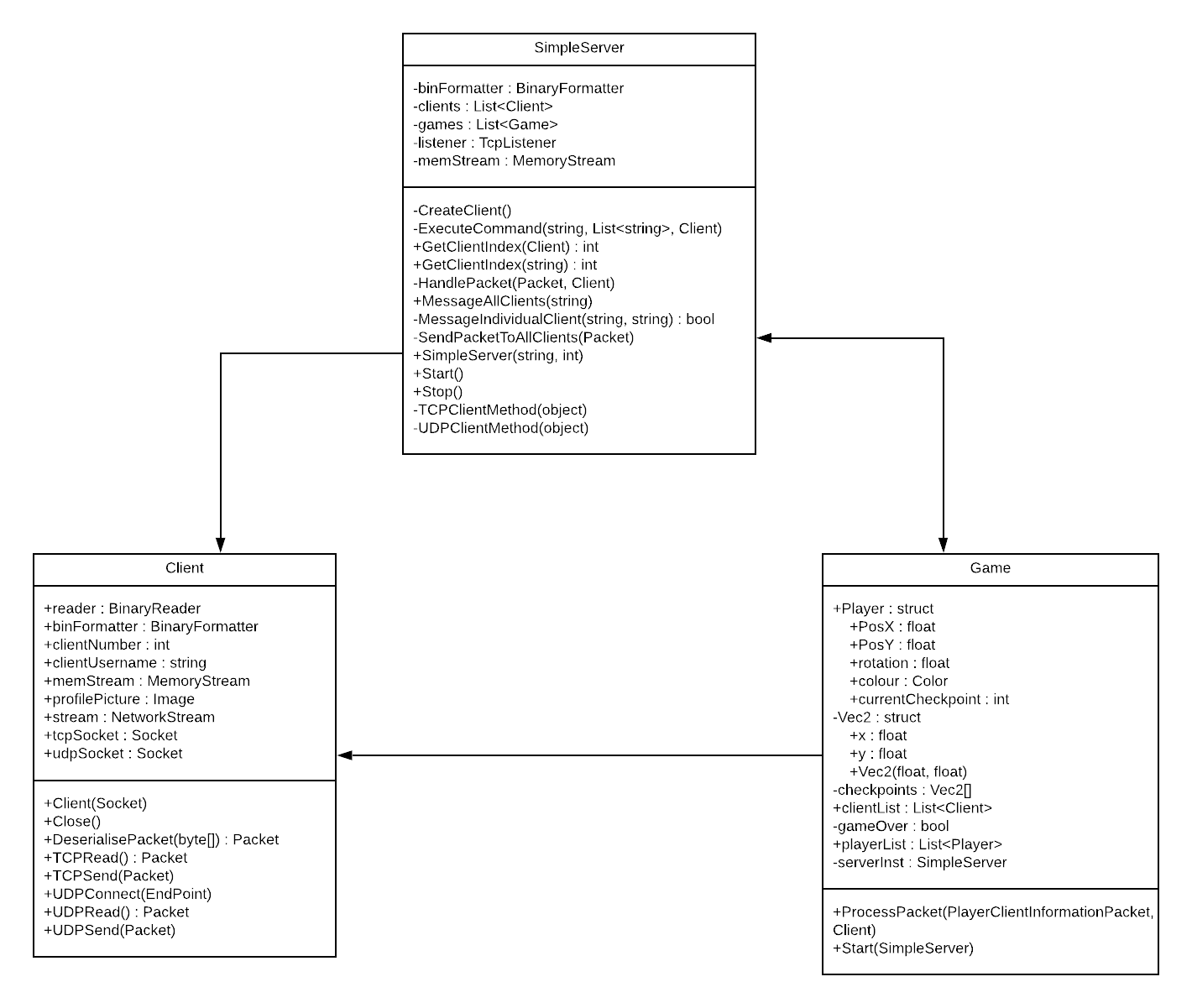
### Figure 1. Server Concurrency Diagram



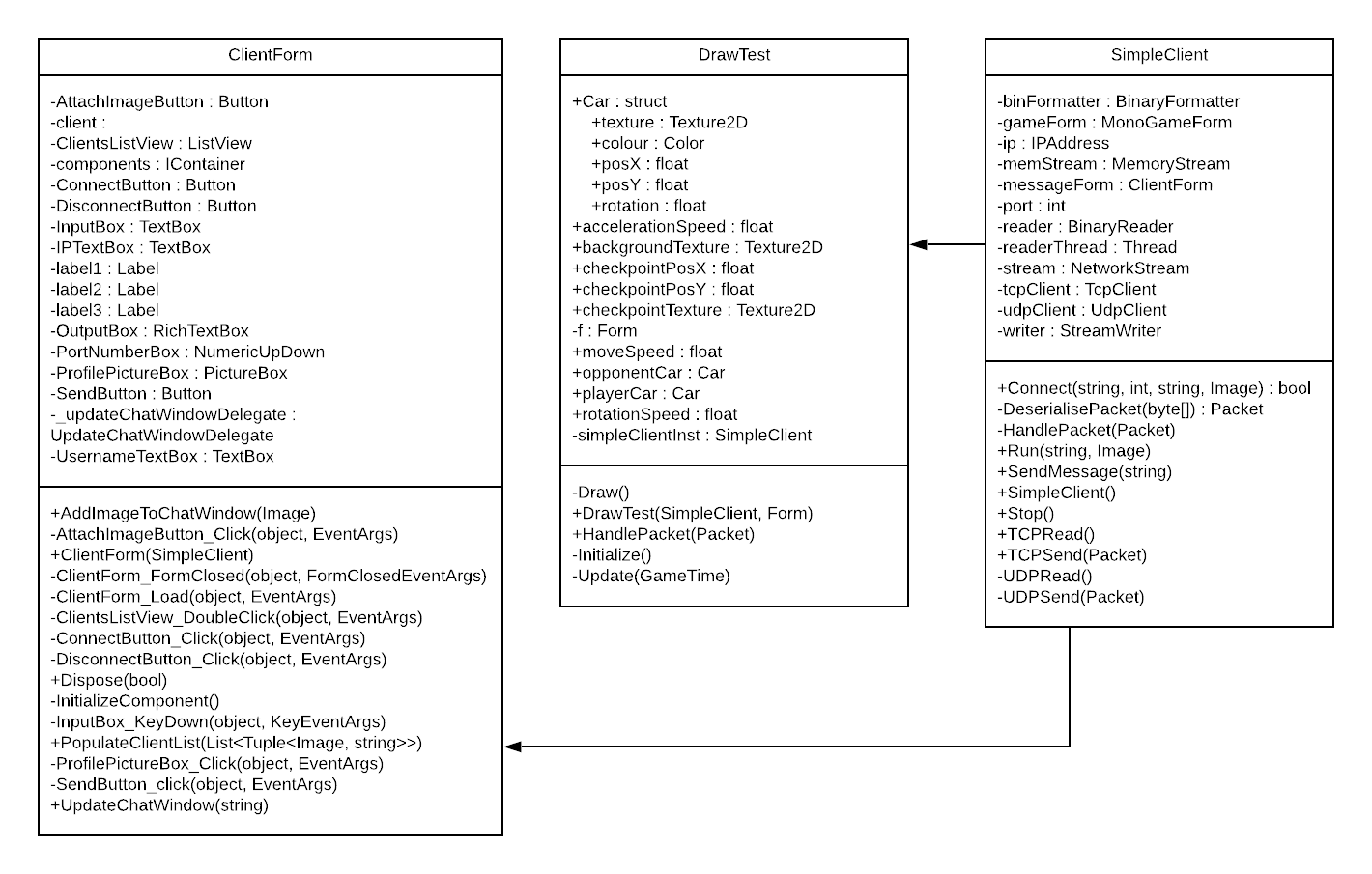
### Figure 2. Client Concurrency Diagram



### Figure 3. Server UML Class Diagram

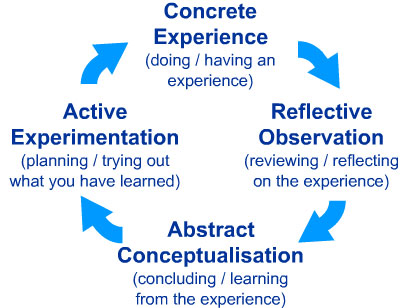
****

### Figure 4. Client UML Class Diagram



### Figure 5. Kolb’s Experimental Learning Cycle

Source: <https://www.simplypsychology.org/learning-kolb.html>



# References

Managingforimpact.org. (2012). Kolb's learning cycle and learning styles. [online] Available at: http://www.managingforimpact.org/tool/kolbs-learning-cycle-and-learning-styles [Accessed 16 Dec. 2019].

Mcleod, S. (2017). Kolb's Learning Styles and Experiential Learning Cycle | Simply Psychology. [online] Simplypsychology.org. Available at: https://www.simplypsychology.org/learning-kolb.html [Accessed 16 Dec. 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 16 Dec. 2019].