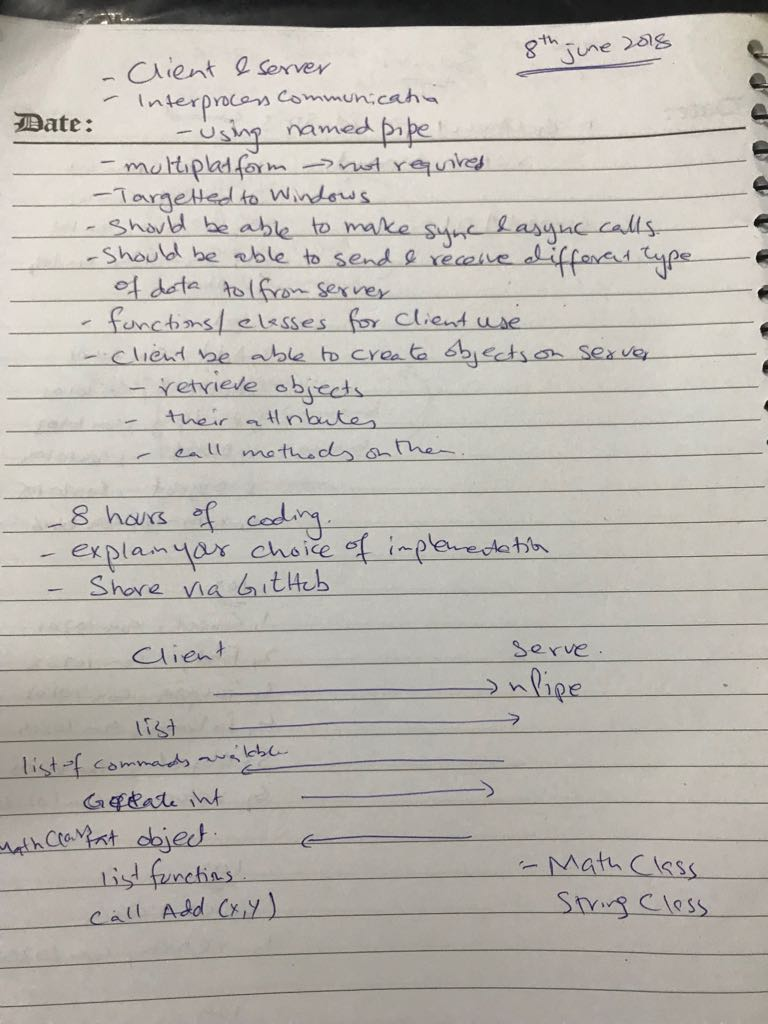
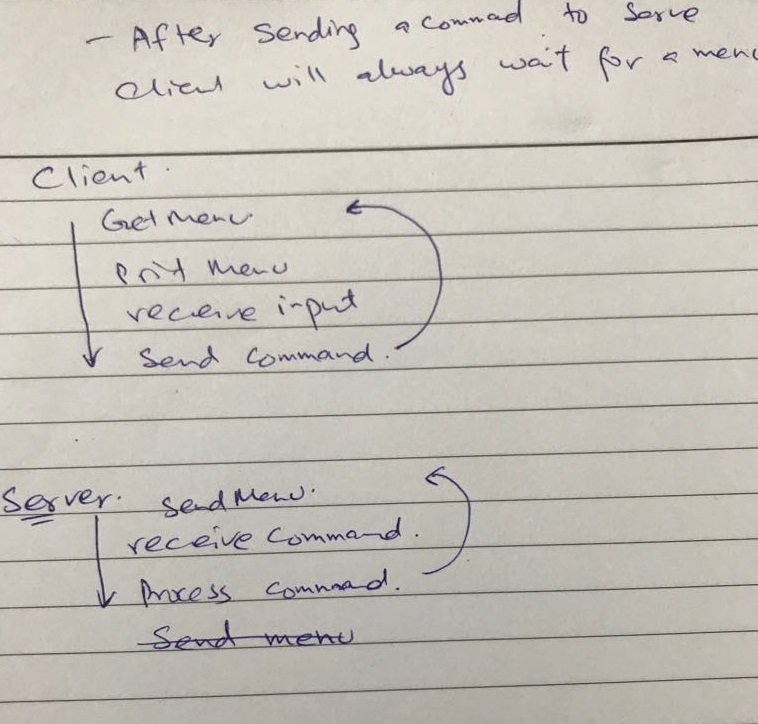
This is the code after around 8 hours of coding. I could not sit and do it in one session because of my commitments here. I worked in three different sessions amounting to 8 (3+3+2) hours of time. These eight hours does not include the time to document and upload to GitHub 😊.

I've implemented a client and server using following plan:

1. In first session I spent around 1 hour on how to implement the main requirements within the specified time. I knew I’ve to stop after 8 hours, so I wanted to do it in a way that it seems logical to me and I have explicit goals to accomplish. I have worked with synchronous named pipe connections earlier so I prioritize and planned my tasks with synchronous communication as top priority. This is a part of planning to prioritize keeping in mind the limitations and strengths. So, if I had asynchronous communication as top priority it *may* have consumed the time without letting me complete the other tasks. In addition, if the async part had taken more time and still remained unfinished it would have negatively impacted my motivation. Initial plan is attached where I’ve made bullets of what is required and what is not required. To keep me focused on the target. It not only highlights the functional requirements but also list down the non-functional requirements (e.g. time, documentations etc.). Lastly, it plots the interaction between client and server.



1. Client and server follow a simple workflow of interaction. This workflow is easily deductible from the code. After it’s been understood, it will not only help in the implementation but will also help in maintenance of the code.



Implementation details:

1. The choice of implementation is using Visual Studio and C++ with MFC. Basically, client and server are console based implementation but I’ve used the libraries from MFC for strings, threads and named pipe connection. As it is not required to be multiplatform and only focused on Windows it will help in speeding up the implementation.
2. Client and Server can send and receive different types of data. Basically, they operate on strings but it also supports the serializable objects. In addition, the menu sent by the server to client is comma separated values. This could have been XML or JSON to emphasize the different type of data, but for the simplicity’s sake it is kept to csv only. First value in the menu is the command ID that will be displayed to the use, second value is the string representation of the command and third is what the server will be understandable to server. So it will know in what “context” the command was issued.
3. Server had the object cached e.g. StringClass and MathClass. These can be created for the respective sub-menu and can be initialized with some random values.
4. Each submenu shows list of commands that “invokes” a function on the server and returns the result. The server registers these methods for the client and client can call them and retrieve the results.
5. Command number 4 in main menu is for the async call. The client opens two connections to the server when created, one for the synchronous calls and one for the asynchronous calls. For async call the implementation is bare minimum. It might not server the actual purpose of the async-ness, but it is there to show how it can be done.
6. Mainly focused on the tasks at hand and keeping them as simple as possible. And it is only an initial draft.