COP5615: Distributed Operating System Principles
Fall 2023

Programming Assignment #1

REPORT

PA1_Team14

NAME	UFID	EMAIL
Nitish Chandra Mahesh	36139637	n.chandramahesh@ufl.edu
Anuj Papriwal	37008807	papriwalanuj@ufl.edu
Pawan Kumar Jagadapuram	73643747	pjagadapuram@ufl.edu
Gopi Amarnath Reddy Bekkem	72188579	bekkem.g@ufl.edu

Overview

In this project, we implement socket communication between a server and clients using the TCP protocol. We have set up the project as an F# console application. The server runs on port 9000 which is not reserved for any other applications. We use the inbuilt .NET System libraries for I/O, Asynchronous/Task/Thread handling and socket handling. The code handles graceful shutdown of server and client connections. Client input validation and error handling is also included.

Compiling and running the project

Requirements:

- *.NET SDK v7.0*

Steps to compile and run:

1. *Compile and run the server program executable in the root folder of the project through the following command*
   > dotnet run

2. *In separate terminals, change directory to the **Client** folder and similarly run the Client program*
   > cd Client
   > dotnet run

Description of the code structure

The application is set up as two separate F# projects bundled in one. There is a client project with Client.fs as the only compilable file and client.fsproj to manage the F# project. The server project has the main file Server.fs in addition to server.fsproj to manage the project. Both F# projects use .NET 7.0 as the framework.

Server.fs

1. The entry point of the program is the main function which calls the `createServer` function.
2. In the createServer function, we instantiate a TCP Listener from the .NET Sockets library and accept incoming connections in a loop on port 9000. For each client, we create a unique GUID, and a CancellationTokenSource (used for thread-safe cancellation of async tasks). An Async Task is created to handle each client's requests (`listenToClient`) and this is spawned into the thread pool with `Async.StartAsTask`.The server workflow is then itself executed as a task on the thread pool.
3. In the `listenToClient` function, we first obtain the client stream to send and receive data over the socket. After the greeting "Hello!" is sent to each client, this function asynchronously reads the stream, waiting for a client's message (`Async.AwaitTask` will wait for the read function to return). We then proceed to perform input validations and

checks returning appropriate error codes for invalid inputs. If there are no errors, we proceed to run the mathematical commands sent by the client and return the result to the client via the communication helper functions.

4. Communication helpers: `sendToClient` and `broadcastToConnectedClients` are functions that are used to send messages (strings) to a single/all clients. We use the NetworkStream to read and write from the stream attached to each client. Async constructs such as `Async.AwaitIAsyncResult` are used to asynchronously write to the network stream and wait for its execution.
5. Bye and terminate commands are handled by helper functions for termination.
6. Shutdown and termination functions: `disconnectClient, disconnectAllClients` and `shutdownServer` are used to dispose of tasks, release resources and close socket connections and listeners. Cancellation Tokens are used for thread-safe cancellation of async computations.
7. Clients and associated tasks/tokens are stored in the list `clientTaskList.`

Client.fs

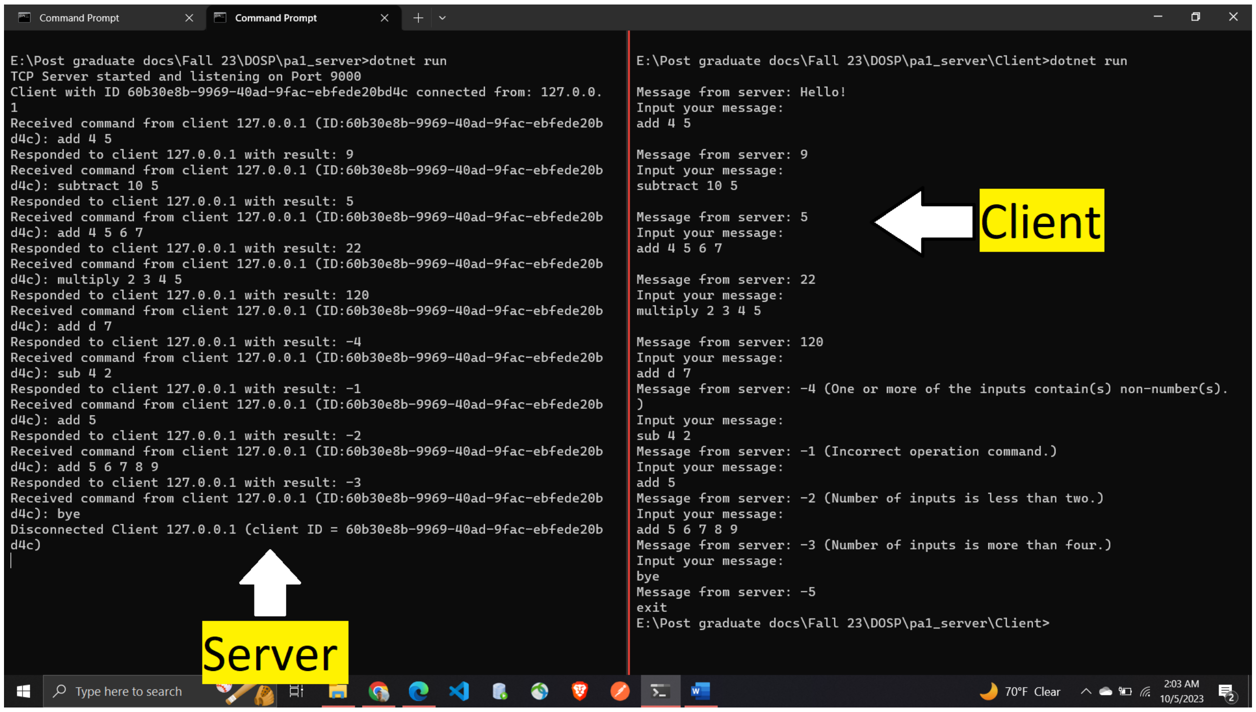
We have implemented a custom type called Client that builds upon the TcpClient type provided by .NET. The Client type has the following methods implemented in it to suit our use case:

1. `connectToServer`: This method starts a connection with the server and runs the "`listenAsync`" method.
2. `listenAsync`: This method creates a network stream and reads the stream using a disposable stream reader. The stream gets disposed of gracefully when the client is closed. This method is also responsible for printing outputs, mapping the error code to a readable format for the user. If a "-5" error code is received from the server the code triggers another method "`closeConnection`" responsible for closing the client connection.
   As per the requirements of the project this method first listens to the server "Hello!" and then runs another method "`readUserInput`".
3. `readUserInput`: This method simply reads the console input from the user and triggers another method "`write`" with the values that have been input.
4. `write`: This method checks if the client is connected to the server or not if yes, starts a new stream writer and sends the user input to the server using the same. Once the input has been sent we flush the stream to ensure data correctness for the next time message that we might have to send.
5. `closeConnection`: This method closes the connection with the server and exits the environment.

In the main function we start with `connectToServer` function by giving it the right port and endpoint to connect with and then use the function `runIndefinitely` which is a recursive function that keeps calling itself after making the main thread sleep for 1 second at a function call.

Result of executions

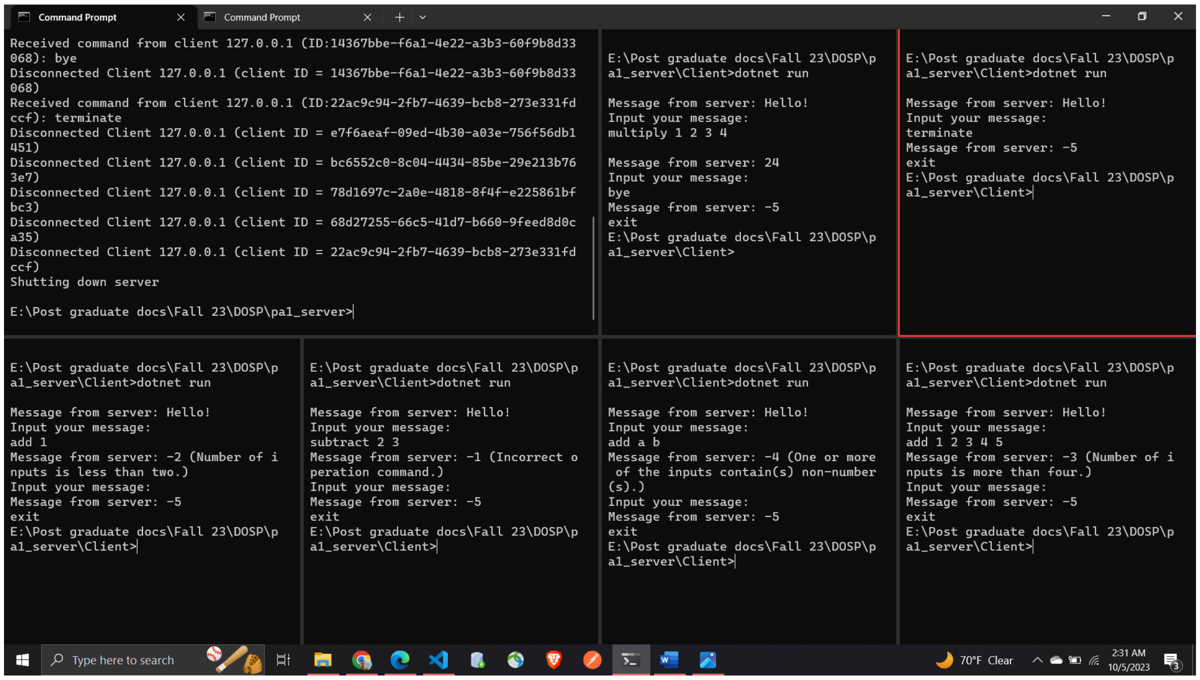
1. Execute multiple valid commands, followed by invalid inputs. Server returns appropriate results/error codes. We then send a “bye” command from the client which disconnects this client from the server and the client exits.



```
E:\Post graduate docs\Fall 23\DOSP\pa1_server>dotnet run
TCP Server started and listening on Port 9000
Client with ID 60b30e8b-9969-40ad-9fac-ebfede20bd4c connected from: 127.0.0.
1
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): add 4 5
Responded to client 127.0.0.1 with result: 9
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): subtract 10 5
Responded to client 127.0.0.1 with result: 5
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): add 4 5 6 7
Responded to client 127.0.0.1 with result: 22
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): multiply 2 3 4 5
Responded to client 127.0.0.1 with result: 120
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): add d 7
Responded to client 127.0.0.1 with result: -4
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): sub 4 2
Responded to client 127.0.0.1 with result: -1
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): add 5
Responded to client 127.0.0.1 with result: -2
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): add 5 6 7 8 9
Responded to client 127.0.0.1 with result: -3
Received command from client 127.0.0.1 (ID:60b30e8b-9969-40ad-9fac-ebfede20b
d4c): bye
Disconnected Client 127.0.0.1 (client ID = 60b30e8b-9969-40ad-9fac-ebfede20b
d4c)
```

```
E:\Post graduate docs\Fall 23\DOSP\pa1_server\Client>dotnet run

Message from server: Hello!
Input your message:
add 4 5

Message from server: 9
Input your message:
subtract 10 5

Message from server: 5
Input your message:
add 4 5 6 7

Message from server: 22
Input your message:
multiply 2 3 4 5

Message from server: 120
Input your message:
add d 7
Message from server: -4 (One or more of the inputs contain(s) non-number(s).
)
Input your message:
sub 4 2
Message from server: -1 (Incorrect operation command.)
Input your message:
add 5
Message from server: -2 (Number of inputs is less than two.)
Input your message:
add 5 6 7 8 9
Message from server: -3 (Number of inputs is more than four.)
Input your message:
bye
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\pa1_server\Client>
```

2. Handling multiple clients concurrently, shutting down some clients and then finally sending a terminate command from one client to shut down all the clients and also the server.



```
Received command from client 127.0.0.1 (ID:14367bbe-f6a1-4e22-a3b3-60f9b8d33
068): bye
Disconnected Client 127.0.0.1 (client ID = 14367bbe-f6a1-4e22-a3b3-60f9b8d33
068)
Received command from client 127.0.0.1 (ID:22ac9c94-2fb7-4639-bcb8-273e331fd
ccf): terminate
Disconnected Client 127.0.0.1 (client ID = e7f6aeaf-09ed-4b30-a03e-756f56db1
451)
Disconnected Client 127.0.0.1 (client ID = bc6552c0-8c04-4434-85be-29e213b76
3e7)
Disconnected Client 127.0.0.1 (client ID = 78d1697c-2a0e-4818-8f4f-e225861bf
bc3)
Disconnected Client 127.0.0.1 (client ID = 68d27255-66c5-41d7-b660-9feed8d0c
a35)
Disconnected Client 127.0.0.1 (client ID = 22ac9c94-2fb7-4639-bcb8-273e331fd
ccf)
Shutting down server

E:\Post graduate docs\Fall 23\DOSP\pa1_server>
```

```
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>dotnet run

Message from server: Hello!
Input your message:
multiply 1 2 3 4

Message from server: 24
Input your message:
bye
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>
```

```
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>dotnet run

Message from server: Hello!
Input your message:
terminate
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>
```

```
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>dotnet run

Message from server: Hello!
Input your message:
add 1
Message from server: -2 (Number of i
nputs is less than two.)
Input your message:
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>
```

```
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>dotnet run

Message from server: Hello!
Input your message:
subtract 2 3
Message from server: -1 (Incorrect o
peration command.)
Input your message:
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>
```

```
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>dotnet run

Message from server: Hello!
Input your message:
add a b
Message from server: -4 (One or more
 of the inputs contain(s) non-number
(s).)
Input your message:
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>
```

```
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>dotnet run

Message from server: Hello!
Input your message:
add 1 2 3 4 5
Message from server: -3 (Number of i
nputs is more than four.)
Input your message:
Message from server: -5
exit
E:\Post graduate docs\Fall 23\DOSP\p
a1_server\Client>
```

Results:

- The results of the application are as expected and per the specification. Multiple clients are handled asynchronously by the server process and appropriate responses/error codes are sent based on client inputs.
- The application scales well with an increasing number of client connections. Tested with up to 14 clients concurrently with no loss in performance or issues.

Scope for improvement:

- We can optimize our data structures (such as lists/arrays) which can prove to be bottlenecks at really large scales.
- Scaling can be more accurately measured by logging performance metrics such as latency and response time.
- If the client program starts executing before the server, it waits on the terminal indefinitely with no output printed. We can handle this edge case and improve user experience with some minor changes.

Contributions of project team members:

Name	***UFID***	***Contribution***
Nitish Chandra Mahesh	36139637	*Programmed the entire server code including base TCP server setup, socket communication, operation/terminate handling, error handling and asynchronous workflows.*
Anuj Papriwal	37008807	*Programmed the entire client code including base TCP setup, initialization, connection with server, handling user I/O, error code handling, bye/terminate handling.*
Pawan Kumar Jagadapuram	73643747	*Tested all commands, exceptions, errors and bye/terminate functionality with multiple clients. Developed the report.*
Gopi Amarnath Reddy Bekkem	72188579	*Tested all commands, exceptions, errors and bye/terminate functionality with multiple clients. Developed the report.*