

# COP5615- Distributed Operating System Principles Fall 2023 Programming Assignment - 1

## REPORT

TEAM - 26

NAME	UFID	EMAIL
VIJAY ABHINAV TELUKUNTA	86262606	vtelukunta@ufl.edu
HARSHITH MUNDADA	70221700	harshith.mundada@ufl.edu
INDU POTLA	84964176	potla.in@ufl.edu
SUCHITHRA MACHA	19116100	smacha@ufl.edu

# CONTENTS:

1. Project Description
  - a. Objectives
  - b. Specifications
2. How to Compile and Run the code
3. Description of Code Structure
4. Execution Results
5. Conclusion
6. Individual Contributions

# 1. PROJECT DESCRIPTION

## Objective:

The goal of the project is to create a concurrent client/server application that uses the TCP/IP protocol stack and socket communication. While emphasizing proper exception handling, it offers practical experience in socket programming. Clients will send arithmetic calculation requests to the server, which will return with the calculation results or the necessary error codes. The server application should be able to manage several client connections at once.

## Specifications:

### 1. Server-Side Program:

- The server-side program has been designed to operate continuously while it waits for new connections from clients.
- It must manage asynchronous operations to handle multiple customers' requests at once.

### 2. Client-Side Program:

- After the server has started running, clients are started on separate machines.
- The client application, which is written in F#, connects to the server and runs concurrently and asynchronously.

### 3. Error Handling:

- The server program is responsible for handling unexpected inputs and errors.

- Error codes (negative numbers) are used to represent various error scenarios, such as incorrect commands, insufficient or excessive inputs, or non-numeric inputs.
- The server program generates appropriate error codes for incorrect commands and communicates these codes to the client.
- Clients print corresponding error messages when receiving error codes.

#### 4. Termination:

- Both the server and the clients must terminate.
- There shouldn't be any lingering threads or processes after termination.

## 2. HOW TO COMPILE AND RUN THE CODE

- We are using Microsoft.NET.Sdk and net7.0 framework.
- Create a project directory and open the terminal at this path of the directory.
- In the project directory, type the command: **dotnet new console --language F# -o server** to create a server folder.
- In the project directory, type the command **dotnet new console --language F# -o client** to create a client folder.
- Upload the server.fs file that we provided, in the server folder created in the above steps and change the filename in server.fsproj file such that they are mapping to server.fs file i.e, Include="server.fs" as mentioned in below image.

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net7.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Compile Include="server.fs" />
  </ItemGroup>

</Project>

```

- Upload the client.fs file that we provided, in the client folder and change the filename in client.fsproj file as well such that they are mapping to client.fs file, i.e, Include="client.fs".

```

1  <Project Sdk="Microsoft.NET.Sdk">
2
3    <PropertyGroup>
4      <OutputType>Exe</OutputType>
5      <TargetFramework>net7.0</TargetFramework>
6    </PropertyGroup>
7
8    <ItemGroup>
9      <Compile Include="client.fs" />
10   </ItemGroup>
11
12 </Project>
13

```

- Open a terminal in the project directory that you created in the first step and navigate to server directory (cd server) and execute **dotnet build** command to compile the server project.
- Open another terminal in the project directory that you created in the first step and navigate to the client directory (cd client) and execute **dotnet build** command to compile the client project.
- Later in the server terminal(i.e terminal in server folder) execute **dotnet run** command to run the server. Open the client terminal(i.e terminal in client folder) and execute **dotnet run** command to run the client program.
- You can open as many client terminals as you require and use the **dotnet run** command in the same way as you did for the first client. Each client terminal acts as a new client using which you can send commands to the server and receive responses.
- Wait for “Hello!” message from server to the client whenever a new client-server connection is established using dotnet run command.
- After that, clients can send requests to the server and the server can respond.

```
C:\Users\munda\Documents\UFclasses\DOSP\projectserver>dotnet build
MSBuild version 17.7.3+8ec440e68 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
projectserver -> C:\Users\munda\Documents\UFclasses\DOSP\projectserver\bin\Debug\net7.0\projectserver.dll

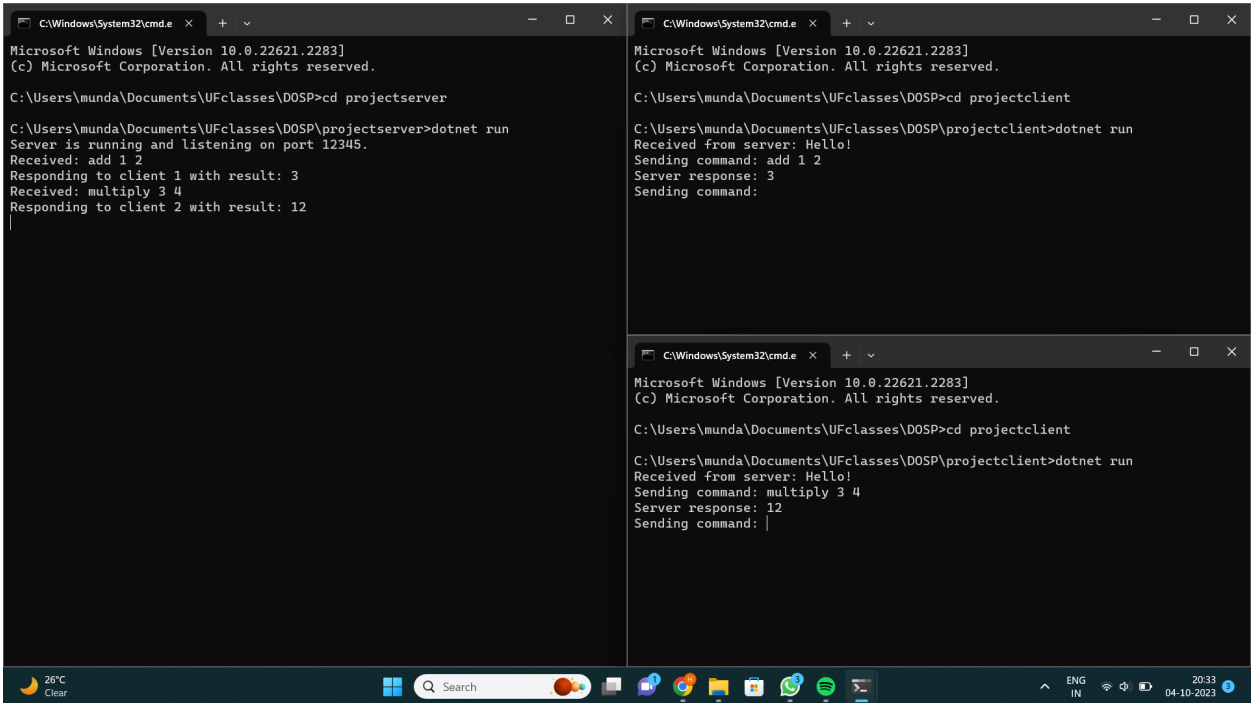
Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.09
```

```
C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet build
MSBuild version 17.7.3+8ec440e68 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
projectclient -> C:\Users\munda\Documents\UFclasses\DOSP\projectclient\bin\Debug\net7.0\projectclient.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.31
```



```
C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\munda\Documents\UFclasses\DOSP>cd projectserver

C:\Users\munda\Documents\UFclasses\DOSP\projectserver>dotnet run
Server is running and listening on port 12345.
Received: add 1 2
Responding to client 1 with result: 3
Received: multiply 3 4
Responding to client 2 with result: 12
|

C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\munda\Documents\UFclasses\DOSP>cd projectclient

C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet run
Received from server: Hello!
Sending command: add 1 2
Server response: 3
Sending command:

C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\munda\Documents\UFclasses\DOSP>cd projectclient

C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet run
Received from server: Hello!
Sending command: multiply 3 4
Server response: 12
Sending command: |
```

In the above screenshots, projectclient and projectserver are the project folders for client and server respectively. These project folders contain the source code files client.fs and server.fs and also fsproj files. Navigate to the projects and run the **dotnet run** command to execute the client and server programs. By doing this process a client-server connection will be established and you can send commands through client to server and receive the responses in the terminal.

### 3. DESCRIPTION OF CODE STRUCTURE

The code is structured to handle multiple clients concurrently, with each client communication managed in its thread. It demonstrates proper error handling, response generation, and termination procedures, making it suitable for a socket communication project in F#.

#### Server-Side:

##### 1. Imports:

- The code begins with several import statements, importing necessary namespaces for socket communication, input/output operations, and threading.

##### 2. Configuration:

- `let port = 12345`: Defines the port number on which the server listens for incoming connections.
- `let mutable clientCounter = 0`: Initializes a mutable counter to keep track of connected clients.
- `let mutable clientSockets = new System.Collections.Concurrent.ConcurrentBag<TcpClient>()`: Creates a concurrent bag to store client sockets.

##### 3. 'handleClient' Function:

- This function handles communication with individual clients.
- It accepts a 'TcpClient' instance and a 'TcpListener' instance.
- Inside the function, it:
  - Sets up stream readers and writers for communication.
  - Send a "Hello!" message to the client upon connection.



- Loops to continuously read and process client requests.
- Handles various client commands, such as arithmetic calculations ("add," "subtract," "multiply"), "bye" to gracefully exit, and "terminate" to exit all clients and the server.
- Provides proper error handling and response generation.
- Closes the client socket when done.

#### 4. 'startServer' Function:

- This function initializes and starts the server.
- It binds the server to the IP address "127.0.0.1" and the predefined port.
- It starts listening for incoming connections.
- Clients are handled in separate threads, allowing for concurrent connections.

#### 5. 'main' Function (Entry Point):

- The 'main' function is the entry point of the program.
- It starts the server in a separate thread using 'startServer()'.

### Client-Side:

#### 1. Imports:

- The code begins with import statements, including the necessary namespaces for socket communication, input/output operations.

#### 2. Configuration:

- 'let serverAddress = "127.0.0.1"': Specifies the IP address of the server to connect to.
- 'let serverPort = 12345': Defines the port number on which the server is listening.

#### 3. ConnectToServer Function:

- This function handles the client-side communication with the server.
- Inside the function, it:
- Creates a `TcpClient` instance and connects it to the specified server address and port.
- Sets up stream readers and writers for communication.
- Reads and prints the initial "Hello" message from the server.
- Loops to continuously read user input, send it to the server, and receive responses.
- Handles server responses, including error codes (-1 to -5) and normal responses.
- Implements a mechanism to exit the loop and the client program gracefully when instructed by the server (e.g., when receiving "-5").

#### 4. 'Main Function' (Entry Point):

- The `main` function is the entry point of the program.
- It calls the `connectToServer()` function to initiate the client-server communication.

The code is structured to allow the client to connect to a server, exchange messages, and handle server responses effectively. It demonstrates proper handling of error codes and graceful termination procedures, making it suitable for a client program in a socket communication project using F#.

## 4. DESCRIPTION & EXECUTION OF RESULTS

Now here we will see some the sample Test cases Execution:  
First Server program starts running and continuously listens to the port 12345

### **TEST CASE 1 :**

- Client 1: Sent a command add 5 8 to the server
- Result: Received a result 13 from the server

### **TEST CASE 2 :**

- Client 2: Sent a command subtract 20 7 to the server
- Result: Received a result 13 from the server

### **TEST CASE 3 :**

- Client 3: Sent a command add d 8 to the server
- Result: Received a result -4 Exception(one or more inputs in the command are not integers)

### **TEST CASE 4 :**

- Client 1: Sent a command bye to the server
- Result: Received a result -5 from the server(Client 1 exits the port removes it's connection with the server)

### **TEST CASE 5 :**

- Client 4: Sent a command multiply 2 3 4 to the server
- Result: Received a result 24 from the server

### **TEST CASE 6 :**

- Client 5: Sent a command add 5 6 7 8 to the server
- Result: Received a result 26 from the server

### TEST CASE 7 :

- Client 5: Sent a command terminate to the server
- Result: Received a result -5 from the server(All the Clients and Server exits after terminate command)

### SERVER RESPONSE:

Server response for above test-cases is:

```
C:\Users\munda\Documents\UFclasses\DOSP\projectserver>dotnet run
Server is running and listening on port 12345.
Received: add 5 8
Responding to client 1 with result: 13
Received: subtract 20 7
Responding to client 2 with result: 13
Received: add d 7
Responding to client 3 with result: -4
Received: bye
Responding to client 1 with result: -5
Received: multiply 2 3 4
Responding to client 4 with result: 24
Received: add 5 6 7 8
Responding to client 5 with result: 26
Received: terminate
Responding to client 5 with result: -5

C:\Users\munda\Documents\UFclasses\DOSP\projectserver>|
```

Clients response for above test cases is as follows:

Client 1:

<pre>C:\Windows\System32\cmd.e x + v Microsoft Windows [Version 10.0.22621.2283] (c) Microsoft Corporation. All rights reserved.  C:\Users\munda\Documents\UFclasses\DOSP&gt;cd projectserver  C:\Users\munda\Documents\UFclasses\DOSP\projectserver&gt;dotnet run Server is running and listening on port 12345. Received: add 5 8 Responding to client 1 with result: 13 Received: subtract 20 7 Responding to client 2 with result: 13 Received: add d 7 Responding to client 3 with result: -4 Received: bye Responding to client 1 with result: -5  </pre>	<pre>C:\Windows\System32\cmd.e x + v Microsoft Windows [Version 10.0.22621.2283] (c) Microsoft Corporation. All rights reserved.  C:\Users\munda\Documents\UFclasses\DOSP&gt;cd projectclient  C:\Users\munda\Documents\UFclasses\DOSP\projectclient&gt;dotnet run Received from server: Hello! Sending command: add 5 8 Server response: 13 Sending command: bye exit  C:\Users\munda\Documents\UFclasses\DOSP\projectclient&gt;</pre>
---	---

Client 2:

```
C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet
run
Received from server: Hello!
Sending command: subtract 20 7
Server response: 13
Sending command: |
```

Client 3:

```
C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet run
Received from server: Hello!
Sending command: add d 7
Server response: one or more of the inputs contain(s) non-number(s)
Sending command: |
```

Client 4:

```
C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet run
Received from server: Hello!
Sending command: multiply 2 3 4
Server response: 24
Sending command: |
```

Client 5:

```
C:\Users\munda\Documents\UFclasses\DOSP\projectclient>dotnet run
Received from server: Hello!
Sending command: add 5 6 7 8
Server response: 26
Sending command: |
```

### TEST CASE 8 :

- Client 1: Sent a command adp 5 8 to the server
- Result: Received a result -1 from the server
- Response to user: incorrect operation command

### TEST CASE 9 :

- Client 1: Sent a command add 1 to the server
- Result: Received a result -2 from the server
- Response to user: number of inputs is less than two.

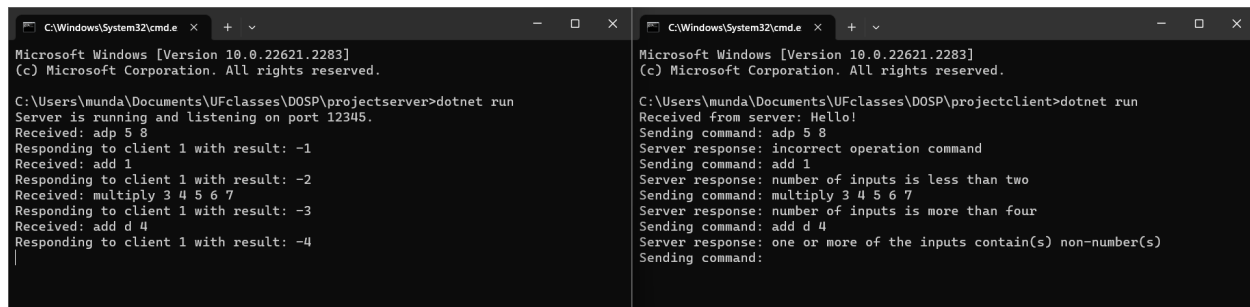
### TEST CASE 10 :

- Client 1: Sent a command multiply 3 4 5 6 7 to the server
- Result: Received a result -3 from the server
- Response to user : number of inputs is more than four

### TEST CASE 10 :

- Client 1: Sent a command add d 4 to the server
- Result: Received a result -4 from the server
- Response to user : one or more of the inputs contain(s) non-number(s)

The test-case results for above test-cases can be depicted in below image:



```
C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\munda\Documents\UFclasses\DO5P\projectserver>dotnet run
Server is running and listening on port 12345.
Received: adp 5 8
Responding to client 1 with result: -1
Received: add 1
Responding to client 1 with result: -2
Received: multiply 3 4 5 6 7
Responding to client 1 with result: -3
Received: add d 4
Responding to client 1 with result: -4

C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\munda\Documents\UFclasses\DO5P\projectclient>dotnet run
Received from server: Hello!
Sending command: adp 5 8
Server response: incorrect operation command
Sending command: add 1
Server response: number of inputs is less than two
Sending command: multiply 3 4 5 6 7
Server response: number of inputs is more than four
Sending command: add d 4
Server response: one or more of the inputs contain(s) non-number(s)
Sending command:
```

## 5. CONCLUSION

This project uses the F# programming language to provide concurrent socket communication. The project aims to create a server with the ability to effectively manage multiple client connections by placing a major emphasis on consistent and error-resistant responses, robust exception handling and termination methods. Additionally, this project introduces the idea of asynchronous client-server interaction, allowing clients to communicate with the server simultaneously.

## 6. INDIVIDUAL CONTRIBUTIONS

### **HARSHITH MUNDADA:**

- I worked on client program, server program, testing and attached screenshots of test implementation and results.
- I have implemented the client and server code.
- I have designed the `handleClient` function, `startServer` function in the server program that plays a vital role in client handling and server listening. It is designed in such a way that it can handle any number of clients simultaneously, accept multiple requests from various clients and send responses to respective clients.
- I have designed the `connectToServer` function in the client program that plays a major role in sending requests to the server and getting the response.

- I have designed the client and server program such that the user is completely aware of the responses that are acquired from the server.
- I have done thorough testing of the code and exception handling using various inputs and analyzing the output for each input.
- I have implemented and pasted the pictures of screenshot results of all the possible test cases.
- I have contributed in the description of steps to compile and run the program and have attached screenshots to ensure easy understanding of the process.
- I have verified the report and all of its sections and ensured that it is properly structured.

#### **VIJAY ABHINAV TELUKUNTA:**

- Wrote code to handle various exception scenarios (e.g., -1, -2, ..., -5) and terminate/bye conditions.
- Implemented the decoding of client requests, mapping them to appropriate operations (add, subtract, multiply, etc.) and performing addition, subtraction or multiplication accordingly.
- Implemented client-side mapping of error codes to their corresponding error messages, enhancing user understanding of errors.
- Implemented the closing of all sockets when a terminate command is issued by a client terminal.
- Ensured a graceful termination of the server, stopping the listener and closing connections properly.
- Conducted code refactoring for improved code structure and readability.
- Added comments throughout the codebase to enhance understanding and maintainability.
- Conducted thorough testing of the program with various scenarios to ensure robustness and reliability.
- Included clear instructions on how to compile and run the project in the report, improving project documentation.



**SUCHITHRA MACHA:**

- My roles and responsibilities in the project include working for the server program, testing the program using various test cases and working on the report.
- Worked on handling exception case scenarios where we will get an error code
- Worked on mapping of error codes to their relevant error messages.
- Worked on Testing all Scenarios to know if we are meeting client requirement and checked for any kind of bugs in the code
- Worked on refactoring and redesigning the report and the code to meet the exact requirements of the client.
- Provided information in the report about imports, setup, and important functions like "handleClient","startServer" and "Main" to help with the server-side code structure's details.
- Providing a comprehensive description and systematically executing the test results in the report.
- Ensured that the report includes test cases that comprehensively cover all the exceptions present in the code.

**INDU POTLA:**

- Contributed in developing the code for a few scenarios, conducting testing, and creating comprehensive documentation.
- Contributed in testing which ensured the project's reliability and functionality.
- Involved in building the scenarios involving error messages from the client. This helped enhance the system's resilience against unexpected input.
- Helped in building the termination case scenarios, where the system handled the client attempts to access the server after the socket connection was terminated.

- I provided a detailed project description that outlined the project's primary objectives and included detailed specifications.
- Contributed in detailing the client-side code structure by providing insights on the imports, configuration and the key functions used like "ConnectToServer" and "Main"
- Documented the server-side code structure by emphasizing the functions "handleClient", "startServer" and "Main".