****

**UNIVERSITY OF NAIROBI**

**FACULTY OF SCIENCE AND TECHNOLOGY**

**DEPARTMENT OF COMPUTING AND INFORMATICS**

CSC 322: NETWORK & DISTRIBUTED PROGRAMMING

**GROUP WORK**

**QUESTION 4**

**GROUP 4**

P15/141298/2020 AUSTIN ONGWAE

P15/139806/2020 CLIFF EZRA

P15/140390/2020 TED MWAI

P15/140597/2020 AJAK ELIJAH

P15/137741/2020 THON PETER

# Question 4

**Convert the two programs (i) class registration and (ii) calculator into distributed programs that use the RPC paradigm. Distributed programming and RPC to be covered in class. You are to develop your own client and server stubs.**

# Student Registration

This example demonstrates a Student Registration system using Remote Procedure Call (RPC). The system consists of a client program and a server program. The client program allows the user to enter student details such as serial number, registration number, and name. It then sends the student information to the server using RPC. The server program receives the student details, checks if the student already exists in a file, and adds the student to the file if they do not exist. The server program returns a status indicating the success or failure of the registration process.

**WE** broke down the implementation into different components:

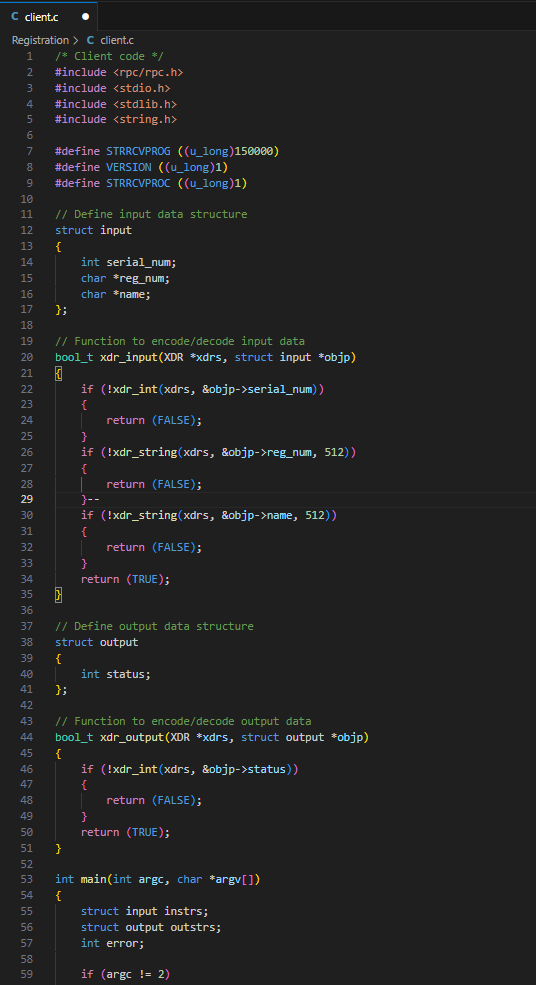
**Client Program (‘client.c’)**

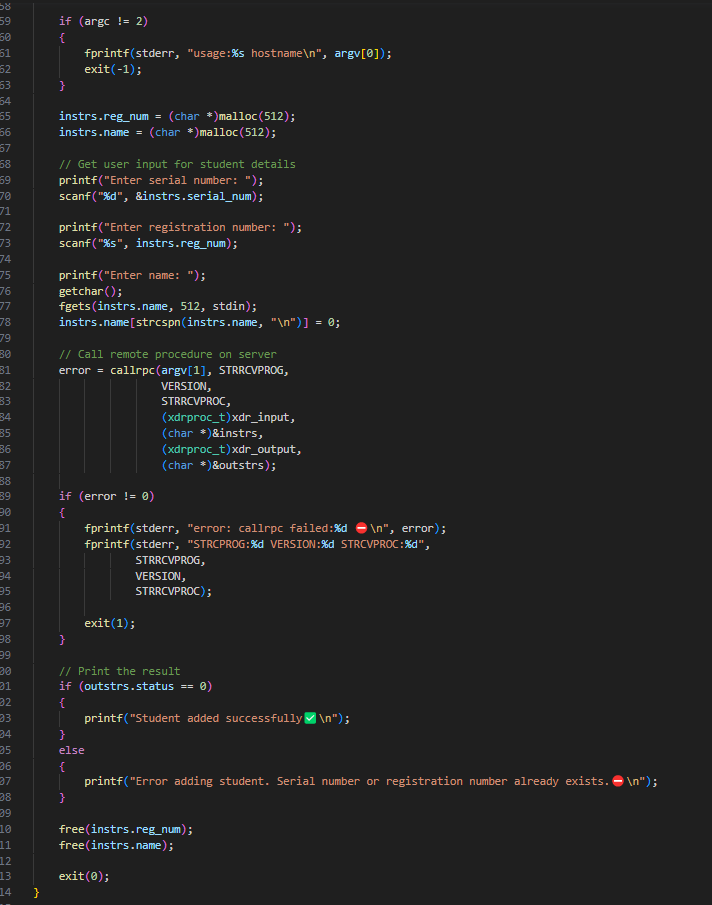
The client program performs the following steps:

1. It includes the necessary header files for **RPC**, standard input/output, and string manipulation.
2. It defines constants for the RPC program number, version number, and procedure number.
3. It defines the input data structure struct input, which contains the student details.
4. It defines the function **xdr\_input** to encode and decode the input data structure.
5. It defines the output data structure struct output, which contains the registration status.
6. It defines the function **xdr\_output** to encode and decode the output data structure.
7. The main function prompts the user to enter the student details (serial number, registration number, and name).
8. It calls the **callrpc** function to send the student details to the server using RPC.
9. It checks the return value of **callrpc** to handle any errors that may occur during the RPC call.
10. It prints the registration status based on the received output.
11. It frees the dynamically allocated memory for the student details.

Finally, it exits the program.

**client.c**

****

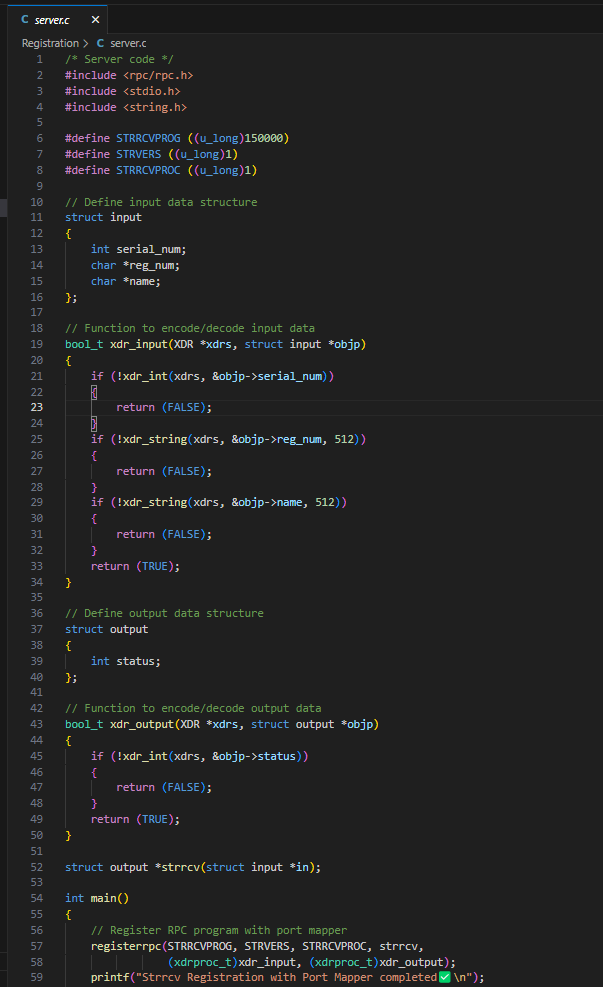
****

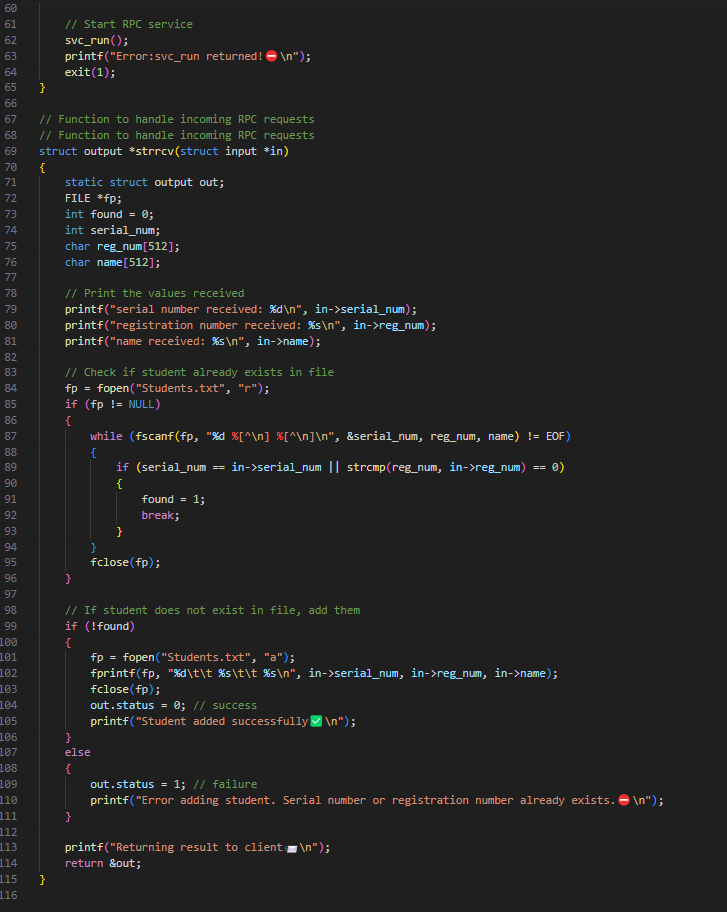
**Server Program (‘server.c’)**

The server program performs the following steps:

1. It includes the necessary header files for RPC and standard input/output.
2. It defines constants for the RPC program number, version number, and procedure number.
3. It defines the input data structure struct input, which contains the student details.
4. It defines the function **xdr\_input** to encode and decode the input data structure.
5. It defines the output data structure struct output, which contains the registration status.
6. The main function registers the RPC program with the port mapper using the **registerrpc** function.
7. It enters the RPC service loop using the **svc\_run** function.
8. When an RPC request is received, it calls the **strrcv** function.
9. The **strrcv** function checks if the student already exists in the file by reading from the file and comparing the serial number and registration number.
10. If the student does not exist, it appends the student details to the file.
11. It sets the registration status in the output data structure based on the success or failure of the registration process.
12. Finally, it returns the output data structure to the client.

**server.c**

****



**Compilation and Execution**

To compile the client and server programs, you can use the provided **Makefile.** Simply run the following command:



This command will compile both the client and server programs.

To run the server program, execute the following command:

****

The server will start and register with the port mapper.

To run the client program, execute the following command, providing the server's hostname as an argument:



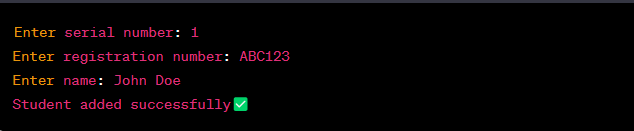
Replace <**hostname**> with the actual hostname or IP address of the server.

**Output**

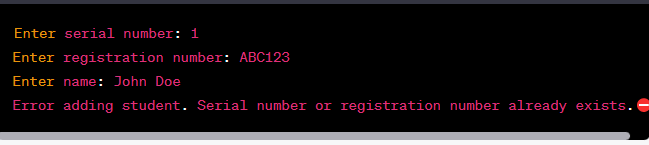
Upon running the client program, it will prompt the user to enter the student details: serial number, registration number, and name.

**Output:**

1. **Successful registration:**

****

1. **Error in registration (existing student):**

****

# Calculator

**Explanation**

The Calculator program demonstrates the implementation of an Iterative Connectionless server and client using a data structure for input data. The client program accepts user input for two numbers and an operation (addition, subtraction, multiplication, or division) to perform on those numbers. It then sends the input data to the server using remote procedure call (RPC). The server receives the input, performs the requested operation, and sends the result back to the client

## Our Implementation

This example demonstrates an Iterative Connectionless server and client. The server and client communicate using a data structure for input data. The server implements a registration process with the port mapper to handle incoming requests. The client program prompts the user to enter two numbers and select an operation. It then sends the input data to the server using remote procedure call (RPC). The server receives the input, performs the requested operation, and sends the result back to the client. Finally, the client program displays the received result.

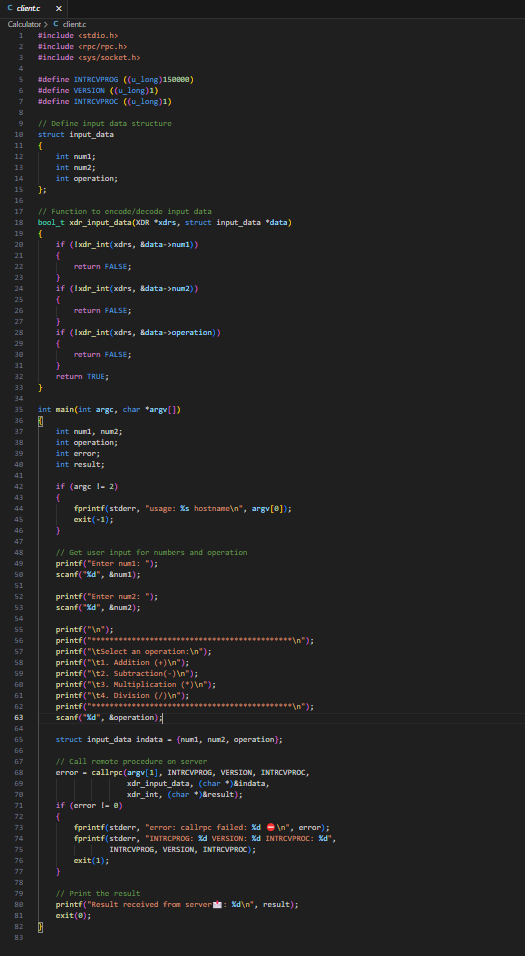
**The implementation follows the following steps:**

1. The client program prompts the user to enter ‘**num1’** and ‘**num2’** as the numbers for the operation.
2. The client program displays a menu of operations and prompts the user to select one.
3. The client program creates an ‘**input\_data**’ structure with the entered values and the selected operation.
4. The client program calls the ‘**callrpc’** function to send the input data to the server using RPC.
5. The server program registers the RPC program with the port mapper using the ‘**registerrpc’** function.
6. The server program enters the RPC service loop using the **‘svc\_run’** function.
7. When the server receives an RPC request, it calls the ‘**intrcv’** function.
8. The ‘**intrcv’** function performs the requested operation on the received input data.
9. The ‘**intrcv’** function returns the result to the client.
10. The client program receives the result from the server and displays it to the user.

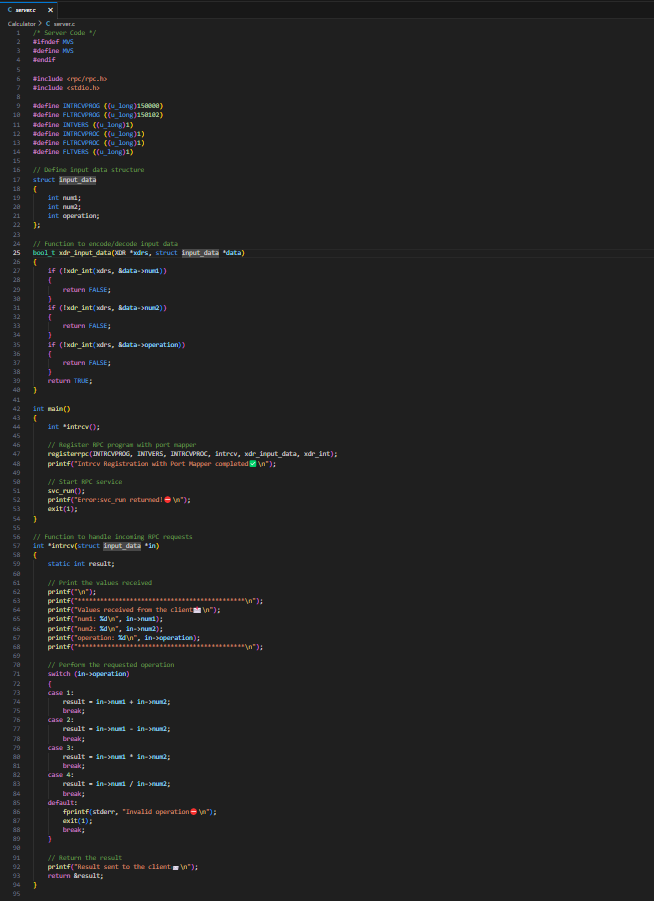
This implementation allows multiple clients to connect to the server in an iterative manner, with each client receiving a response based on its individual request.

## Code

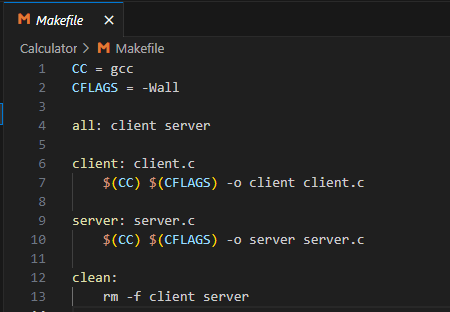
server.cpp



client.cpp



Makefile



**Compilation**

To compile the server program, use the following command:



**Output**

To run the programs, execute the following commands in separate terminal windows:



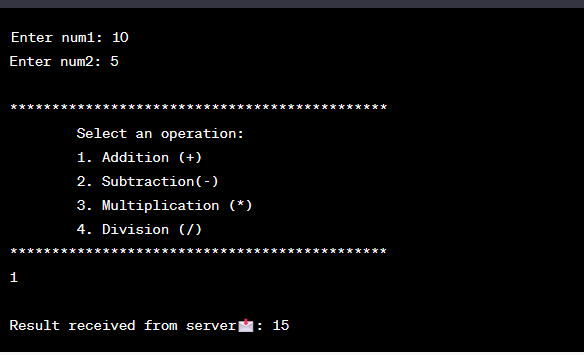


Make sure to replace <hostname> with the actual hostname or IP address of the server.

The server program will register the RPC program with the port mapper and start listening for incoming requests. The client program will prompt you to enter two numbers and select an operation. It will then send the input data to the server using RPC. The server will receive the input, perform the requested operation, and send the result back to the client. The client program will display the received result.

Here is what you will see:

**Client output**

****

**Server output**

****

When the client program is executed, it prompts the user to enter **num1** and **num2**, and then select an operation. In this example, the user enters **10** for **num1**, **5** for **num2**, and selects addition (operation code 1). The client sends the input data to the server using **RPC**.

The server receives the input and performs the addition operation (**10 + 5**). It then prints the received values (**num1**, **num2**, and operation) and sends the result back to the client. The client program displays the received result, which in this case is **15**.

# REFERNCES

1. [Iterative, Concurrent, Connectionless and Connection-oriented Servers](https://www3.nd.edu/~cpoellab/teaching/cse354/sp9.pdf)
2. "RPC Programming Guide" - The Open Group. [Online]. Available: https://pubs.opengroup.org/onlinepubs/9696989899/. [Accessed: May 21, 2023].
3. "RFC 1057 - Remote Procedure Call Protocol Specification Version 2" - Sun Microsystems. [Online]. Available: https://tools.ietf.org/html/rfc1057. [Accessed: May 21, 2023].
4. "The GNU C Library" - Free Software Foundation. [Online]. Available: https://www.gnu.org/software/libc/. [Accessed: May 21, 2023].
5. "gcc(1) - Linux manual page" - Linux Programmer's Manual. [Online]. Available: https://man7.org/linux/man-pages/man1/gcc.1.html. [Accessed: May 21, 2023].
6. "make(1) - Linux manual page" - Linux Programmer's Manual. [Online]. Available: https://man7.org/linux/man-pages/man1/make.1.html. [Accessed: May 21, 2023].