

## **UNIVERSITY OF NAIROBI**

**FACULTY OF SCIENCE AND TECHNOLOGY**

**DEPARTMENT OF COMPUTING AND INFORMATICS**

CSC 322: NETWORK & DISTRIBUTED PROGRAMMING

**GROUP WORK**

## 

**GROUP MEMBERS.**

**WEKULO FIONA ABIGAEL - P15/1905/2020**

**IGADWA DEBI TELOORI - P15/1906/2020**

**MWANGI ENOCH BABU - P15/140432/2020**

**KARANJA SAMUEL MWANGI - P15/1933/2020**

**EMMANUEL MBITHI MUTHIANI - P15/1894/2020**

**Assignment 2**

Develop a client server program which implements a class registration platform. The program should allow the user to register themselves by entering their details which include the student registration number and names. The program should maintain a list of registered students in a text file with the following format. No double registrations,

| S/No Registration No. Names |
| --- |
| 1 P15/1254/2020 Joyce Wanjala  2 P15/23675/2020 Mosses Kiprotich  etc |

### 1.Iterative Connectionless

This is a simple C program for adding unique serial numbers, registration numbers, and names to a file. It uses a UDP-based socket communication protocol to interact between the server and the client. The client can enter a serial number, registration number, and name, which is sent to the server. The server then checks if the serial number and registration number already exist in a file. If not, it adds them to the file and sends a confirmation message to the client.

The server-side code creates a server-socket using socket() with the AF\_INET domain and SOCK\_DGRAM type, which specifies that the socket will use UDP for data transmission. It then binds the socket to a port number using bind() and listens for incoming data from the client. When the server receives a message, it processes it by checking if the serial number and registration number already exist in the file using fopen(), fgets(), and strstr(). If either of them does not exist in the file, it appends them to the file using fopen() and fprintf(). Finally, it sends a confirmation message back to the client using sendto().

The client-side code creates a client-socket using socket() with the AF\_INET domain and SOCK\_DGRAM type, which specifies that the socket will use UDP for data transmission. It then specifies the server's address using sockaddr\_in struct and inet\_addr(). It then prompts the user to enter a serial number. If the user enters "exit", the program will exit. Otherwise, it sends the serial number to the server using sendto(). The client then waits for a confirmation message from the server using recvfrom(). If the server has added the serial number successfully, it prompts the user to enter a registration number. If the registration number does not already exist in the file, the server adds it to the file and sends a confirmation message back to the client. Finally, the client prompts the user to enter a name, which is added to the file if it is unique.

Overall, this is a simple program that demonstrates how to use UDP-based sockets for client-server communication in C programming.

### 2.Iterative Connection oriented

The program above is a simple client-server application that uses TCP/IP to communicate between the client and the server. The client sends commands to the server and the server responds with data or status messages. The program supports two commands: "view" and "register".

The application layer protocol used in this program is a custom protocol that has been designed specifically for this application. The protocol consists of a set of commands and responses that are exchanged between the client and the server.

The commands and responses are defined as follows:

1. "view" command: This command is used by the client to request the server to view the contents of the database file. When the server receives this command, it reads the contents of the database file and sends it back to the client as a response.
2. "register" command: This command is used by the client to register a new entry in the database file. The client sends the registration data to the server, and the server verifies that the registration number is not already in use. If the registration number is not in use, the server adds the new entry to the database file and sends a "Registration complete!" response to the client. If the registration number is in use, the server sends a "Registration number in use!" response to the client.
3. "unknown command" response: This response is sent by the server to the client if the server receives a command that is not recognized.

The data that is exchanged between the client and the server is in the form of a struct called "info", which contains three fields: "name", "reg", and "serial". The "name" field is a string that represents the name of the person registering a new entry in the database. The "reg" field is a string that represents the registration number of the vehicle. The "serial" field is a string that represents the serial number of the vehicle.

In summary, the application layer protocol used in this program is a custom protocol that is designed specifically for this client-server application. The protocol consists of a set of commands and responses that are exchanged between the client and the server. The data is exchanged in the form of a struct called "info", which contains the registration data

### 3.Concurrent Connectionless

This is a UDP server that listens for incoming data from clients on port 3000. It receives data from clients and processes it to add a serial number, registration number, and name to a text file. The server runs an infinite loop waiting for incoming requests, and once a request is received, it creates a new thread to handle that request.

The handle\_client function takes in a pointer to a struct ClientData that contains information about the server socket, the name of the file to write to, the client's address, and the size of the client's address. It then enters an infinite loop waiting for incoming data from the client. Once data is received, it processes it and adds the appropriate information to the text file. If the client sends "exit" as the data, the loop breaks and the thread terminates.

The main function creates the server-socket, binds it to port 3000, and starts listening for incoming requests. Once a request is received, it creates a new thread to handle that request using the handle\_client function. The main function then goes back to listening for incoming requests.

### 4.Concurrent Connection oriented

The program implements a simple client-server model where the server listens for incoming connections and the clients connect to the server. The server listens on port 3000 and accepts one client connection at a time. Once a connection is established, the server spawns a new thread to handle requests from that client.

The client sends three pieces of data to the server: a serial number, a registration number, and a name. The server receives the data, checks if the serial and registration numbers already exist in a file named "concurrent\_connection.txt". If they do not exist, the server writes the data to the file and sends a confirmation message back to the client. If they do exist, the server sends an error message back to the client. The server also writes the name to the file regardless of whether it already exists.

The client can exit the connection by sending the string "exit" to the server.

**Assignment 2**

Develop a client server program which implements a simple calculator. The calculator can perform the following operations (+, -, \*, /) chosen by the user on any two integers supplied by the user.

1.Iterative Connectionless

This program implements a simple client-server model of a calculator using UDP sockets.

The server listens on a specific port and waits for requests from clients. It accepts a message from a client, which includes two operands and an operator to perform arithmetic operations. The server then performs the requested operation and sends the result back to the client.

The client prompts the user to enter two numbers and select an operator from a menu. It then sends the message to the server, which performs the calculation and sends the result back to the client. The client displays the result to the user and prompts the user to perform another operation or exit.

The communication between the client and the server uses the User Datagram Protocol (UDP), which is a lightweight protocol that does not guarantee reliable delivery of messages. Therefore, this program is suitable for applications where speed is more important than reliability, such as video conferencing or online gaming.

2.Iterative Connection oriented

The given code implements a simple calculator that allows the client to send two numbers and an operator to the server, and the server returns the result of the calculation to the client. The client and server communicate using sockets and the TCP protocol.

The server listens on port 4444 and waits for incoming connections. When a client connects, the server logs a message, reads the message from the client, parses the message to extract the two numbers and the operator, performs the calculation, and sends the result back to the client. The server continues to listen for messages until the client disconnects.

The client connects to the server on port 4444, prompts the user to enter two numbers and select an operator, sends the message to the server, and waits for the result. When the client receives the result, it displays the result to the user and prompts the user to enter two numbers again. The client continues to send messages to the server until the user enters "q" to quit.

3.Concurrent Connectionless

This is a program written in C++ that implements a simple client-server model for performing basic arithmetic operations.

The server waits for incoming connections on port 4444 and creates a thread to handle each incoming client connection. The server reads messages from the client containing two numbers and an operator (+, -, \*, or /) and sends back the result of the arithmetic operation.

The client connects to the server on port 4444 and prompts the user to enter two numbers and select an arithmetic operation. The client sends the message containing the numbers and operator to the server, reads the response, and displays it to the user.

This program uses the POSIX thread library to handle multiple clients concurrently.

Note that this code is missing the code for handling the user's choice of arithmetic operator in the client code, and also does not have any error checking, like checking if the user has entered a valid operator or dividing by zero.

4.Concurrent Connection oriented

The program implements a simple client-server application where the client sends two numbers and an operator to the server, and the server performs the specified arithmetic operation and returns the result to the client. The communication is done over TCP/IP sockets.

The server program creates a socket using socket() and binds it to an IP address and port using bind(). It then listens for incoming connections using listen(), and for each incoming connection, it accepts it using accept() and creates a new thread to handle the communication with that client. The new thread reads messages from the client using recv(), parses the message to extract the two numbers and the operator, performs the specified arithmetic operation, and sends the result back to the client using send(). If any error occurs during the communication, the thread terminates and the client socket is closed.

The client program creates a socket using socket() and connects it to the server using connect(). It then prompts the user to enter two numbers and select an operator, and sends the numbers and operator to the server using send(). It then waits for the server to send the result using recv() and displays the result to the user. If any error occurs during the communication, the client socket is closed and the program terminates.