

## **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 |

<https://github.com/swafey-karanja/C-programming./tree/main/Question%202>

# 1.CONCURRENT CONNECTION-ORIENTED.

This code implements a simple client-server model where the server listens on port 3000 and accepts connections from multiple clients. The clients can send serial numbers, registration numbers, and names, which are stored in a file named "concurrent\_connection.txt".

The main() function in the server code does the following:

1. Creates a server-socket and binds it to port 3000.
2. Starts listening for requests from clients.
3. When a request arrives, it accepts the connection, creates a new thread to handle the client, and passes the client's socket and filename to the thread.
4. The thread receives data from the client, checks if the serial number or registration number already exists in the file, and writes them to the file if they don't exist. Then it sends a response to the client.
5. When the client sends "exit", the thread breaks out of the loop and closes the connection with the client.

The main() function in the client code does the following:

1. Creates a client-socket and connects it to the server on port 3000.
2. Sends a serial number to the server, receives a response from the server, and prints it.
3. Sends a registration number to the server, receives a response from the server, and prints it.
4. Sends a name to the server, receives a response from the server, and prints it.
5. Sends "exit" to the server to close the connection.

Overall, this code is a simple example of a client-server model with multiple clients and a shared resource (the file "concurrent\_connection.txt"). The use of threads allows the server to handle multiple clients concurrently. However, the code has some potential issues, such as race conditions when multiple threads try to write to the file simultaneously, lack of error handling, and lack of proper synchronization between threads.

Conceptual Server Algorithm.

The server algorithm for this program is as follows:

1. Create a server-socket (listener) and bind it to a port (in this case, port 3000).
2. Start listening for requests using the listen() function.
3. When a request arrives, create another (normal) socket and build a TCP connection with the client using the accept() function.
4. Create a new thread to handle this client.
5. In the new thread, receive a request from the client using the recv() function.
6. If the request is to exit, close the connection with the client using the close() function and terminate the thread.
7. Otherwise, process the request as follows:
   * Read the serial number from the request and check if it already exists in the file.
   * If it doesn't exist, add it to the file and send a success message to the client using the send() function.
   * If it already exists, send an error message to the client using the send() function and continue to the next step.
   * Read the registration number from the request and check if it already exists in the file.
   * If it doesn't exist, add it to the file and send a success message to the client.
   * If it already exists, send an error message to the client and continue to the next step.
   * Read the name from the request and add it to the file.
   * Send a success message to the client using the send() function.
8. Go back to step 5 and wait for the next request from the client.
9. When the client disconnects, close the connection with the client using the close() function and terminate the thread.
10. Go back to step 3 and wait for the next request from a client.

Code.

*Server-side.*

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <pthread.h>

struct ClientData {

int client\_socket;

std::string filename;

ClientData(int client\_socket, std::string filename)

: client\_socket(client\_socket), filename(filename) {}

};

void\* handle\_client(void\* arg) {

ClientData\* client\_data = (ClientData\*)arg;

int client\_socket = client\_data->client\_socket;

std::string filename = client\_data->filename;

std::string serial\_number;

std::string registration\_number;

std::string name;

// Handle multiple requests from this client

while (true) {

// Receive request from client

char buffer[1024];

int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

}

// Check if the client wants to exit

if (std::string(buffer, bytes\_received) == "exit") {

break;

}

// Process request

serial\_number = std::string(buffer, bytes\_received);

bool exists = false;

std::ifstream infile(filename);

std::string line;

while (getline(infile, line)) {

if (line == serial\_number) {

exists = true;

break;

}

}

infile.close();

if (!exists) {

std::ofstream outfile(filename, std::ios\_base::app);

outfile << serial\_number << "\n";

outfile.close();

send(client\_socket, "Serial number added.\n", 20, 0);

} else {

send(client\_socket, "Serial number already exists.\n", 29, 0);

continue;

}

bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

}

registration\_number = std::string(buffer, bytes\_received);

exists = false;

infile.open(filename);

while (getline(infile, line)) {

if (line == registration\_number) {

exists = true;

break;

}

}

infile.close();

if (!exists) {

std::ofstream outfile(filename, std::ios\_base::app);

outfile << registration\_number << "\n";

outfile.close();

send(client\_socket, "Registration number added.\n", 25, 0);

} else {

send(client\_socket, "Registration number already exists.\n", 34, 0);

}

bytes\_received = recv(client\_socket, buffer,sizeof(buffer),0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

}

name = std::string(buffer, bytes\_received);

std::ofstream outfile(filename, std::ios\_base::app);

outfile << name << "\n";

outfile.close();

send(client\_socket, "Name added.\n", 11, 0);

}

// Close connection with client

close(client\_socket);

delete client\_data;

return nullptr;

}

// Serial Number Program

int main() {

std::string filename = "concurrent\_connection.txt";

// Create a server-socket (listener) and bind it to a port (higher than 1024)

int server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

std::cerr << "Failed to create server socket\n";

return 1;

}

sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(3000);

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(server\_socket, (sockaddr\*)&server\_address, sizeof(server\_address)) == -1) {

std::cerr << "Failed to bind server socket\n";

return 1;

}

// Start listening for requests

if (listen(server\_socket, 1) == -1) {

std::cerr << "Failed to listen on server socket\n";

return 1;

}

std::cout << "Server listening on port 3000\n";

while (true) {

// Create another (normal) socket and build a TCP connection when a request arrives

sockaddr\_in client\_address;

socklen\_t client\_address\_size = sizeof(client\_address);

int client\_socket = accept(server\_socket, (sockaddr\*)&client\_address, &client\_address\_size);

if (client\_socket == -1) {

std::cerr << "Failed to accept client connection\n";

continue;

}

// Create a new thread to handle this client

pthread\_t thread;

ClientData\* client\_data = new ClientData(client\_socket, filename);

pthread\_create(&thread, nullptr, handle\_client, client\_data);

}

// Close server socket

close(server\_socket);

return 0;

}

*Client-side.*

#include <iostream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

// Serial Number Program

int main() {

// Create a socket and connect to the server

int client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_socket == -1) {

std::cerr << "Failed to create client socket\n";

return 1;

}

sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(3000);

server\_address.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (connect(client\_socket, (sockaddr\*)&server\_address, sizeof(server\_address)) == -1) {

std::cerr << "Failed to connect to server\n";

return 1;

}

std::cout << "Connected to server\n";

std::string serial\_number;

std::string registration\_number;

std::string name;

// Loop until the user chooses to exit

while (true) {

std::cout << "Enter a serial number (or 'exit' to quit): ";

std::getline(std::cin, serial\_number);

// Check if the user wants to exit

if (serial\_number == "exit") {

send(client\_socket, serial\_number.c\_str(), serial\_number.size(), 0);

break;

}

// Send the serial number to the server

int bytes\_sent = send(client\_socket, serial\_number.c\_str(), serial\_number.size(), 0);

if (bytes\_sent == -1) {

std::cerr << "Failed to send data to server\n";

continue;

}

// Wait for response from server

char buffer[1024];

int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

continue;

}

std::string response(buffer, bytes\_received);

std::cout << response << std::endl;

if (response == "Serial number already exists.\n") {

continue;

}

while (true) {

std::cout << "Enter a Registration Number (P15/...): ";

std::getline(std::cin, registration\_number);

// Send the registration number to the server

bytes\_sent = send(client\_socket, registration\_number.c\_str(), registration\_number.size(), 0);

if (bytes\_sent == -1) {

std::cerr << "Failed to send data to server\n";

break;

}

// Wait for response from server

bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

break;

}

response = std::string(buffer, bytes\_received);

std::cout << response << std::endl;

if (response == "Registration number already exists.\n") {

continue;

} else {

break;

}

}

std::cout << "Enter a name: ";

std::getline(std::cin, name);

// Send the name to the server

bytes\_sent = send(client\_socket, name.c\_str(), name.size(), 0);

if (bytes\_sent == -1) {

std::cerr << "Failed to send data to server\n";

continue;

}

// Wait for response from server

bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

continue;

}

response = std::string(buffer, bytes\_received);

std::cout << response << std::endl;

}

// Close the connection

close(client\_socket);

return 0;

}

# 

# 2.CONCURRENT CONNECTIONLESS.

This is a UDP server that listens for incoming datagrams from clients on port 3000. When it receives a datagram, it processes the message and sends a response back to the client. The server listens indefinitely until terminated manually.

The purpose of the server is to add serial numbers, registration numbers, and names to a file. When the server receives a datagram with a serial number, it checks if the serial number already exists in the file. If it does not exist, the server appends the serial number to the file and sends a response to the client saying that the serial number was added. If the serial number already exists, the server sends a response to the client saying that the serial number already exists.

Similarly, when the server receives a datagram with a registration number or name, it checks if the registration number or name already exists in the file. If it does not exist, the server appends the registration number or name to the file and sends a response to the client saying that the registration number or name was added. If the registration number or name already exists, the server sends a response to the client saying that the registration number or name already exists.

The server creates a thread to handle each client that connects to it. The thread reads the client's requests and sends responses back. When the client sends an "exit" message, the thread terminates and the client disconnects.

**Conceptual server algorithm.**

The program is a UDP server that listens on port 3000 for requests from clients. The conceptual server algorithm is as follows:

1. Create a server-socket (listener) with socket() and bind it to a port (3000) using bind().
2. Continuously listen for incoming requests from clients using recvfrom(). The requests are expected to contain data in the form of strings.
3. When a request is received, check if the client wants to exit. If so, break from the loop.
4. Otherwise, process the request by checking if the received data already exists in the file named "serial\_reg\_and\_name.txt".
5. If the data does not exist, append it to the end of the file. Otherwise, skip appending and inform the client that the data already exists.
6. Send a response to the client indicating whether the data was added or not using sendto().
7. Repeat steps 2 to 6 until the client exits.
8. When the client exits, clean up and return to listening for requests.

The program uses pthreads to handle multiple clients concurrently. When a client connects, a new thread is created to handle the client's requests using the handle\_client() function. The thread is passed a struct containing information about the client, including the server socket, filename, client address, and client address size. The handle\_client() function performs steps 2 to 8 of the server algorithm for each client.

Code.

*Server-side.*

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <pthread.h>

struct ClientData {

int server\_socket;

std::string filename;

sockaddr\_in client\_address;

socklen\_t client\_address\_size;

ClientData(int server\_socket, std::string filename, sockaddr\_in client\_address, socklen\_t client\_address\_size)

: server\_socket(server\_socket), filename(filename), client\_address(client\_address), client\_address\_size(client\_address\_size) {}

};

void\* handle\_client(void\* arg) {

ClientData\* client\_data = (ClientData\*)arg;

int server\_socket = client\_data->server\_socket;

std::string filename = client\_data->filename;

sockaddr\_in client\_address = client\_data->client\_address;

socklen\_t client\_address\_size = client\_data->client\_address\_size;

std::string serial\_number;

std::string registration\_number;

std::string name;

// Handle multiple requests from this client

while (true) {

// Receive request from client

char buffer[1024];

int bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0,(sockaddr\*)&client\_address,&client\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

}

// Print received data

std::cout << "Received data from client: " << std::string(buffer, bytes\_received) << "\n";

// Check if the client wants to exit

if (std::string(buffer, bytes\_received) == "exit") {

break;

}

// Process request

serial\_number = std::string(buffer, bytes\_received);

bool exists = false;

std::ifstream infile(filename);

std::string line;

while (getline(infile, line)) {

if (line == serial\_number) {

exists = true;

break;

}

}

infile.close();

if (!exists) {

std::ofstream outfile(filename, std::ios\_base::app);

outfile << serial\_number << "\n";

outfile.close();

sendto(server\_socket, "Serial number added.\n", 20, 0,(sockaddr\*)&client\_address,client\_address\_size);

} else {

sendto(server\_socket, "Serial number already exists.\n", 29, 0,(sockaddr\*)&client\_address,client\_address\_size);

continue;

}

bytes\_received = recvfrom(server\_socket, buffer,sizeof(buffer),0,(sockaddr\*)&client\_address,&client\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

}

// Print received data

std::cout << "Received data from client: " << std::string(buffer, bytes\_received) << "\n";

registration\_number = std::string(buffer, bytes\_received);

exists = false;

infile.open(filename);

while (getline(infile, line)) {

if (line == registration\_number) {

exists = true;

break;

}

}

infile.close();

if (!exists) {

std::ofstream outfile(filename, std::ios\_base::app);

outfile << registration\_number << "\n";

outfile.close();

sendto(server\_socket, "Registration number added.\n", 25, 0,(sockaddr\*)&client\_address,client\_address\_size);

} else {

sendto(server\_socket, "Registration number already exists.\n", 34, 0,(sockaddr\*)&client\_address,client\_address\_size);

}

bytes\_received = recvfrom(server\_socket, buffer,sizeof(buffer),0,(sockaddr\*)&client\_address,&client\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

}

// Print received data

std::cout << "Received data from client: " << std::string(buffer, bytes\_received) << "\n";

name = std::string(buffer, bytes\_received);

std::ofstream outfile(filename, std::ios\_base::app);

outfile << name << "\n";

outfile.close();

sendto(server\_socket, "Name added.\n", 11, 0,(sockaddr\*)&client\_address,client\_address\_size);

}

delete client\_data;

return nullptr;

}

int main() {

std::string filename = "serial\_reg\_and\_name.txt";

// Create a server-socket (listener) and bind it to a port (higher than 1024)

int server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (server\_socket == -1) {

std::cerr << "Failed to create server socket\n";

return 1;

}

sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(3000);

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(server\_socket, (sockaddr\*)&server\_address, sizeof(server\_address)) == -1) {

std::cerr << "Failed to bind server socket\n";

return 1;

}

std::cout << "Server listening on port 3000\n";

while (true) {

// Receive request from client

char buffer[1024];

sockaddr\_in client\_address;

socklen\_t client\_address\_size = sizeof(client\_address);

int bytes\_received = recvfrom(server\_socket, buffer,sizeof(buffer),0,(sockaddr\*)&client\_address,&client\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

continue;

}

// Print received data

std::cout << "Received data from client: " << std::string(buffer, bytes\_received) << "\n";

// Check if the client wants to exit

if (std::string(buffer, bytes\_received) == "exit") {

break;

}

// Create a new thread to handle this client

pthread\_t thread;

ClientData\* client\_data = new ClientData(server\_socket, filename, client\_address, client\_address\_size);

pthread\_create(&thread, nullptr, handle\_client, client\_data);

}

// Close server socket

close(server\_socket);

return 0;

}

*Client-side*.

#include <iostream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

int main() {

// Create a socket

int client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (client\_socket == -1) {

std::cerr << "Failed to create client socket\n";

return 1;

}

sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(3000);

server\_address.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

std::string serial\_number;

std::string registration\_number;

std::string name;

// Loop until the user chooses to exit

while (true) {

std::cout << "Enter a serial number (or 'exit' to quit): ";

std::getline(std::cin, serial\_number);

// Check if the user wants to exit

if (serial\_number == "exit") {

sendto(client\_socket, serial\_number.c\_str(), serial\_number.size(), 0,(sockaddr\*)&server\_address,sizeof(server\_address));

break;

}

// Send the serial number to the server

int bytes\_sent = sendto(client\_socket, serial\_number.c\_str(), serial\_number.size(), 0,(sockaddr\*)&server\_address,sizeof(server\_address));

if (bytes\_sent == -1) {

std::cerr << "Failed to send data to server\n";

continue;

}

// Print sent data

std::cout << "Sent data to server: " << serial\_number << "\n";

// Wait for response from server

char buffer[1024];

socklen\_t server\_address\_size = sizeof(server\_address);

int bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0,(sockaddr\*)&server\_address,&server\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

continue;

}

std::string response(buffer, bytes\_received);

std::cout << response << std::endl;

if (response == "Serial number already exists.\n") {

continue;

}

while (true) {

std::cout << "Enter a Registration Number (P15/...): ";

std::getline(std::cin, registration\_number);

// Send the registration number to the server

bytes\_sent = sendto(client\_socket, registration\_number.c\_str(), registration\_number.size(), 0,(sockaddr\*)&server\_address,sizeof(server\_address));

if (bytes\_sent == -1) {

std::cerr << "Failed to send data to server\n";

break;

}

// Print sent data

std::cout << "Sent data to server: " << registration\_number << "\n";

// Wait for response from server

bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0,(sockaddr\*)&server\_address,&server\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

break;

}

response = std::string(buffer, bytes\_received);

std::cout << response << std::endl;

if (response == "Registration number already exists.\n") {

continue;

} else {

break;

}

}

std::cout << "Enter a name: ";

std::getline(std::cin, name);

// Send the name to the server

bytes\_sent = sendto(client\_socket, name.c\_str(), name.size(), 0,(sockaddr\*)&server\_address,sizeof(server\_address));

if (bytes\_sent == -1) {

std::cerr << "Failed to send data to server\n";

continue;

}

// Print sent data

std::cout << "Sent data to server: " << name << "\n";

// Wait for response from server

bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0,(sockaddr\*)&server\_address,&server\_address\_size);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

continue;

}

response = std::string(buffer, bytes\_received);

std::cout << response << std::endl;

}

// Close the socket

close(client\_socket);

return 0;

}

# 

# 

# 3.ITERATIVE CONNECTION-ORIENTED.

The code implements a simple client-server program for registering and viewing student information. The server listens for incoming connections and handles requests to view or register a new student. The client can send requests to the server to view or register new students.

When a request to view students is received, the server opens a file containing the student information, reads the contents of the file, and sends the data back to the client.

When a request to register a new student is received, the server reads the registration data from the client, checks if the registration number is already in use, and saves the data to a file if the registration number is not in use.

Overall, the code seems to be functioning as intended, although there are a few areas where it could be improved. For example, the code should check for errors when sending or receiving data over the network, and handle those errors appropriately. Additionally, there are some potential security vulnerabilities, such as the use of strcpy and strcat without bounds checking, which could allow a malicious client to exploit a buffer overflow.

**Conceptual server algorithm.**

The conceptual server algorithm for the program is:

1. Create a socket for the server
2. Set socket options to allow reuse of address and port
3. Bind the socket to a port and address
4. Listen for incoming connections
5. Accept incoming connections and handle them
6. When a connection is accepted, receive a command from the client
7. If the command is "view", open the file containing the database of registrations and send its contents to the client
8. If the command is "register", receive the registration data from the client, check if the registration number is already in use, and save the data to the database file
9. If the command is unknown, send an error message to the client
10. Close the connection

The handle\_request function handles the client's request by following steps 7 to 9 of the above algorithm. The while loop in the main function handles steps 5 to 10, while the steps 1 to 4 are performed before the loop.

Code.

*Server-side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

static FILE \*database;

struct info {

char name[10];

char reg[10];

char serial[10];

};

void handle\_request(int sock);

int main(int argc, char const \*argv[])

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

// Create a socket for the server

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Set socket options to allow reuse of address and port

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

&opt, sizeof(opt)))

{

perror("setsockopt failed");

exit(EXIT\_FAILURE);

}

// Bind the socket to a port and address

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(8080);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0)

{

perror("listen failed");

exit(EXIT\_FAILURE);

}

// Accept incoming connections and handle them

while (1)

{

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address,

(socklen\_t \*)&addrlen)) < 0)

{

perror("accept failed");

exit(EXIT\_FAILURE);

}

handle\_request(new\_socket);

close(new\_socket);

}

return 0;

}

void handle\_request(int sock)

{

char buffer[1024] = {0};

int valread = read(sock, buffer, 1024);

if (strcmp(buffer, "view") == 0)

{

printf("Viewing........\n");

// Send the contents of the file to the client

database = fopen("registrations.txt", "a+");

if (database == NULL)

{

send(sock, "Error openning database", strlen("Error openning database"), 0);

}

else

{

char line[1024];

char result[1024];

char temp[1024];

while(fgets(temp,50,database)){

strcpy(line,temp);

strcat(line,"\n");

strcat(result,line);

}

send(sock, result, strlen(result), 0);

strcpy(result,"");

fclose(database);

printf("Done sending!\n");

}

}

else if (strcmp(buffer, "register") == 0)

{

// Receive the registration data from the client and save it to the file

struct info info;

printf("Registering........\n");

valread = read(sock, &info, sizeof(info));

database = fopen("registrations.txt", "a+");

if (database == NULL)

{

send(sock, "Error openning database", strlen("Error openning database"), 0);

}

char temp[100];

int index = 0;

while(fgets(temp,100,database)!=NULL){

char \*token = strtok(temp," ");

int count = 0;

while(token != NULL){

if (count== 2){

printf("3: %s\n",token);

if(strstr(token,info.reg)!=NULL){

index++;

printf("%d",index);

}

}

count++;

token = strtok(NULL, " ");

}

}

if (index == 0)

{

fprintf(database, "%s %s %s\n", info.name, info.serial, info.reg);

fclose(database);

send(sock, "Registration complete!", strlen("Registration complete!"), 0);

printf("Done registering\n");

}else{

send(sock, "Registration number in use!", strlen("Registration number in use!"), 0);

printf("Registration failed!!");

}

}else

{

send(sock, "unknown command", strlen("unknown command"), 0);

}

}

*Client-side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

struct info {

char name[10];

char reg[10];

char serial[10];

};

int main(int argc, char const \*argv[]) {

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

char buffer[1024] = {0};

struct info info;

// Creating socket file descriptor

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

memset(&serv\_addr, '0', sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

// Main menu

int option;

printf("\n1. Register a new user\n");

printf("2. View existing registrations\n");

printf("3. Exit\n");

printf("Enter an option: ");

scanf("%d", &option);

switch(option) {

case 1: // Register a new user

send(sock, "register", sizeof("register"), 0);

printf("Enter name: ");

scanf("%s", info.name);

printf("Enter serial number: ");

scanf("%s", info.serial);

printf("Enter registration number: ");

scanf("%s", info.reg);

send(sock, &info, sizeof(info), 0);

read(sock, buffer, 1024);

printf("%s\n",buffer);

if(strcmp(buffer,"error")==0) {exit(1);}

break;

case 2: // View existing registrations

send(sock, "view", strlen("view"), 0);

valread = read(sock, buffer, 1024);

printf("Registrations:\n");

printf("%s",buffer);

break;

case 3: // Exit

printf("Exiting program\n");

break;

default:

printf("Invalid option\n");

break;

}

close(sock);

return 0;

}

# 4.ITERATIVE CONNECTIONLESS.

This code implements a simple program for adding serial numbers, registration numbers, and names to a file using socket programming in C. The program consists of two parts: a server program and a client program.

The server program listens for requests from the client program and processes them accordingly. When a client connects to the server, the server creates a UDP socket and binds it to port 3000. The server then waits for incoming messages from the client. If the message is "exit," the server terminates the connection. Otherwise, the server checks if the serial number exists in the file. If it does not exist, the server appends the serial number to the file and sends a confirmation message to the client. If it exists, the server sends an error message to the client. The server then waits for the registration number and name from the client and follows a similar procedure for adding these to the file.

The client program allows the user to input serial numbers, registration numbers, and names, which are sent to the server for processing. The client program creates a UDP socket and connects it to the server address (localhost, port 3000). The client then loops until the user enters "exit." For each loop iteration, the client prompts the user to enter a serial number, registration number, and name. The client then sends these values to the server for processing and receives a confirmation message in response.

Overall, the program is a simple example of how socket programming can be used to create a simple client-server application in C.

**Conceptual server algorithm.**

The server algorithm for this program is as follows:

1. Create a server-socket (listener) and bind it to a port (in this case, port 3000).
2. Enter into an infinite loop to listen for client requests.
3. Receive a request from the client, which is a serial number.
4. Check if the serial number already exists in a file. If not, add the serial number to the file and send a confirmation message to the client.
5. Receive the next request from the client, which is a registration number.
6. Check if the registration number already exists in the file. If not, add the registration number to the file and send a confirmation message to the client.
7. Receive the final request from the client, which is a name.
8. Add the name to the file and send a confirmation message to the client.
9. Loop back to step 3 and wait for the next client request.
10. If the client sends the message "exit", break out of the loop and close the server socket.

Code.

*Server-side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

// Serial Number Program

int main() {

char\* filename = "iterative\_connectionless-C.txt";

char serial\_number[1024];

char registration\_number[1024];

char name[1024];

// Create a server-socket (listener) and bind it to a port (higher than 1024)

int server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (server\_socket == -1) {

fprintf(stderr, "Failed to create server socket\n");

return 1;

}

struct sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(3000);

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(server\_socket, (struct sockaddr\*)&server\_address, sizeof(server\_address)) == -1) {

fprintf(stderr, "Failed to bind server socket\n");

return 1;

}

printf("Server listening on port 3000\n");

while (true) {

// Receive request from client

char buffer[1024];

struct sockaddr\_in client\_address;

socklen\_t client\_address\_size = sizeof(client\_address);

int bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr\*)&client\_address, &client\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from client\n");

continue;

}

// Check if the client wants to exit

if (strncmp(buffer, "exit", bytes\_received) == 0) {

break;

}

// Process request

strncpy(serial\_number, buffer, bytes\_received);

serial\_number[bytes\_received] = '\0';

bool exists = false;

FILE\* infile = fopen(filename, "a+");

char line[1024];

while (fgets(line, sizeof(line), infile)) {

if (strstr(line, serial\_number) != NULL) {

exists = true;

break;

}

}

fclose(infile);

if (!exists) {

FILE\* outfile = fopen(filename, "a+");

fprintf(outfile, "%s\n", serial\_number);

fclose(outfile);

sendto(server\_socket, "Serial number added.\n", 20, 0, (struct sockaddr\*)&client\_address, client\_address\_size);

} else {

sendto(server\_socket, "Serial number already exists.\n", 29, 0, (struct sockaddr\*)&client\_address, client\_address\_size);

continue;

}

bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr\*)&client\_address, &client\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from client\n");

continue;

}

strncpy(registration\_number, buffer, bytes\_received);

registration\_number[bytes\_received] = '\0';

exists = false;

infile = fopen(filename, "a+");

while (fgets(line, sizeof(line), infile)) {

if (strstr(line, registration\_number) != NULL) {

exists = true;

break;

}

}

fclose(infile);

if (!exists) {

FILE\* outfile = fopen(filename, "a+");

fprintf(outfile, "%s\n", registration\_number);

fclose(outfile);

sendto(server\_socket, "Registration number added.\n", 25, 0, (struct

sockaddr\*)&client\_address, client\_address\_size);

} else {

sendto(server\_socket, "Registration number already exists.\n", 34, 0, (struct sockaddr\*)&client\_address, client\_address\_size);

continue;

}

bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr\*)&client\_address, &client\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from client\n");

continue;

}

strncpy(name, buffer, bytes\_received);

name[bytes\_received] = '\0';

FILE\* outfile = fopen(filename, "a+");

fprintf(outfile, "%s\n", name);

fclose(outfile);

// Send confirmation message to client

sendto(server\_socket, "Name added.\n", 12, 0, (struct sockaddr\*)&client\_address, client\_address\_size);

}

// Close server socket

close(server\_socket);

return 0;

}

*Client-side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

// Serial Number Program

int main() {

// Create a socket

int client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (client\_socket == -1) {

fprintf(stderr, "Failed to create client socket\n");

return 1;

}

struct sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(3000);

server\_address.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

char serial\_number[1024];

char registration\_number[1024];

char name[1024];

// Loop until the user chooses to exit

while (1) {

printf("Enter a serial number (or 'exit' to quit): ");

fgets(serial\_number, 1024, stdin);

serial\_number[strcspn(serial\_number, "\n")] = 0;

// Check if the user wants to exit

if (strcmp(serial\_number, "exit") == 0) {

sendto(client\_socket, serial\_number, strlen(serial\_number), 0, (struct sockaddr\*)&server\_address, sizeof(server\_address));

break;

}

// Send the serial number to the server

int bytes\_sent = sendto(client\_socket, serial\_number, strlen(serial\_number), 0, (struct sockaddr\*)&server\_address, sizeof(server\_address));

if (bytes\_sent == -1) {

fprintf(stderr, "Failed to send data to server\n");

continue;

}

// Wait for response from server

char buffer[1024];

socklen\_t server\_address\_size = sizeof(server\_address);

int bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, (struct sockaddr\*)&server\_address, &server\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from server\n");

continue;

}

buffer[bytes\_received] = '\0';

printf("%s\n", buffer);

if (strcmp(buffer, "Serial number already exists.\n") == 0) {

continue;

}

while (1) {

printf("Enter a Registration Number (P15/...): ");

fgets(registration\_number, 1024, stdin);

registration\_number[strcspn(registration\_number, "\n")] = 0;

// Send the registration number to the server

bytes\_sent = sendto(client\_socket, registration\_number, strlen(registration\_number), 0, (struct sockaddr\*)&server\_address, sizeof(server\_address));

if (bytes\_sent == -1) {

fprintf(stderr, "Failed to send data to server\n");

break;

}

// Wait for response from server

bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, (struct sockaddr\*)&server\_address, &server\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from server\n");

break;

}

buffer[bytes\_received] = '\0';

printf("%s\n", buffer);

if (strcmp(buffer, "Registration number already exists.\n") == 0) {

continue;

} else {

break;

}

}

printf("Enter a name: ");

fgets(name, 1024, stdin);

name[strcspn(name, "\n")] = 0;

// Send the name to the server

bytes\_sent = sendto(client\_socket, name, strlen(name), 0, (struct sockaddr\*)&server\_address, sizeof(server\_address));

if (bytes\_sent == -1) {

fprintf(stderr, "Failed to send data to server\n");

continue;

}

// Wait for response from server

bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, (struct sockaddr\*)&server\_address, &server\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from server\n");

continue;

}

buffer[bytes\_received] = '\0';

printf("%s\n", buffer);

// Print the summary of the entered data

printf("Summary:\nSerial Number: %s\nRegistration Number: %s\nName: %s\n", serial\_number, registration\_number, name);

}

// Close the socket

close(client\_socket);

return 0;

}