CONCURRENT CONNECTION ORIENTED PROGRAM

# Group Members:

Reuben Jefwa Balozi: P15/1930/2020

Sydney Nzunguli Kathina: P15/139965/2020

Sameera Bashir Kherdin: P15/1924/2020

Halima Ali Wario: P15/1903/2020

Douglas Omega: P15/1902/2020

# Server Program:

1. The program creates a socket using the socket() function. The socket is of IPV4 and TCP protocol(SOCK\_STREAM).

2. Bind socket with address and port

3. Listen for any connection requests

4. Accept connection request

5. Create child process and hand it the client

6. Receive data from the client

7. Process the client’s request

8. Formulate and send response to the client

9. Close file descriptor

10.Close main server socket.

All these have been highlighted in the text in red

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <sys/types.h>

#include <netinet/in.h>

#define SERVER\_PORT 3000

void handle\_connection(int j);

int main(int argc, char \*\*argv) {

**//creating a socket using the socket function**

//Sock: Socket file descriptor

//AF\_INET: Address family used for the socket in this case IPv4

//SOCK\_STREAM: The type of socket to create in this case a stream socket since it is a connection-oriented socket

//0: The protocol to use this case, the default protocol for the given address family and socket type

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

if (sock < 0) {

perror("socket");

exit(EXIT\_FAILURE);

}

//Creating a server\_addr struct that will store information about the server

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr));

//The address family used for the socket

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

//Converts string IP address to binary form

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

//Converts a port number to network byte order

server\_addr.sin\_port = htons(SERVER\_PORT);

**//binding the socket to the server information using the bind function**

if (bind(sock, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("bind");

exit(EXIT\_FAILURE);

}

**//Listen for incoming connections. 8:Maximum number of pending connections**

if (listen(sock, 8) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", SERVER\_PORT);

//Infinite loop to handle client connections

while (1) {

//Creating a client\_addr struct that will store information about the client

struct sockaddr\_in client\_addr;

socklen\_t client\_addr\_len = sizeof(client\_addr);

**//Accept incoming connections using the accept function**

//Sock: Socket file descriptor

//(struct sockaddr\*)&client\_addr: Pointer to the client information

//&client\_addr\_len: Size of the client information

int j = accept(sock, (struct sockaddr\*)&client\_addr, &client\_addr\_len);

if (j < 0) {

perror("accept");

continue;

}

**//creating child processes**

pid\_t pid = fork();

if (pid == 0) {

// child process

close(sock);

handle\_connection(j);

close(j);

exit(EXIT\_SUCCESS);

} else if (pid > 0) {

// parent process

close(j);

} else {

perror("fork");

}

}

**//close the socket file descriptor**

close(sock);

return 0;

}

void handle\_connection(int j) {

**//Receive data from the client using the recv function and storing it in client\_data variable**

char client\_data[100];

int n = recv(j, client\_data, sizeof(client\_data), 0);

if (n < 0) {

perror("recv");

exit(EXIT\_FAILURE);

}

client\_data[100] = '\0';

**//Process client request**

//Checking if the data received already exists int the file

// Open the file in read mode

FILE\* fp = fopen("registration.txt", "r");

// Check if the file was opened successfully

if (fp == NULL) {

printf("Error opening file!");

}

// Read each line from the file and check if received record already exists

char line[100];

while (fgets(line, 100, fp)) {

//char line\_details variable that will hold each line in the file that will be used to compare with client\_data;

char line\_details[100];

sscanf(line, "%s", line\_details);

if (strcmp(client\_data, line\_details)==0) {

//Record already exists in the file

//close the file

fclose(fp);

**//Send reply to the client informing client that their record already exists in the** **file**

send(j,"Record already exists!!!", strlen("Record already exists!!!"), 0);

//close the socket file descriptor

close(j);

}

}

//Record does not exist in the file

// Close the file

fclose(fp);

//Open the file in append mode

fopen("registration.txt", "a");

// Check if the file was opened successfully

if (fp == NULL) {

printf("Error opening file!");

}

//Write the contents of what was recieved from the client into the file

fprintf(fp, "%s\n", client\_data);

//close the file

fclose(fp);

//Send reply of success to the client

send(j, "Records have been saved successfully", strlen("Records have been saved successfully"), 0);

}

# Client Program:

1. The program creates a socket using the socket() function. The socket is of IPV4 and TCP protocol(SOCK\_STREAM).

2. The client program connects to the server using the connect() function.

3. The client program sends data to the server using the send() function.

4. The client program receives a reply from the server using the recv() function.

5. The client program closes its socket using the close() function.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <unistd.h>

#define SERVER\_PORT 5000

int main(int argc, char\*\*argv) {

if(argc != 2){

perror("Usage: TCPClient <IP address of the server");

exit(1);

}

**//Creating a socket using the socket function**

//Sock: Socket file descriptor

//AF\_INET: Address family used for the socket in this case IPv4

//SOCK\_STREAM: The type of sockt to create in this case a stream socket since it is a connection-oriented socket

//0: The protocol to use this case, the default protocol for the given address family and socket type

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

if (sock < 0) {

perror("socket");

exit(EXIT\_FAILURE);

}

//Creating a server\_addr struct that will store information about the server

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr));

//The address family used for the socket

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

//Converts string IP address to binary form

server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

//Converts a port number to network byte order

server\_addr.sin\_port = htons(SERVER\_PORT);

**//Conecting to the server using the connect function**

//Sock: Socket file descriptor

//(struct sockaddr\*)&server\_addr: A pointer to the server information

//sizeof(server\_addr): the size of the server information struct

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

perror("connect");

exit(EXIT\_FAILURE);

}

printf("Connected to server\n");

//Variable for storing name

char name[40];

//Variable for storing registration\_number

char registration\_number[20];

//Variable for storing serial number

int serial\_number;

//Capturing serial number from user and storing it a variable

printf("Enter your serialnumber:(MiddlepartofregnoLastpartofregno) ");

scanf("%d", &serial\_number);

//Capturing registration number from user and storing it in a variable

printf("Enter your registration number: ");

scanf("%s", registration\_number);

//Capturing full name from user and storing it in a variable

printf("Enter your full name:(FirstnameLastname) ");

scanf("%s", name);

//Constructing a message(client\_data) to send to the server by concatenating user's input into a single string

char client\_data[100];

sprintf(client\_data, "%d,%s,%s", serial\_number, registration\_number, name);

**//Sending the message to the server using the send function**

if (send(sock, client\_data, strlen(client\_data), 0) < 0) {

perror("send");

exit(EXIT\_FAILURE);

}

// Variable for receiving response from the server

char server\_response[100];

**//Receiving response from the server**

int j = recv(sock, server\_response, sizeof(server\_response), 0);

if (j < 0) {

perror("recv");

exit(EXIT\_FAILURE);

}

else

{

printf("%s\n", server\_response);

bzero(server\_response, sizeof(server\_response));

}

**//Close the socket file descriptor**

close(sock);

return 0;

}