ITERATIVE CONNECTIONLESS 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:

The conceptual server algorithm for the server program is as follows:

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

2. The program binds the socket with an address and port using the bind() function.

3. The program then receives message from the client through the created socket using the recvfrom() function.

4. The program services the request of the client.

5. The program formulates and sends responses to the client using the sendto() function.

6. The program closes the socket file descriptor using the close() function.

All these have been highlighted in red text in the document.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#define SERVER\_PORT 1500

int main() {

//creating a socket using the socket function

//Sock: Socket file descriptor

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

//SOCK\_DGRAM: The type of socket to create in this case a datagram socket since it is a connectionless 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\_DGRAM, 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 a port number to network byte order

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

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

**//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);

}

printf("Server is ready to receive from port %d...\n", SERVER\_PORT);

//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);

**//Recieve data from the client using the recvfrom function and storing it in client\_data variable**

char client\_data[100];

int n = recvfrom(sock, client\_data, sizeof(client\_data), 0, (struct sockaddr \*) &client\_addr, &client\_addr\_len);

if (n < 0) {

perror("recvfrom");

exit(EXIT\_FAILURE);

}

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

**//Process client request**

//Checking if the data received already exists in 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);

**//Formulate and send reply to the client informing client that their record already exists**

int j = sendto(sock, "Record already exists!!!", strlen("Record already exists!!!"), 0, (struct sockaddr\*)&client\_addr, sizeof(client\_addr));

if (j < 0) {

perror("sendto");

exit(EXIT\_FAILURE);

}

close(j);

//close the socket file descriptor

close(sock);

return 1;

}

}

//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 received from the client into the file

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

//Close the file

fclose(fp);

**//Formulate and Send response of success to the client**

int j = sendto(sock, "Records have been saved sucessfully", strlen("Records have been saved successfully"), 0, (struct sockaddr\*)&client\_addr, sizeof(client\_addr));

if (j < 0) {

perror("sendto");

exit(EXIT\_FAILURE);

}

close(j);

**//Close the socket file descriptor**

close(sock);

return 0;

}

# Client Program:

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

2. gethostbyname() function.

3. The program then send a message to the server through the created socket using the sendto() function.

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

5. The program closes the socket file descriptor using the close() function.

All these have been highlighted in red text in the document.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <netinet/in.h>

#include <netdb.h>

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 1500

int main() {

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

//Sock: Socket file descriptor

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

//SOCK\_DGRAM: The type of sockt to create in this case a datagram socket since it is connectionless

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

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

struct hostent \*host;

//The address family used for the socket

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

//Converts a port number to network byte order

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

**//Gethostbyname() function**

host = gethostbyname(SERVER\_IP);

server\_addr.sin\_addr = \*((struct in\_addr \*)host->h\_addr);

//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 form 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 sendto() function**

int j = sendto(sock, client\_data, strlen(client\_data), 0, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

if (j < 0) {

perror("sendto");

exit(EXIT\_FAILURE);

}

//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);

//Receiving response from the server using recvfrom() function and printing it out

char server\_response[100];

int k = recvfrom(sock, server\_response, sizeof(server\_response), 0, (struct sockaddr\*)&client\_addr, &client\_addr\_len);

if (k < 0) {

perror("recvfrom");

exit(EXIT\_FAILURE);

}

else{

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

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

}

**//Close the socket file descriptor**

close(sock);

return 0;

}