Logo, company name

Description automatically generated

EE353: Computer Networks

Assignment No. #01

Submitted to: Dr. Muhammad Zeeshan

Section: BSCS-10-A

Submitted by:

|  |  |
| --- | --- |
| Name | CMS ID |
| Hassaan Nadeem | 340187 |
| Baqir e Muhammad | 356157 |

Assignment 1

# Description

The program is based on a server-client model with the receiver acting as the client and the sender as server. The working of the program is as follows:

1. Server turns on and creates a socket. The socket IP is generated via a C script using gethostname() and gethostbyname() functions of netdb.h library. The IP is printed on the server terminal for ease as it is a dynamic IP and not hardcoded. The port number on the other hand is hardcoded to be 8080.
2. Client connects to the server using the server IP and port number.
3. If connection is successful, client receives a message and server prints a message as well.
4. If connection is unsuccessful, client receives a “*time out”* message after 7 seconds. The client retries twice before timing out.
5. Once the connection has been established, the client requests a file from server. The file name is sent by writing in socket.
6. Server receives the file name from socket and checks in its local directory. Then it writes back in socket as to whether it exists or not.
7. Client reads from socket to see if file exists or not. This is done by string comparison, if the received string contains the string *“exists”* then the file exists. This comparison is done via strstr function (strcmp is not used as the buffer might have some garbage values already and hence is not reliable).
8. If the file exists, the client becomes ready to receive data from socket. Else the client writes in terminal that the file does not exist.
9. If the file exists, the server opens the file as FILE object and reads its data into an array. The array is then sent to the client via the socket. The packet size used here is 32 Bytes but can be up to 1000-1500 Bytes.
10. The client creates an empty FILE object with the same extension as the requested file (using substrings) and reads data from buffer. The data is then written in the new file and the process is repeated.
11. The server keeps sending data via a loop. Once the file ends, the loop breaks and a final message is written in the socket. This message contains a unique string saying *“EOF”* 32 times. The data is repeated 32 times to make sure the write function receives whole message as it is a blocking code. Once this last message is sent, the server prints on the terminal that the file has been sent successfully and the socket is closed from server side.
12. On the client side, in the loop each message is examined to check if it contains the string *EOF*. If it does (meaning that the file has ended), it writes the final bytes in the file before closing the socket from client side as well.
13. The interface then provides the client to reconnect to the server in case more files are to be requested.
14. The server on the other hand remains open looking for further connections.

The program works on any type of file extension as it uses substrings to generate the new file of same extension. As long as the data of the file is not encrypted, it can be sent to the client. Furthermore, the program also allows connection between two different machine as long as they are on the same network. This is done by getting the server IP using a script each time the server is launched as the IP is dynamic and cannot be hardcoded. The program as of right now does not work on different networks as there is a conflict of public and private IP. This can be solved via TCP Hole-Punching technique by using a server. However, the functionality has not been coded as of yet.

Finally, the code of both server and client is given below. Comments have been added for ease of read. The screenshots showing the working are also attached.

**sender.c**

//Sender - Server

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h> //for socket reading and writing

#include <arpa/inet.h>  //for inet\_ntoa

#include <netdb.h>  //for gethostname and hostent

#define MAX\_SIZE 32

char\* get\_machine\_ip() {

    char buffer[256];

    struct hostent \*hostentry;

    int hostname;

    char \*IP;

    hostname = gethostname(buffer, sizeof(buffer));     // To retrieve hostname

    if (hostname == -1) {

        puts("Error getting hostname");

        exit(0);

    }

    hostentry = gethostbyname(buffer);      // To retrieve host information

    if (hostentry == NULL) {

        puts("Error getting hostname");

        exit(0);

    }

    IP = inet\_ntoa(\*((struct in\_addr\*) hostentry->h\_addr\_list[0]));

    return IP;

}

int main() {

    int socketfd, newsocketfd, n, portno = 8080;

    char buffer[MAX\_SIZE], file\_name[MAX\_SIZE];

    FILE \*fp;

    struct sockaddr\_in server\_addr, client\_addr;

    struct hostent \*server;

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

    char \*ip = get\_machine\_ip();   //get machine public ip

    server = gethostbyname(ip);

    socketfd = socket(AF\_INET, SOCK\_STREAM, 0);   //IPv4 protocols & 2-way reliable connection based

    if (socketfd < 0) {

        perror("Error while opening socket");

        return 0;

    }

    puts("Socket opened");

    bzero((char \*) &server\_addr, sizeof(server\_addr));  //erasing sizeof(server\_addr) bits starting at &server\_addr

    server\_addr.sin\_family = AF\_INET; //server address protocol

    bcopy((char \*)server->h\_addr, (char \*)&server\_addr.sin\_addr.s\_addr, server->h\_length);    //Server IP = Host IP

    server\_addr.sin\_port = htons(portno);     //Port no in "Host Byte Order"

    if (bind(socketfd, (struct sockaddr \*) &server\_addr, sizeof(server\_addr)) < 0) {  //Binding socket to server

        perror("Error while binding socket");

        return 0;

    }

    puts("Socket binded");

    printf("\nServer Address: %s:%d\n", ip, portno);

    listen(socketfd, 5);  //Listening for connections in socket -> Max 5

    while(1) {

        puts("\nWaiting for connection...");

        newsocketfd = accept(socketfd, (struct sockaddr \*) &client\_addr, &client\_len);     //Accepting connection

        if (newsocketfd < 0) {

            perror("Error on accepting connection");

        }

        else {

            //newsocketfd -> pair socket -> Sender + Receiver Pair

            printf("Client connected with ip %s\n", inet\_ntoa(client\_addr.sin\_addr));

            bzero(buffer, MAX\_SIZE);    //Clearing Buffer cuz its C duh

            while(1) {

                puts("Waiting for request...");

                bzero(buffer, MAX\_SIZE); //clearing char array (string)

                n = read(newsocketfd, buffer, MAX\_SIZE);  //reading file name -> via receiver/client request

                if (n < 0) {

                    perror("Error while reading from socket");

                }

                else {

                    if (!strcmp(buffer, "exit")) {

                        printf("Client %s disconnected\n", inet\_ntoa(client\_addr.sin\_addr));

                        close(newsocketfd); //Disconnect from pair socket

                        break;

                    }

                    strcpy(file\_name, buffer);  //saving file\_name for future use

                    printf("File request received from client %s\nFile name: %s\n", inet\_ntoa(client\_addr.sin\_addr), buffer);

                    //Send file

                    if(fp = fopen(buffer, "rb")) {  //File exists

                    //TO use buffer

                        char exists[MAX\_SIZE] = "exists";

                        write(newsocketfd, exists, MAX\_SIZE);

                        while (1) {

                            size\_t num\_read = fread(buffer, 1, MAX\_SIZE, fp);

                            if (num\_read == 0) {    //No elements returns -> EOF

                                break;

                            }

                            n = write(newsocketfd, buffer, num\_read);

                            if (n < 0) // Error

                                perror("Error while writing to socket");

                            else if (n == 0)    //Socket Closed

                                break;

                        }

                        char end[MAX\_SIZE] = "EOFEOFEOFEOFEOFEOFEOFEOF";  //Repeating string of EOF to denote end of file

                        write(newsocketfd, end, MAX\_SIZE);  //Sending End-of-File to receiver

                        printf("%s sent to client %s\n", file\_name, inet\_ntoa(client\_addr.sin\_addr));

                        if (fp != NULL)  {

                            fclose(fp);

                        }

                        puts("Client disconnected");

                        close(newsocketfd); //closing socket as file transfer complete

                        break;

                    }

                    else {

                        char exists[MAX\_SIZE] = "!exists";

                        write(newsocketfd, exists, MAX\_SIZE);

                        puts("File does not exist");

                    }

                }

            }

        }

    }

    return 0;

}

**receiver.c**

//Receiver - Client

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h> //for socket reading and writing

#include <netdb.h>  //for gethostname and hostent

#include <netinet/tcp.h>    //for setsockopt -> reduce connection timeout

#define MAX\_SIZE 32

char \*get\_filename\_ext(char \*filename) {

    char \*dot = strrchr(filename, '.');

    return dot;

}

int main() {

    int socketfd, portno = 8080, n, input, file\_number = 0;

    char buffer[MAX\_SIZE], file\_name[MAX\_SIZE], ip[MAX\_SIZE], port[MAX\_SIZE];

    FILE \*output;

    struct sockaddr\_in server\_addr;

    struct hostent \*server;

    printf("Enter Host IP: ");

    fgets(ip, MAX\_SIZE, stdin);

    ip[strcspn(ip, "\n")] = 0;  //removes trailing new line that fgets creates

    printf("Enter Server Port: ");

    fgets(port, MAX\_SIZE, stdin);

    port[strcspn(port, "\n")] = 0;  //removes trailing new line that fgets creates

    portno = atoi(port);

    printf("Server Address: %s:%d\n\n", ip, portno);

    while (1) {

        puts("1. Connect to Socket\n2. Enter new IP\n3. Exit");

        printf(": ");

        fgets(buffer, MAX\_SIZE, stdin);

        input = atoi(buffer);

        if (input == 1) {   //Connect to Server

            server = gethostbyname(ip);

            socketfd = socket(AF\_INET, SOCK\_STREAM, 0);   //IPv4 protocols & 2-way reliable connection based

            if (socketfd < 0) {

                perror("Error while opening socket");

            }

            else {

                bzero((char \*) &server\_addr, sizeof(server\_addr));  //erasing sizeof(server\_addr) bits starting at &server\_addr

                server\_addr.sin\_family = AF\_INET; //server address protocol

                bcopy((char \*)server->h\_addr, (char \*)&server\_addr.sin\_addr.s\_addr, server->h\_length);    //Server IP = Host IP

                server\_addr.sin\_port = htons(portno);     //Host byte order

                //To speed-up connect timeoutput

                int synRetries = 2; // Send a total of 3 SYN packets => Timeoutput ~7s

                setsockopt(socketfd, IPPROTO\_TCP, TCP\_SYNCNT, &synRetries, sizeof(synRetries));

                //Reference https://stackoverflow.com/a/46473173

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

                    perror("Error connecting");

                }

                else {

                    puts("\nSuccessfully connected to Server");

                    while(1) {

                            puts("1. Request File\n2. Close Socket");

                            printf(": ");

                            fgets(buffer, MAX\_SIZE, stdin);

                            input = atoi(buffer);

                        if (input == 1) {    //Request File

                            printf("\nEnter the file name: ");

                            memset(buffer, 0, MAX\_SIZE);    //clearing buffer cuz its C duh -> no risk

                            fgets(buffer, MAX\_SIZE - 1, stdin);

                            buffer[strlen(buffer) - 1] = '\0';

                            strcpy(file\_name, buffer);  //saving file name -> to separate extension later on

                            n = write(socketfd, buffer, strlen(buffer));  //Sending file name to server

                            if (n < 0) {

                                perror("Error while writing to socket");

                            }

                            else {

                                read(socketfd, buffer, MAX\_SIZE);     //Read whether file exists or not

                                if(!strcmp(buffer, "exists")) {  //file exists

                                    char new\_file\_name[MAX\_SIZE] = "receive\_";

                                    char temp[5];   //temp array to store file\_number (as string)

                                    char \*ext = get\_filename\_ext(file\_name);   //file extension

                                    sprintf(temp, "%d", file\_number);   //int to string conversion

                                    strcat(new\_file\_name,temp); //concatinating file number to file\_name

                                    strcat(new\_file\_name,ext);

                                    file\_number++;

                                    output = fopen(new\_file\_name, "wb");

                                    while (1) {

                                        n = read(socketfd, buffer, MAX\_SIZE);

                                        if(strstr(buffer, "EOF")) {  //check for EOF -> Custom repeating string that denotes end of string

                                            explicit\_bzero(strstr(buffer, "EOF"), MAX\_SIZE); //Clear n bytes of memory from the point where the first EOF identified

                                            fwrite(buffer, 1, n, output);  //Write final transferred buffer

                                            break;

                                        }

                                        if (n < 0)

                                            perror("Error while reading from socket");

                                        else if (n == 0) // Socket closed -> Transfer completed

                                            break;

                                        fwrite(buffer, 1, n, output);

                                    }

                                    puts("File transfer complete");

                                    if (output != NULL) {

                                        fclose(output);

                                    }

                                    puts("Disconnected from Socket");

                                    break;  //server closes socket after file transfer hence breaking the loop

                                }

                                else {  //Does not exist

                                    puts("\nFile does not exist in the server database");

                                }

                            }

                        }

                        else if (input == 2){

                            char msg[6] = "exit";

                            write(socketfd, msg, strlen(msg));

                            close(socketfd);

                            puts("\nDisconnected from Socket");

                            break;

                        }

                        puts("");

                    }

                }

            }

        }

        else if (input == 2){

            printf("\nEnter New Host IP: ");

            fgets(ip, MAX\_SIZE, stdin);

            ip[strcspn(ip, "\n")] = 0;  //removes trailing new line that fgets creates

            printf("Enter Server Port: ");

            fgets(port, MAX\_SIZE, stdin);

            port[strcspn(port, "\n")] = 0;  //removes trailing new line that fgets creates

            portno = atoi(port);

            printf("Server Address: %s:%d\n\n", ip, portno);

        }

        else if (input == 3){

            exit(0);

        }

        puts("");   //new line

    }

    return 0;

}

**Screenshots**

1. Server turns on – Prints the IP on terminal

Text

Description automatically generated

1. Client takes server IP and port

Text

Description automatically generated

1. Client connects to the server
   1. Connection successful

Text

Description automatically generated

* 1. Connection fail - Server off

Text

Description automatically generated

* 1. Connection fail – Wrong IP

Text

Description automatically generated

1. Server prints that client has connected

Text

Description automatically generated

1. Client requests file
   1. File exists

Text

Description automatically generated

* 1. File does not exist

Text

Description automatically generated

1. Server receives file request
   1. File exists

Text

Description automatically generated

* 1. File does not exist

Text

Description automatically generated

1. Client reconnects and requests another file with different extension

Text

Description automatically generated

Text

Description automatically generated

1. File data (to show data transfer)
   1. Server side

Text

Description automatically generated

* 1. Client side

A screenshot of a computer

Description automatically generated with medium confidence