

Karnataka Law Society's
GOGTE INSTITUTE OF TECHNOLOGY
UDYAMBAG, BELAGAVI – 590010
(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)
(Approved by AICTE, New Delhi)

Department of Computer Science and Engineering



A Course Activity Report

Submitted in the partial fulfillment for the academic requirement of

7th Semester B.E.
in

“Network Programming Lab”

Submitted By:

Name: ABDUL MALIK

USN: 2GI19CS400

GUIDE:
Prof. Naitik Suryavanshi

HOD:
Dr. Vijay S. Rajpurohit

2021– 2022

CERTIFICATE



This is to certify that the project entitled “**Implement simple file server using sockets**” is a bonafide record of the Project work done by ABDUL MALIK (2GI19CS400) under my supervision and guidance, in partial fulfillment of the requirements for the Outcome Based Education paradigm in Computer Science & Engineering from the Gogte Institute of Technology, Belgaum for the academic year 2021-2022.

Prof. Naitik Suryavanshi

Dr. Vijay S. Rajpurohit
Professor & Head
Dept. of Computer Science and Engg.

Place: KLS Gogte Institute of Technology, Belgaum.

Date: 03 – 01– 2022

Course Project Report

PROBLEM STATEMENT: Implement simple file server using sockets. The file server should be able to take the request from any client and return the requested file to client or return error message, status to client. Consider all the possible inputs for the file server. Implement using programming. Compare this result with FTP by using suitable tools.

	Batch No. :		
1.	Project Title: Implement simple file server using sockets.	Marks Range	USN
			2GI19CS400
2.	Problem statement (PO2)	0-1	
3.	Objectives of defined problem statement (PO1, PO2)	0-2	
4.	Design/algorithm/ flowchart/ methodology (PO3)	0-3	
5.	Implementation details /Function/ Procedures/ Classes and Objects (Language/Tools) (PO1, PO3, PO4, PO5)	0-4	
6.	Working model of the final solution (PO3, PO12)	0-5	
7.	Report and Oral presentation skill (PO9, PO10)	0-5	
	Total	20	

* 20 marks is converted to 10 marks for CGPA calculation

Table of contents

- **Introduction:**
- **Theory.**
- **Source Code .**
- **Output.**
- **Conclusion.**
- **Reference.**

Introduction:

TCP refers to the Transmission Control Protocol, which is a highly efficient and reliable protocol designed for end-to-end data transmission over an unreliable network.

A TCP connection uses a three-way handshake to connect the client and the server. It is a process that requires both the client and the server to exchange synchronization (SYN) and acknowledge (ACK) packets before the data transfer takes place.

Some important features of TCP:

- It's a connection-oriented protocol.
- It provides error-checking and recovery mechanisms.
- It helps in end-to-end communication.

Theory :

- Project structure

The project is divided into two files:

1. client.c
2. server.c

The client.c file contains the code for the client-side, which reads the text file and sends it to the server and the server.c file receives the data from the client and saves it in a text file.

• Client

The client performs the following functions.

1. Start the program
2. Declare the variables and structures required.
3. A socket is created and the connect function is executed.
4. The file is opened.
5. The data from the file is read and sent to the server.
6. The socket is closed.
7. The program is stopped.

• Server

The server performs the following functions.

1. Start the program.
2. Declare the variables and structures required.
3. The socket is created using the socket function.
4. The socket is bound to the specific port.
5. Start listening for the connections.

6. Accept the connection from the client.
7. Creates a child process to handle request client among multiple clients.
8. Close server socket descriptor
9. Create a new file.
10. Receives the data from the client.
11. Write the data into the file.
12. The program is stopped.

Source Code :

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include
<arpa/inet.h>#define
SIZE 1024
void write_file(int sockfd,char
    *outputFile){ int n;
    FILE *fp;
    char *filename =
    outputFile;char
    buffer[SIZE];
    fp = fopen(filename, "a");
    printf("\n Data sent to created output file is:
    ");while (1) {
        n = recv(sockfd, buffer, SIZE,
        0);printf("%s",buffer);
        if (n <= 0){
            break; }
        fprintf(fp, "%s",
        buffer);bzero(buffer,
        SIZE);
    }
    fclose(fp)
    ;return;
}
int main(int argc,
    char**argv){ char *ip =
    "127.0.0.1"; int port =
    8080;
```

```

int e;
int listenfd, connfd,
n;pid_t childpid;
socklen_t clilen;
int sockfd, new_sock;
struct sockaddr_in server_addr,
new_addr;socklen_t addr_size;
char buffer[SIZE];
sockfd = socket(AF_INET, SOCK_STREAM,
0);if(sockfd < 0) {
    perror("Error in
    socket");exit(1);
}

printf("Server socket created
successfully.\n");server_addr.sin_family =
AF_INET; server_addr.sin_port = port;
server_addr.sin_addr.s_addr = inet_addr(ip);
e = bind(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr));if(e < 0) {
    perror("Error in
    bind");exit(1);
}
printf("Binding
successfull.\n");
if(listen(sockfd, 10) == 0){
    printf("Listening ..\n");
}
else {
    perror("Error in
    listening");exit(1);
}

```

```

int
k=0;
for(;;)
{
    k++;
    addr_size = sizeof(new_addr);
    new_sock = accept(sockfd, (struct sockaddr*)&new_addr,
    &addr_size);if ( (childpid = fork ()) == 0 ) {
        printf ("\n\nChild created for dealing with client %d request",k);
        //close listening
        socketclose (listenfd);
        write_file(new_sock,argv[1]);
        printf("\nData written in the file successfully.\n");
    }
}
return 0;
}

```

Client.c

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include
<arpa/inet.h>#define
SIZE 1024
void send_file(FILE *fp, int
sockfd){ int n;
char data[SIZE] = {0};
while(fgets(data, SIZE, fp) != NULL)
{
    if (send(sockfd, data, sizeof(data), 0) == -1)
        { perror("Error in sending file.");
        exit(1);
        }
    bzero(data, SIZE);
}
}
int main(int argc, char**

```



```

argv){ char *ip =
"127.0.0.1"; int port =
8080;
int e;
int sockfd;
struct sockaddr_in
server_addr; FILE *fp;
char *filename = argv[1];
sockfd = socket(AF_INET,
SOCK_STREAM,0);
if(sockfd < 0) {
perror("Error in
socket");exit(1);
}
printf("Server socket created
successfully.\n");server_addr.sin_family =
AF_INET; server_addr.sin_port = port;
server_addr.sin_addr.s_addr = inet_addr(ip);
e = connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if(e == -1) {
perror("Error in
socket");exit(1); }
printf("Connected to
Server.\n");fp =
fopen(filename, "r");
if (fp == NULL) {
perror("Error in reading
file.");exit(1);
}
send_file(fp, sockfd);
printf("File data sent
successfully.\n");printf("Closing the
connection.\n");
close(sockfd)
;return 0;
}

```

Output:-

- Server contains two files named file1.txt and file2.txt

- File1.txt content:

A screenshot of a text editor window titled 'file1.txt' showing the following content:

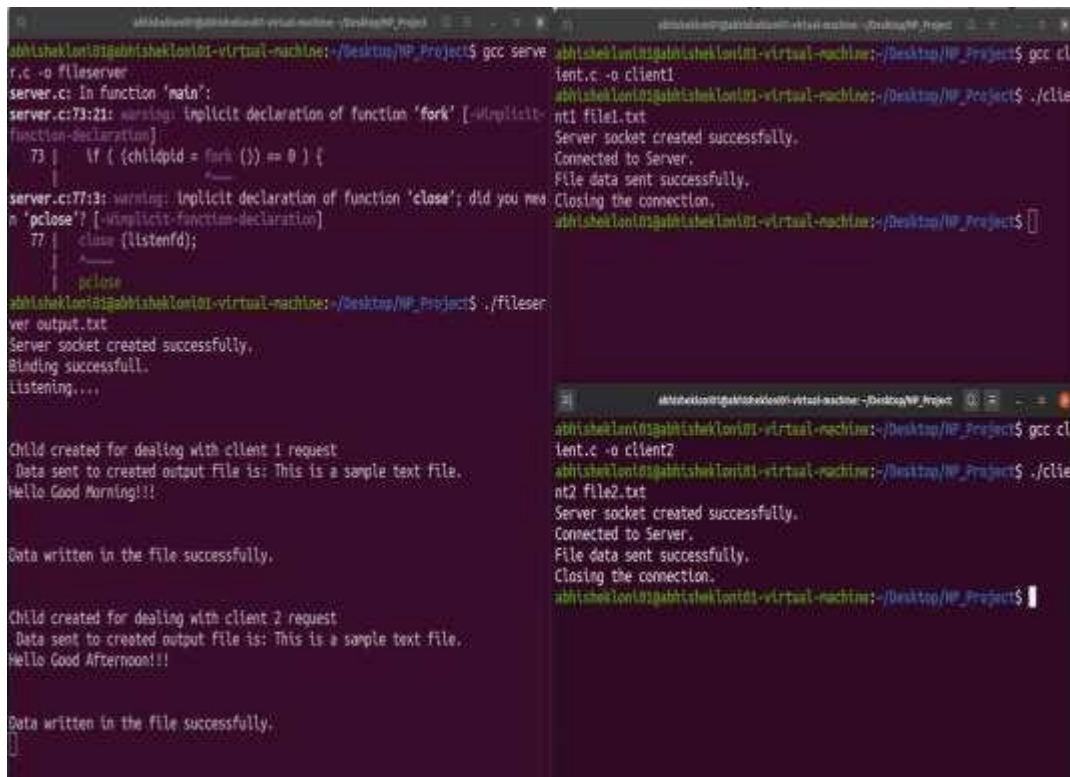
```
1 This is a sample text file.
2 Hello Good Morning!!!
3
```

- File2.txt content:

A screenshot of a text editor window titled 'file2.txt' showing the following content:

```
1 This is a sample text file.
2 Hello Good Afternoon!!!
3
```

➤ Communication Between File server and two clients:

A screenshot of a terminal window showing the execution of a C program for a file server and two clients. The server program is compiled and run, showing warnings about implicit function declarations for 'fork' and 'close'. It then listens for connections. Two client programs are run, each connecting to the server and sending data from file1.txt and file2.txt respectively. The server output shows the data being received and written to output files.

```
abhishek@abhishek-l01:~/Desktop/MP_Project$ gcc server.c -o fileserver
server.c: In function 'main':
server.c:73:21: warning: implicit declaration of function 'fork' [-Wimplicit-function-declaration]
   73 |     if ( (childpid = fork ( )) == 0 ) {
       |                     ^
server.c:77:3: warning: implicit declaration of function 'close'; did you mean 'pclose'? [-Wimplicit-function-declaration]
   77 |     close (listenfd);
       |     ^~~~~
   77 |     pclose
       |     ^~~~~
abhishek@abhishek-l01:~/Desktop/MP_Project$ ./fileserver
Server socket created successfully.
Binding successful.
Listening....

Child created for dealing with client 1 request
Data sent to created output File is: This is a sample text file.
Hello Good Morning!!!

Data written in the File successfully.

Child created for dealing with client 2 request
Data sent to created output File is: This is a sample text file.
Hello Good Afternoon!!!

Data written in the File successfully.
[]

abhishek@abhishek-l01:~/Desktop/MP_Project$ gcc client.c -o client1
abhishek@abhishek-l01:~/Desktop/MP_Project$ ./client1 file1.txt
Server socket created successfully.
Connected to Server.
File data sent successfully.
Closing the connection.
abhishek@abhishek-l01:~/Desktop/MP_Project$ []

abhishek@abhishek-l01:~/Desktop/MP_Project$ gcc client.c -o client2
abhishek@abhishek-l01:~/Desktop/MP_Project$ ./client2 file2.txt
Server socket created successfully.
Connected to Server.
File data sent successfully.
Closing the connection.
abhishek@abhishek-l01:~/Desktop/MP_Project$ []
```

- **Client1 appends its message into output.txt file from Server.**



A screenshot of a text editor window titled "output.txt" with a path of "~/Desktop/MP_Project". The editor contains three lines of text: "1 This is a sample text file.", "2 Hello Good Morning!!!", and "3". The cursor is positioned at the end of the third line.

- **Client2 appends its message into output.txt file from Server.**



A screenshot of a text editor window titled "output.txt" with a path of "~/Desktop/MP_Project". The editor contains five lines of text: "1 This is a sample text file.", "2 Hello Good Morning!!!", "3", "4 This is a sample text file.", and "5 Hello Good Afternoon!!!". The cursor is positioned at the end of the fifth line.

CONCLUSION:

In this project, we implemented File server using socket programming to handle multiple client requests to access files from server. We understood how Inter process communication works with socket programming and steps involved in communication. We also understood Concurrent Server concept to handle multiple client requests.

REFERENCES:

<https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/>