KARNATAKA LAW SOCIETY'S

GOGTE INSTITUTE OF TECHNOLOGY

UDYAMBAG, BELGAVI-590008

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

(APPROVED BY AICTE, NEW DELHI)

Department of Computer Science and engineering

2020-2021

Course Activity Report

Network Programming

Submitted in the partial fulfilment for the academic requirement of

7th Semester (CSE)

Submitted by

Jyotiprasad Patil – 2GI18CS059

GUIDE

Prof. Natik Suryavanshi

Prof. Sagar Pujar

Course Seminar report and ppt content

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.

Marks's allocation:

1.	Problem Statement	Marks	USN			
		Range	2GI18CS059			
2.	Abstract (PO2)	0-2				
3.	Application of the topic to the course (PO2)	0-3				
4.	Literature survey and its findings (PO2)	0-4				
5.	Methodology, Results and Conclusion (PO1,PO3,PO4)	0-6				
6.	Report and Oral presentation skill (PO9,PO10)	0-5				
	Total	20				

*** 20 marks is converted to 10 marks for CGPA calculation**

1.Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
2.Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and Engineering sciences.
3.Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6.The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need
for sustainable development.
8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9.Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.
10.Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

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.

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
2. Server

The client.c file contains the code for the client-side, which read 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 variable and structure required.

3.The socket is created using the socket function.

4.The socket is binded 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 socket close
                 (listenfd);
                 write_file(new_sock,argv[1]);
                 printf("\nData written in the file successfully.\n");

           }
  }
```

```
   return 0;
}
```

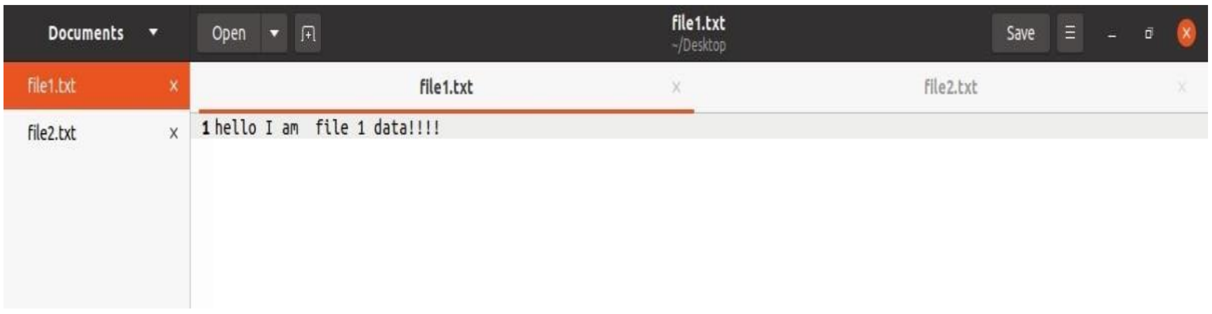
Client.c

```
#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;
}
```

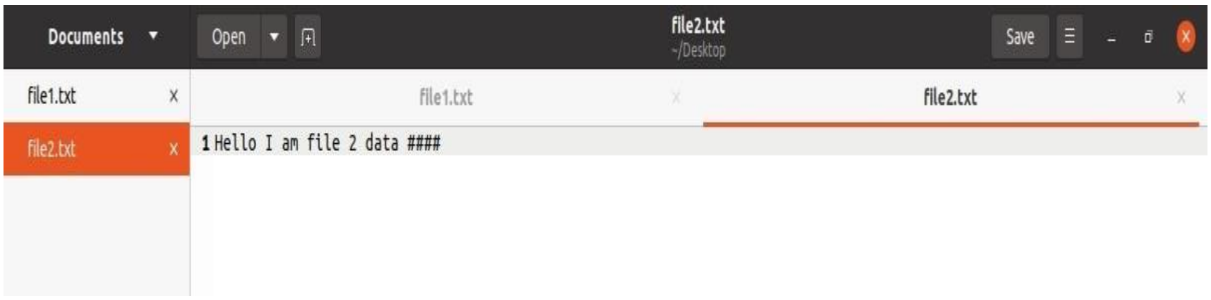
Output:

Sever containing files as file1.txt and file2.txt
File1.txt



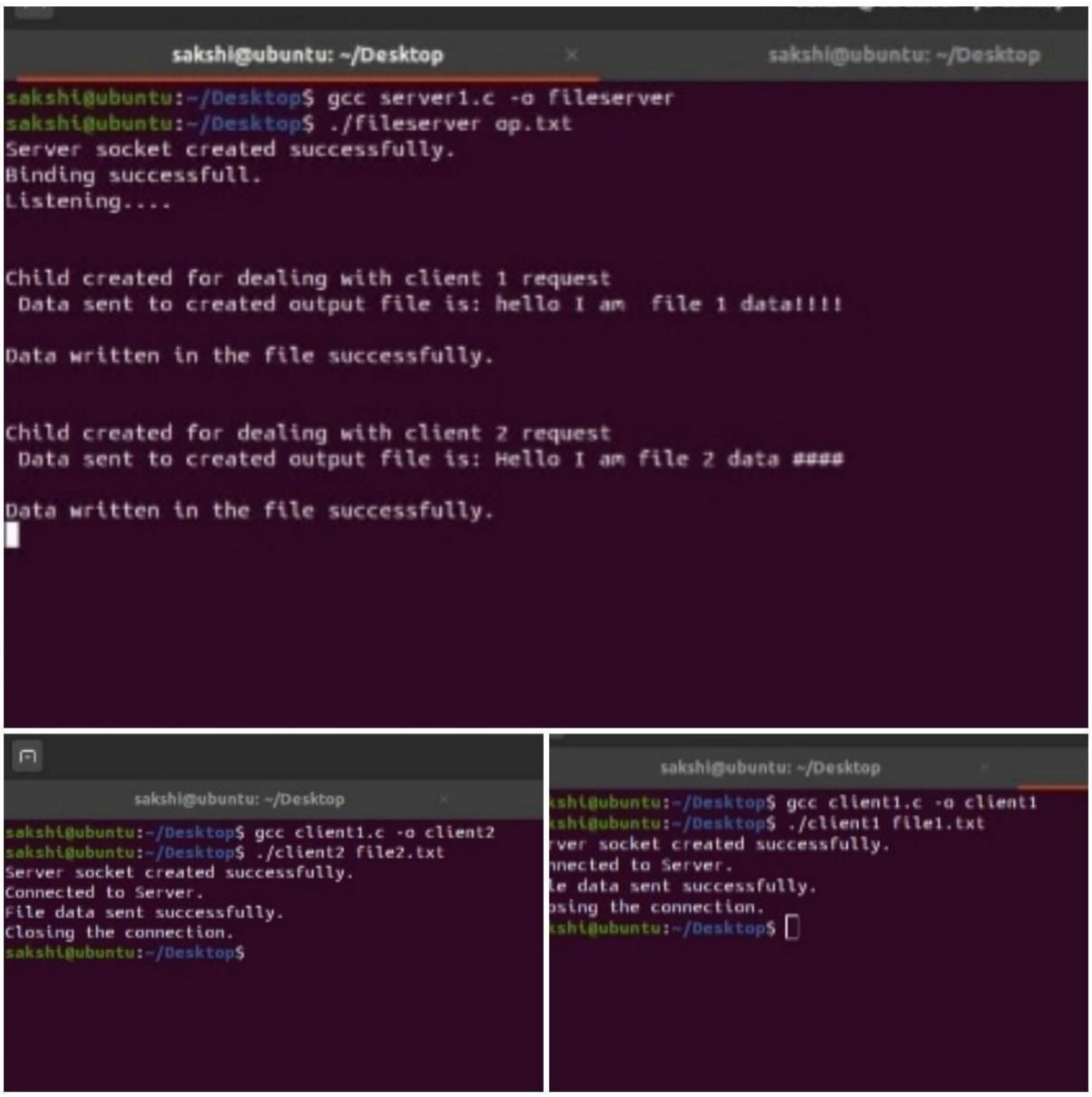
```
hello I am  file 1 data!!!!
```

File2.txt



```
Hello I am file 2 data ####
```

File Server and Two Clients (Client 1 and Client 2) execution.

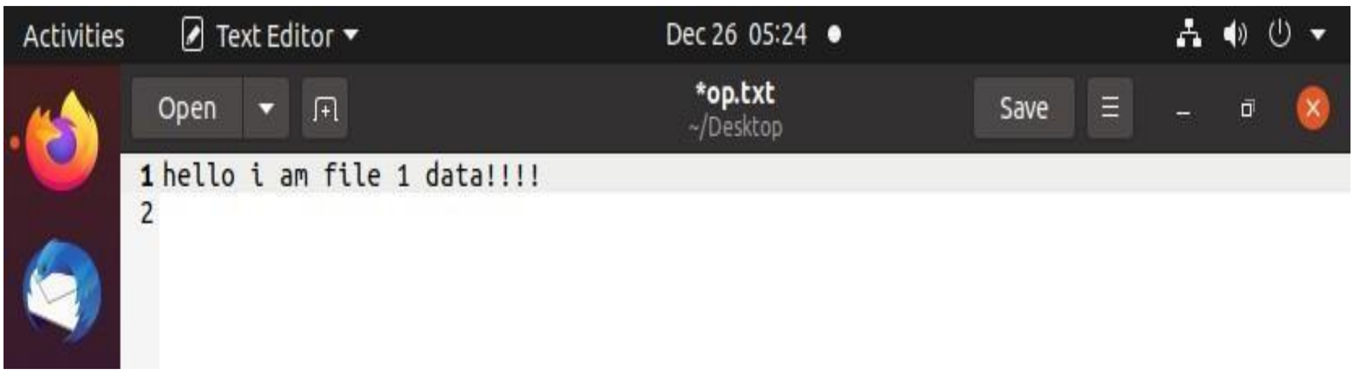


```
sakshi@ubuntu:~/Desktop$ gcc server1.c -o fileserver
sakshi@ubuntu:~/Desktop$ ./fileserver op.txt
Server socket created successfully.
Binding successfull.
Listening....


Child created for dealing with client 1 request
 Data sent to created output file is: hello I am  file 1 data!!!!

Data written in the file successfully.


Child created for dealing with client 2 request
 Data sent to created output file is: Hello I am file 2 data ####

Data written in the file successfully.
```

```
sakshi@ubuntu:~/Desktop$ gcc client1.c -o client2
sakshi@ubuntu:~/Desktop$ ./client2 file2.txt
Server socket created successfully.
Connected to Server.
File data sent successfully.
Closing the connection.
sakshi@ubuntu:~/Desktop$
```

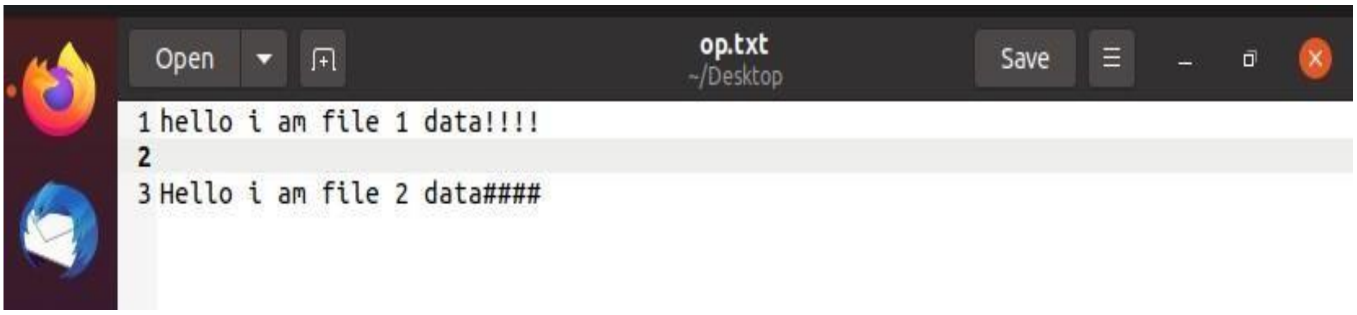
```
kshi@ubuntu:~/Desktop$ gcc client1.c -o client1
kshi@ubuntu:~/Desktop$ ./client1 file1.txt
rver socket created successfully.
nnected to Server.
le data sent successfully.
osing the connection.
kshi@ubuntu:~/Desktop$
```

Client1 appends its message into output.txt file from server.



```
hello i am file 1 data!!!!

```

Client2 appends its message into output.txt file from server



```
hello i am file 1 data!!!!

Hello i am file 2 data####
```

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/