KARNATAKA LAW SOCIETY'S

GOGTE INSTITUTE OF TECHNOLOGY

UDYAMBAG, BELGAVI-590008

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

(APPROVED BY AICTE, NEW DELHI)

Course Activity Report

Implementation of Simple File server using sockets.

Submitted in the partial fulfilment for the academic requirement of

7th Semester B.E

IN

Network Programming Lab

Submitted by

Slno	Name	USN
1	Aniruddh M	2GI8CS025

GUIDED BY

Dr. S.F.Rodd, Prof. Naitik.S.T
Prof. Department of Computer Science
KLS Gogte Institute of Technology, Belagavi

COURSE PROJECT REPORT (Academic Year 2020-21)

SUBJECT: Network Programming Lab

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.

SUBJECT CODE: 18CSL77 **Date: 24/12/2021**

Team Members Details:

Slno	Name	USN
1	Aniruddh M	2GI18CS025

Marks allocation:

	Batch No:					
1	Project Title: **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 Range	**USN**			
			2GI18CS025			
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				

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.c
2. server.c

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 variables and structures 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;
}
```

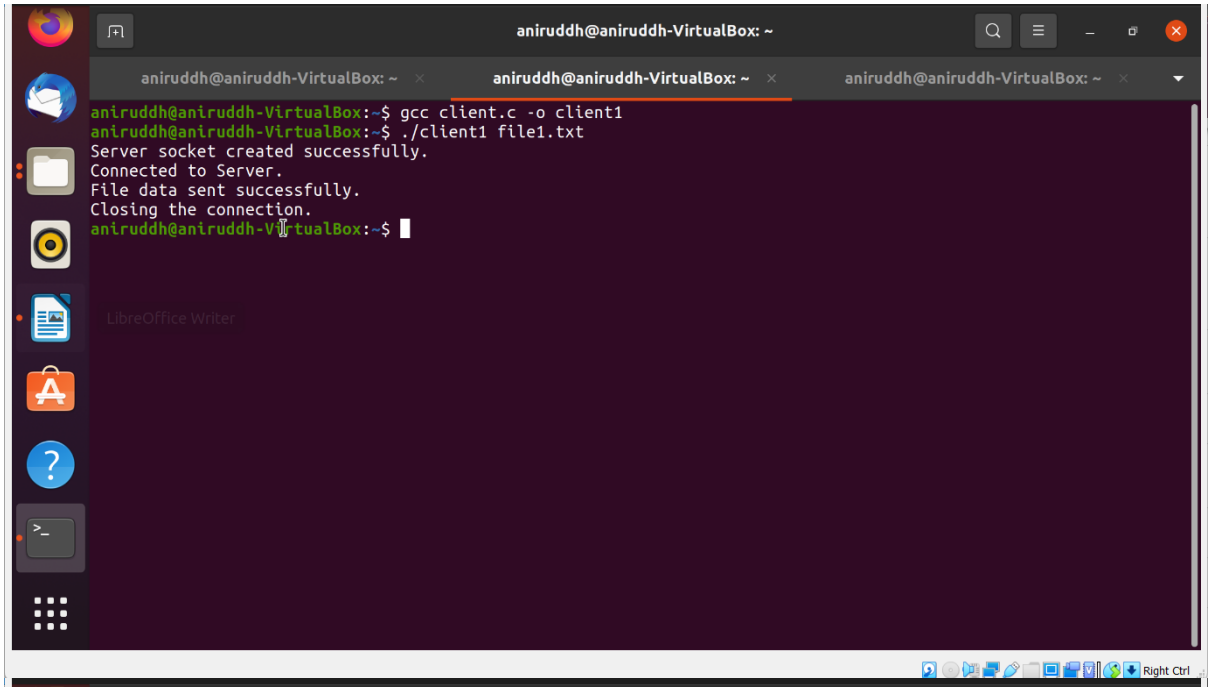
Client.c

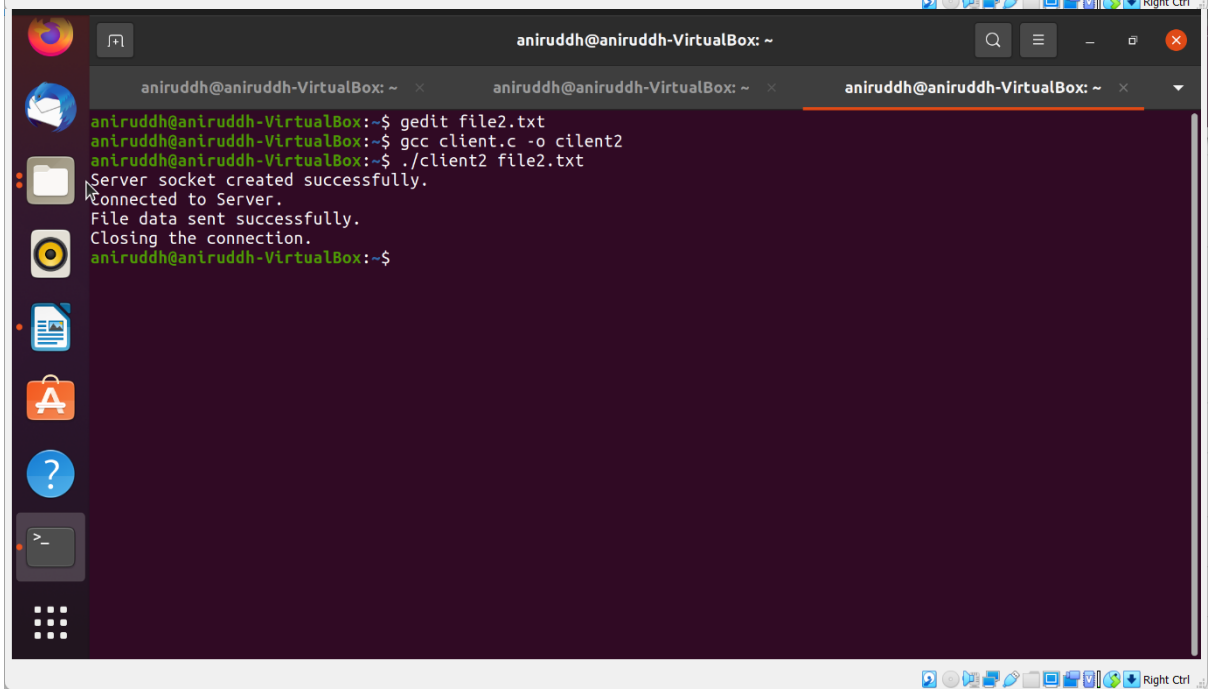
```
 #include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

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");
```

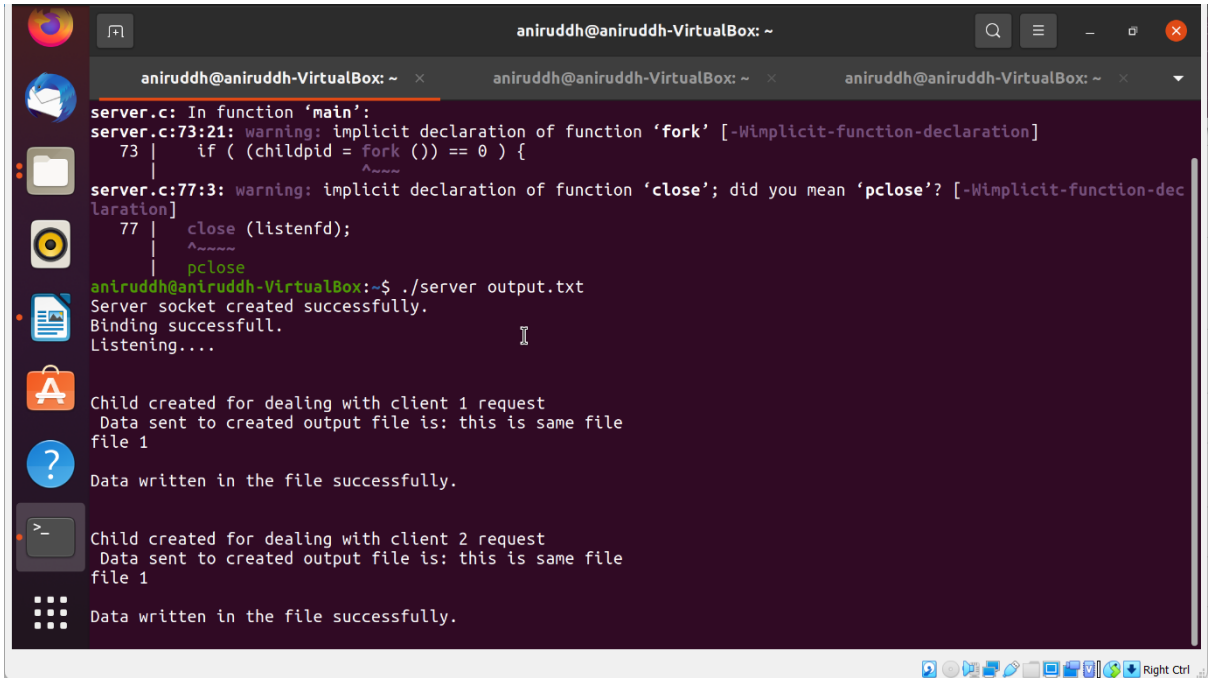
```
  fp = fopen(filename, "r");
  if (fp == NULL) {
    perror("Error in reading file.");
    exit(1);
  }

  send_file(fp, sockfd);
  printf("File data sent successfully.\n");

          printf("Closing the connection.\n");
  close(sockfd);
  return 0;
}
```

Output:-

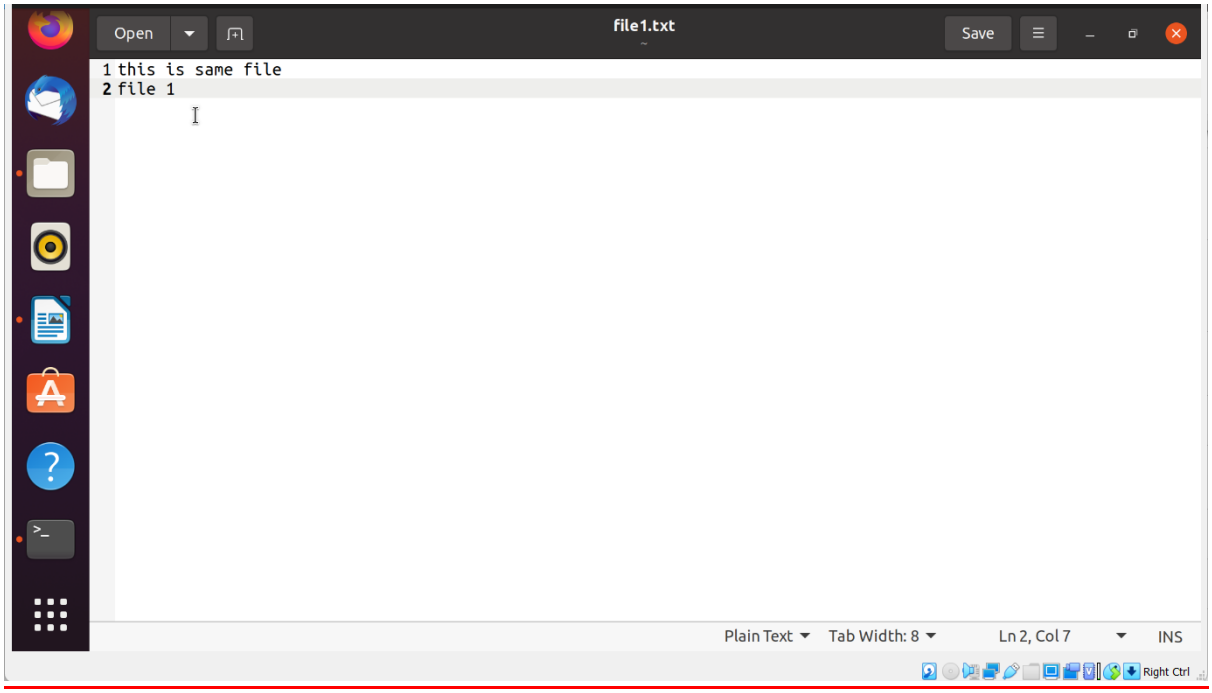
- **File Server and Two Clients(client1 and client2) execution.**



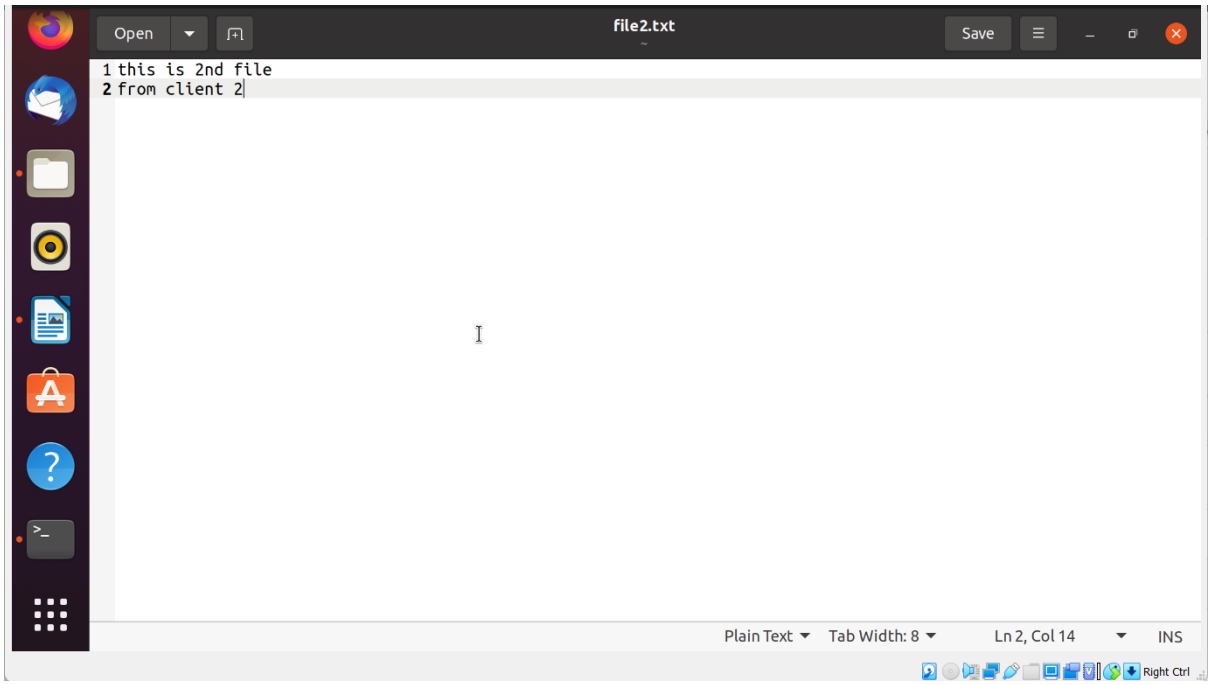


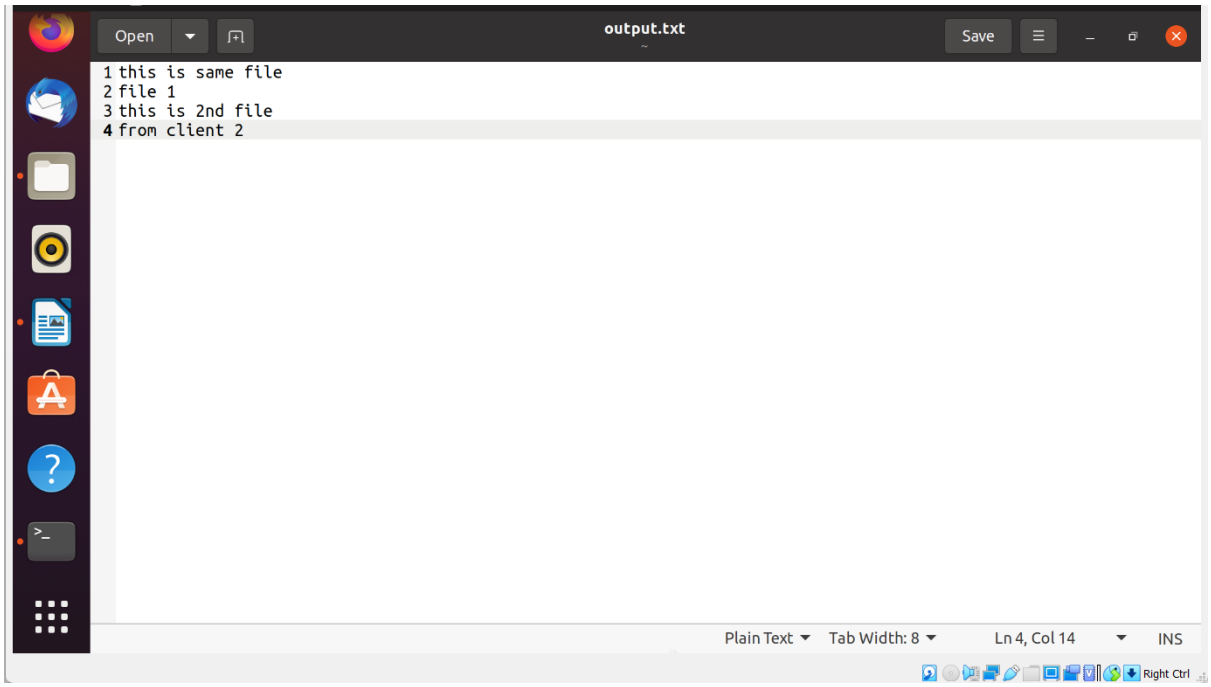


❖ Client1 appends its message into output.txt file from Server.



❖ Client2 appends its message into output.txt file from Server.





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/