NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA)

Department of Computer Science and Engineering

Academic Year: 2017-2018

Socket Programming Report on

‘MULTITHREADED CLIENT-SERVER FILESYSTEM’

Submitted by

ANKIT DATTA	**1NT15CS028**
HARSHITH NARAHARI	**1NT15CS064**
ROSHAN BADRINATH	**1NT15CS140**

Under the able guidance of

Mrs. Archana Naik

Associate Professor, Dept. of CSE

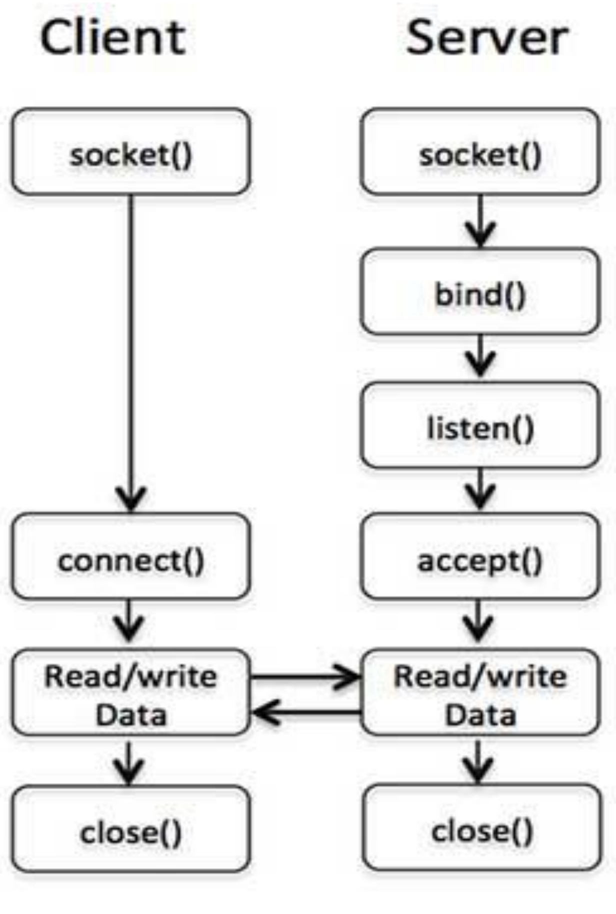
TABLE OF CONTENTS

INTRODUCTION TO SOCKET PROGRAMMING

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

A client is a desktop computer or workstation that is capable of obtaining information and applications from a server. A server is a computer program that provides a service to other computer programs (and its users). The client and server communicate by writing to or reading from their sockets. Every TCP connection can be uniquely identified by its two endpoints.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



MODULAR DESCRIPITON

The objective of the project is to share files. A client can either read the contents of a file or write to a file. We have used the concept of multithreading to connect multiple clients to the server the same time. Depending on the operation (read or write), access to file is given. To achieve synchronization, we have used the concept of semaphores. The following functions have been used to fulfill this objective:

1. socket() function:

This function is used to create a socket. Its general syntax looks as shown below:
int sockfd = socket(domain, type, protocol);

where:

sockfd: is a socket descriptor, an integer (like a file-handle)

domain: is an integer which refers to the communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: is the communication type. It can take the following values:

SOCK_STREAM: for TCP (which is a connection oriented protocol and offers reliable services)

SOCK_DGRAM: for UDP (which is a connectionless protocol and offers unreliable services)

protocol: refers to the protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.

2. bind() function:

After creation of the socket, the bind function binds the socket to the address and port number specified in (custom data structure). In the code, we bind the server to the local host, hence we use INADDR_ANY to specify the IP address. Its general syntax is:

*int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

3. listen() function:

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED. Its general syntax is:

int listen(int sockfd, int backlog);

4. accept() function:

This function extracts the first connection request on the queue of pending connections for the listening socket, ‘sockfd’, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data. Its general syntax is:

*int new_socket = accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);*

5. connect() function:

The connect() system call connects the socket referred to by the file descriptor 'sockfd' to the address specified by 'addr'. The server’s address and port is specified in 'addr'. Its general syntax is:

*int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);*

6. send() function:

The send() function shall initiate transmission of a message from the specified socket to its peer. The send() function shall send a message only when the socket is connected. Its general syntax is:

*ssize_t send(int socket, const void *buffer, size_t length, int flags);*

where:

socket: Specifies the socket file descriptor.

buffer: Points to the buffer containing the message to send.

length: Specifies the length of the message in bytes.

flags: Specifies the type of message transmission.

7. recv() function:

The recv() system call is used to receive messages from a socket. Its general syntax is:

*ssize_t recv(int sock_fd, void *buf, size_t len, int flags);*

8. memset()

Copies character c to the first n characters of the string.

*memset(void *str, int c, size_t n);*

9. inet_pton()

inet_pton() also takes an address family (either AF_INET or AF_INET6) in the af parameter. The src parameter is a pointer to a string containing the IP address in printable form. Lastly the dst parameter points to where the result should be stored, which is probably a struct in_addr or struct in6_addr.

*int inet_pton(int af, const char *src, void *dst);*

10. setsocketopt()

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

*int setsocketopt(int socket, int level, int opt_name, const void *opt_value, int opt_len);*

11. pthread_create() function:

The pthread_create() function creates a new thread, with attributes specified by attr, within a process. If attr is NULL, the default attributes shall be used. If the attributes specified by attr are modified later, the thread's attributes shall not be affected. Upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by thread. . Its general syntax is:

*int pthread_create(pthread_t *restrict tid, const pthread_attr_t *restrict tattr,*
void(*start_routine)(void *), void *restrict arg);*

12. sem_wait() function:

The sem_wait() function locks the semaphore referenced by sem by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread will not return from the call to sem_wait() until it either locks the semaphore or the call is interrupted by a signal. Its general syntax is:

*int sem_wait(sem_t *sem);*

13. sem_post() function:

The sem_post() increments (unlocks) the semaphore pointed to by sem.If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait() call will be woken up and proceed to lock the semaphore.

*int sem_wait(sem_t *sem);*

14. sem_init() function:

The sem_init() function is used to initialise the unnamed semaphore referred to by sem. The value of the initialised semaphore is value. Following a successful call to sem_init(), the semaphore may be used in subsequent calls to sem_wait(), sem_post(), and sem_destroy(). This semaphore remains usable until the semaphore is destroyed.

*int sem_init(sem_t *sem, int pshared, unsigned int value);*

SERVER PROGRAM CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>

static sem_t rd[5], wrt[5];
static int readcnt[5];

void *server(void *tskt) {

        int skt = *(int *) tskt;
        char buffer[1024] = { 0 };
        char op[10], c[2], *filename;
        filename = (char *) malloc(20 * sizeof(char));
        char msg[100];
        int j;

        memset(c, 0, 1024);
        recv(skt, c, 1024, 0);
        j = atoi(c);

        switch (j) {
        case 1:
                sprintf(filename, "%s.txt", "Dir1/File1");
                break;
        case 2:
                sprintf(filename, "%s.txt", "Dir1/File2");
                break;
        case 3:
                sprintf(filename, "%s.txt", "Dir2/File1");
                break;
        case 4:
                sprintf(filename, "%s.txt", "Dir2/File2");
                break;
        }

        memset(op, 0, 1024);
        recv(skt, op, 1024, 0);
        printf("%s\n", op);
```

```
if (strcmp(op, "read") == 0) {

        sem_wait(&rd[j]);
        readcnt[j]++;
        if (readcnt[j] == 1)
                sem_wait(&wrt[j]);
        sem_post(&rd[j]);

        strcpy(msg, "Reading Mode");
        send(skt, msg, strlen(msg), 0);

        FILE *fp;
        fp = fopen(filename, "r");
        memset(buffer, 0, 1024);
        char ch;
        int i = 0;
        while ((ch = fgetc(fp)) != EOF) {
                buffer[i++] = ch;
                buffer[i] = '\0';
        }
        fclose(fp);
        printf("File Contents Sent\n");
        send(skt, buffer, strlen(buffer), 0);

        memset(buffer, 0, 1024);
        recv(skt, buffer, 1024, 0);
        printf("%s\n", buffer);

        sem_wait(&rd[j]);
        readcnt[j]--;
        if (readcnt[j] == 0)
                sem_post(&wrt[j]);
        sem_post(&rd[j]);
        printf("End Read\n");

        strcpy(msg, "Close");
        send(skt, msg, strlen(msg), 0);

} else if (strcmp(op, "write") == 0) {

        sem_wait(&wrt[j]);

        strcpy(msg, "Writing Mode");
```

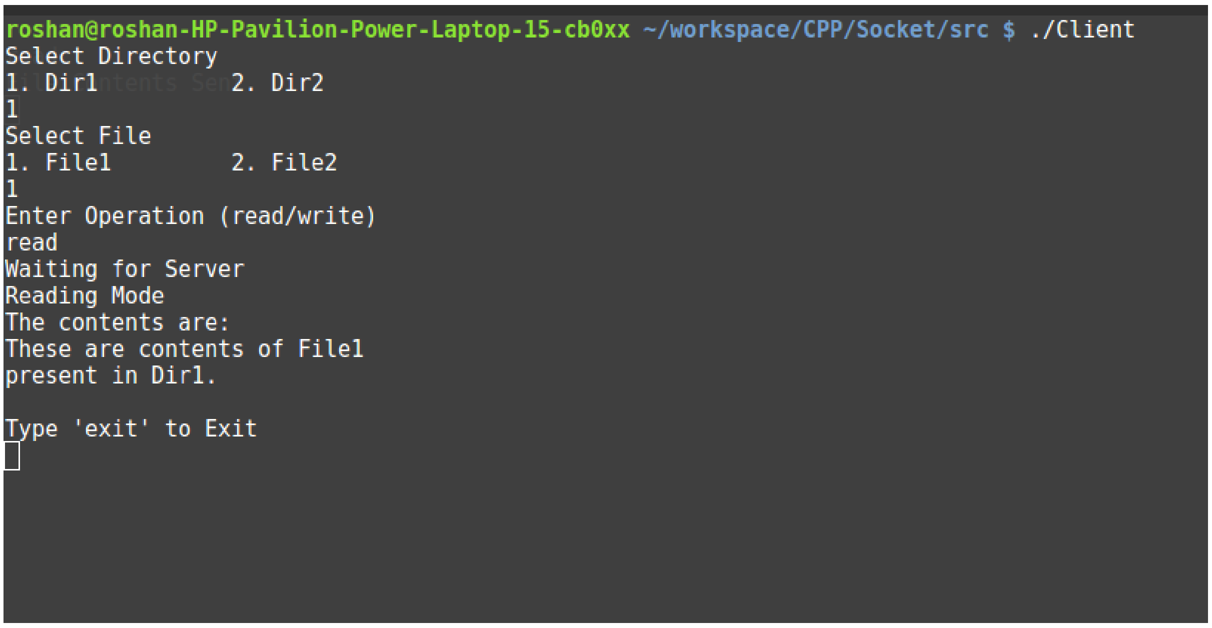
```
                send(skt, msg, strlen(msg), 0);

                memset(buffer, 0, 1024);
                recv(skt, buffer, 1024, 0);
                printf("%s\n", buffer);
                FILE *fp;
                fp = fopen(filename, "w");
                fprintf(fp, "%s", buffer);
                fclose(fp);

                sem_post(&wrt[j]);
                printf("End Write\n");

        } else {

        }

        return 0;
}

void init() {
        int i;
        for (i = 0; i < 5; i++) {
                readcnt[i] = 0;
                sem_init(&wrt[i], 1, 1);
                sem_init(&rd[i], 1, 1);

        }
}

int main() {

        int server_fd, skt, readmsg;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);
        pthread_t t;
        init();

        server_fd = socket(AF_INET, SOCK_STREAM, 0);

        setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
                        sizeof(opt));
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons(5000);

        bind(server_fd, (struct sockaddr *) &address, sizeof(address));
        listen(server_fd, 3);

        while (1) {
                skt = accept(server_fd, (struct sockaddr *) &address,
                                (socklen_t*) &addrlen);
                pthread_create(&t, NULL, server, (void *) &skt);
        }

        return 1;
}
```

CLIENT PROGRAM CODE

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>

int main(int argc, char const *argv[]) {
        struct sockaddr_in address;
        int sock = 0, readmsg;
        struct sockaddr_in serv_addr;
        char wrtmsg[10];
        char buffer[1024] = { 0 };
        sock = socket(AF_INET, SOCK_STREAM, 0);

        char msg[100];
        int d, f;
        char c[2];
        memset(&serv_addr, '0', sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(5000);

        inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);
        connect(sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr));

        a: printf("Select Directory\n");
        printf("1. Dir1 \t 2. Dir2\n");
        scanf("%d", &d);
        printf("Select File\n");
        printf("1. File1 \t 2. File2\n");
        scanf("%d", &f);

        switch (d) {
        case 1:
                switch (f) {
                case 1:
                        c[0] = '1';
                        c[1] = '\0';
                        break;
                case 2:
                        c[0] = '2';
                        c[1] = '\0';
                        break;
        default:
            printf("Invalid Entry");
            goto a;
            break;
        }
        break;

    case 2:
        switch (f) {
        case 1:
            c[0] = '3';
            c[1] = '\0';
            break;
        case 2:
            c[0] = '4';
            c[1] = '\0';
            break;

        default:
            printf("Invalid Entry");
            goto a;
            break;
        }
        break;

    default:
        printf("Invalid Entry");
        goto a;
        break;
    }

    send(sock, c, strlen(c), 0);
    c: printf("Enter Operation (read/write)\n");
    scanf("%s", wrtmsg);
    send(sock, wrtmsg, strlen(wrtmsg), 0);

    if (strcmp(wrtmsg, "read") == 0) {

        printf("Waiting for Server\n");

        memset(buffer, 0, 1024);
        recv(sock, buffer, 1024, 0);
        printf("%s\n", buffer);
```

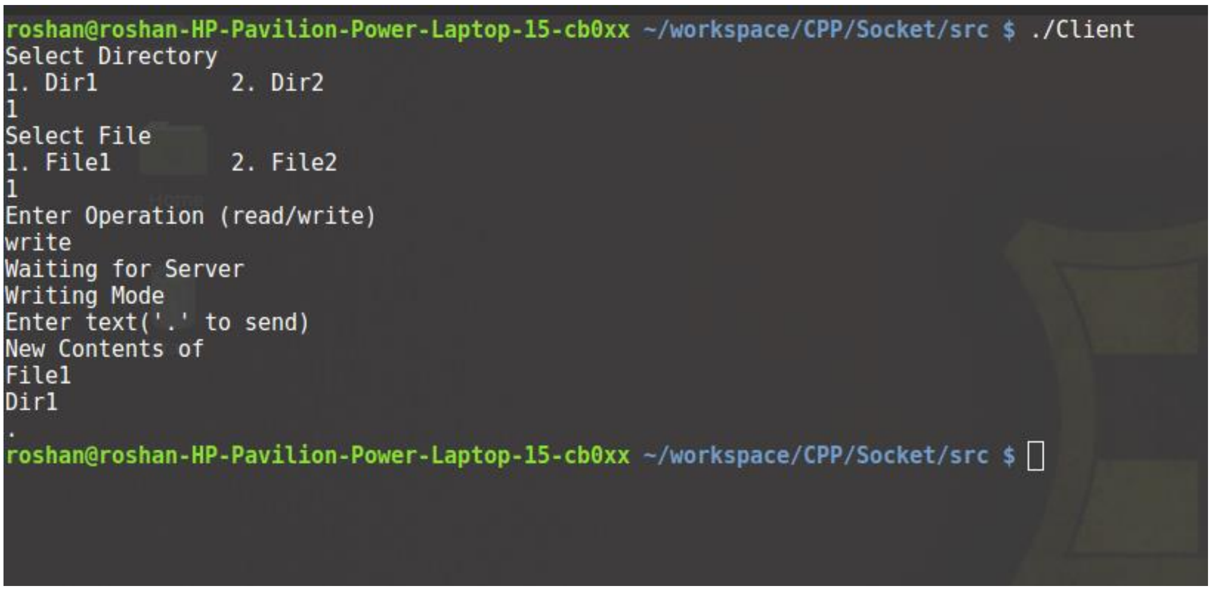
```
		printf("The contents are:\n");
		memset(buffer, 0, 1024);
		recv(sock, buffer, 1024, 0);
		printf("%s\n", buffer);

		b: printf("Type 'exit' to Exit\n");
		scanf("%s", msg);
		if (strcmp(msg, "exit") != 0)
			goto b;
		send(sock, msg, strlen(msg), 0);

		memset(buffer, 0, 1024);
		recv(sock, buffer, 1024, 0);
		printf("%s\n", buffer);

	} else if (strcmp(wrtmsg, "write") == 0) {

		printf("Waiting for Server\n");

		memset(buffer, 0, 1024);
		recv(sock, buffer, 1024, 0);
		printf("%s\n", buffer);

		memset(buffer, 0, 1024);
		memset(msg, 0, 100);
		printf("Enter text('.' to send)\n");
		getchar();
		scanf("%[^\n]", msg);
		while (strcmp(msg, ".") != 0) {
			strcat(buffer, msg);
			strcat(buffer, "\n");
			memset(msg, 0, 100);
			getchar();
			scanf("%[^\n]", msg);
		}

		send(sock, buffer, strlen(buffer), 0);

	} else {
		goto c;
	}

	return 0;
}
```

OUTPUT SCREENSHOTS



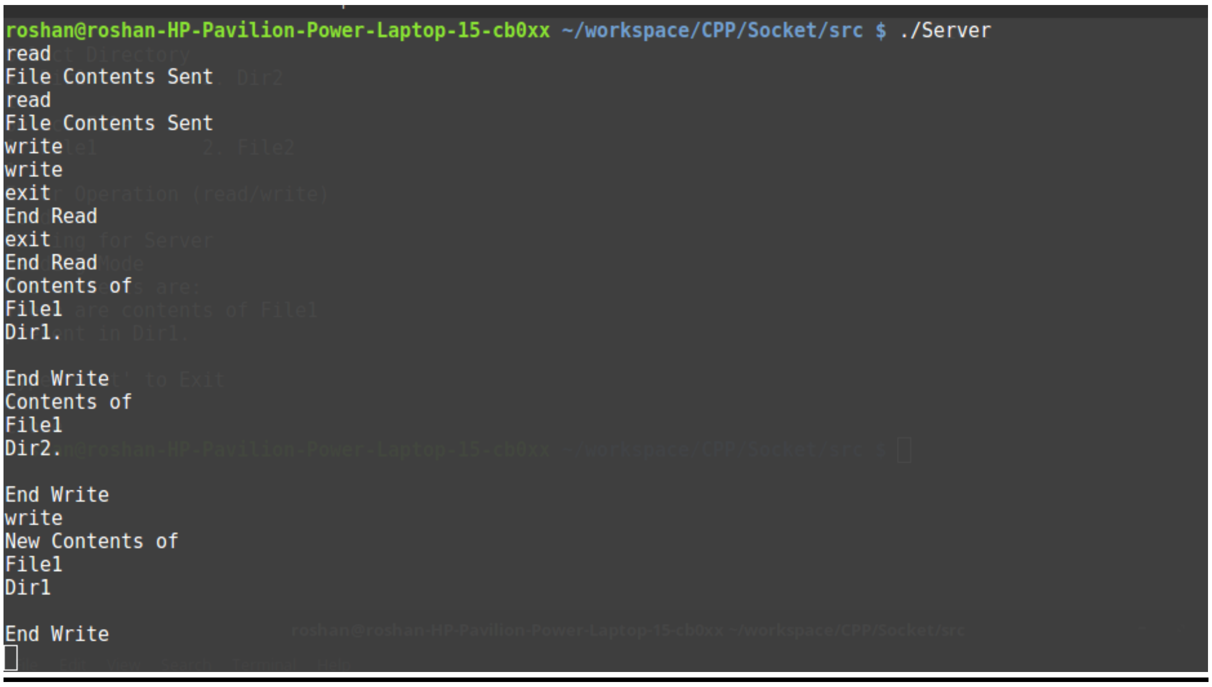
```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1           2. Dir2
1
Select File
1. File1          2. File2
1
Enter Operation (read/write)
read
Waiting for Server
Reading Mode
The contents are:
These are contents of File1
present in Dir1.

Type 'exit' to Exit
```

Fig 1. Output screen of the Client reading the contents of a file.



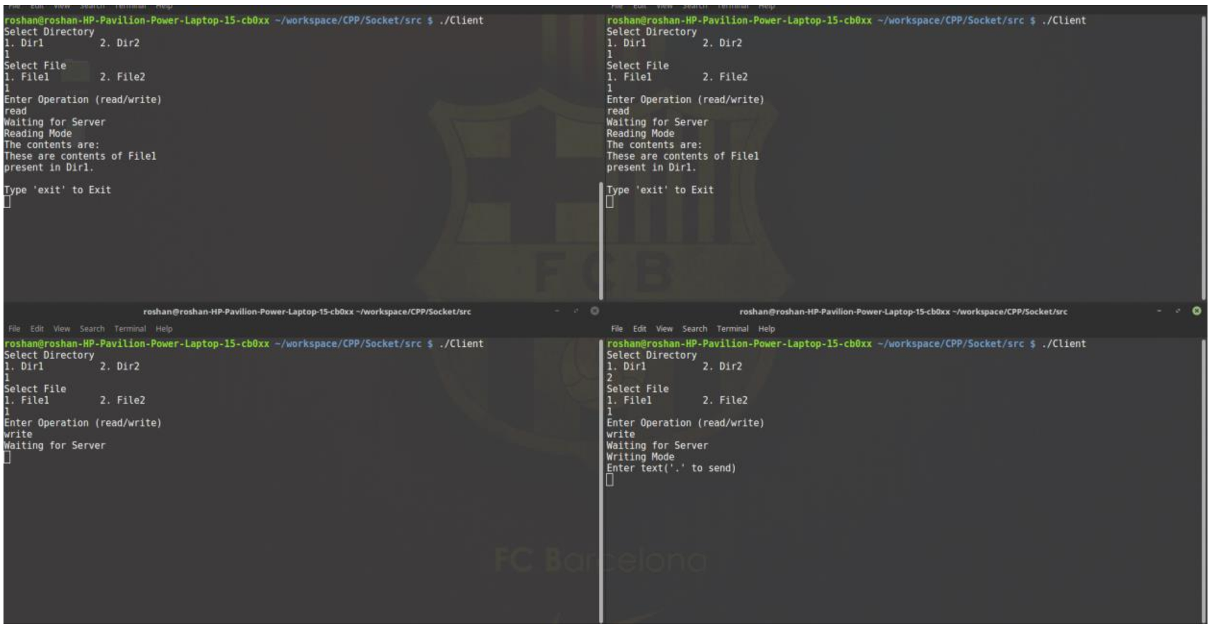
```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1           2. Dir2
1
Select File
1. File1          2. File2
1
Enter Operation (read/write)
write
Waiting for Server
Writing Mode
Enter text('.' to send)
New Contents of
File1
Dir1
.
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $
```

Fig 2. Output screen of the Client writing into a file.



```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Server
read
File Contents Sent
read
File Contents Sent
write
write
exit
End Read
exit
End Read
Contents of
File1
Dir1.

End Write
Contents of
File1
Dir2.

End Write
write
New Contents of
File1
Dir1

End Write
```

Fig 3. Output screen of the Server.



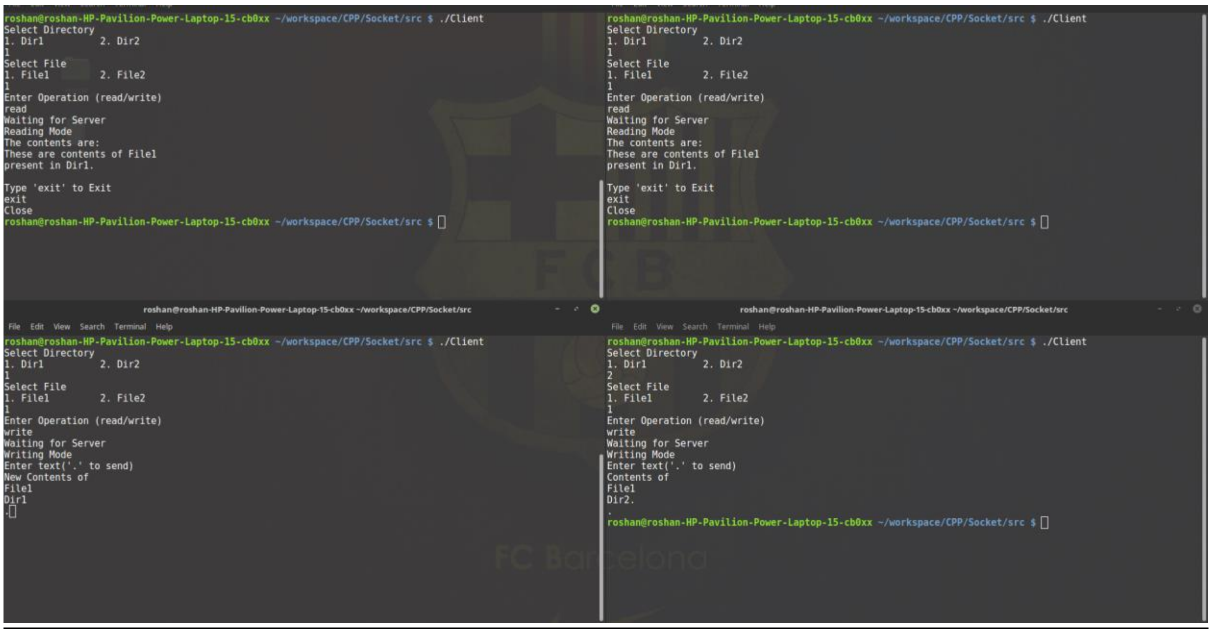
```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1         2. Dir2
1
Select File
1. File1        2. File2
1
Enter Operation (read/write)
read
Waiting for Server
Reading Mode
The contents are:
These are contents of File1
present in Dir1.

Type 'exit' to Exit
```

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1         2. Dir2
1
Select File
1. File1        2. File2
1
Enter Operation (read/write)
read
Waiting for Server
Reading Mode
The contents are:
These are contents of File1
present in Dir1.

Type 'exit' to Exit
```

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1         2. Dir2
1
Select File
1. File1        2. File2
1
Enter Operation (read/write)
write
Waiting for Server
```

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1         2. Dir2
2
Select File
1. File1        2. File2
1
Enter Operation (read/write)
write
Waiting for Server
Writing Mode
Enter text('.' to send)
```

Fig 4. Output screen of the 4 clients accessing server at the same time. Here, a client who wants to write into Dir1/File1.txt is waiting for other clients who are reading to exit. The 4th client can write into Dir2/File1.txt as there are no readers.


roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src
File Edit View Search Terminal Help
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src
File Edit View Search Terminal Help

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1          2. Dir2
1
Select File
1. File1         2. File2
1
Enter Operation (read/write)
read
Waiting for Server
Reading Mode
The contents are:
These are contents of File1
present in Dir1.

Type 'exit' to Exit
exit
Close
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ 
```

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1          2. Dir2
1
Select File
1. File1         2. File2
1
Enter Operation (read/write)
read
Waiting for Server
Reading Mode
The contents are:
These are contents of File1
present in Dir1.

Type 'exit' to Exit
exit
Close
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ 
```

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1          2. Dir2
1
Select File
1. File1         2. File2
1
Enter Operation (read/write)
write
Waiting for Server
Writing Mode
Enter text('.' to send)
New Contents of
File1
Dir1
.
```

```
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ ./Client
Select Directory
1. Dir1          2. Dir2
2
Select File
1. File1         2. File2
1
Enter Operation (read/write)
write
Waiting for Server
Writing Mode
Enter text('.' to send)
Contents of
File1
Dir2.
.
roshan@roshan-HP-Pavilion-Power-Laptop-15-cb0xx ~/workspace/CPP/Socket/src $ 
```

Fig 5. Output screen of the 4 clients accessing server at the same time. Here, the client who wants to write into Dir1/File1.txt can proceed to write as other clients who were reading the same file have completed reading it.

BIBLIOGRAPHY

1. TCP/IP Protocol Suite - by Behrouz A. Forouzan

2. https://www.geeksforgeeks.org/socket-programming-cc/

3. https://www.tutorialspoint.com/unix_sockets/