NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

Department of Computer Engineering

COMPUTER NETWORKS

41 CE-B

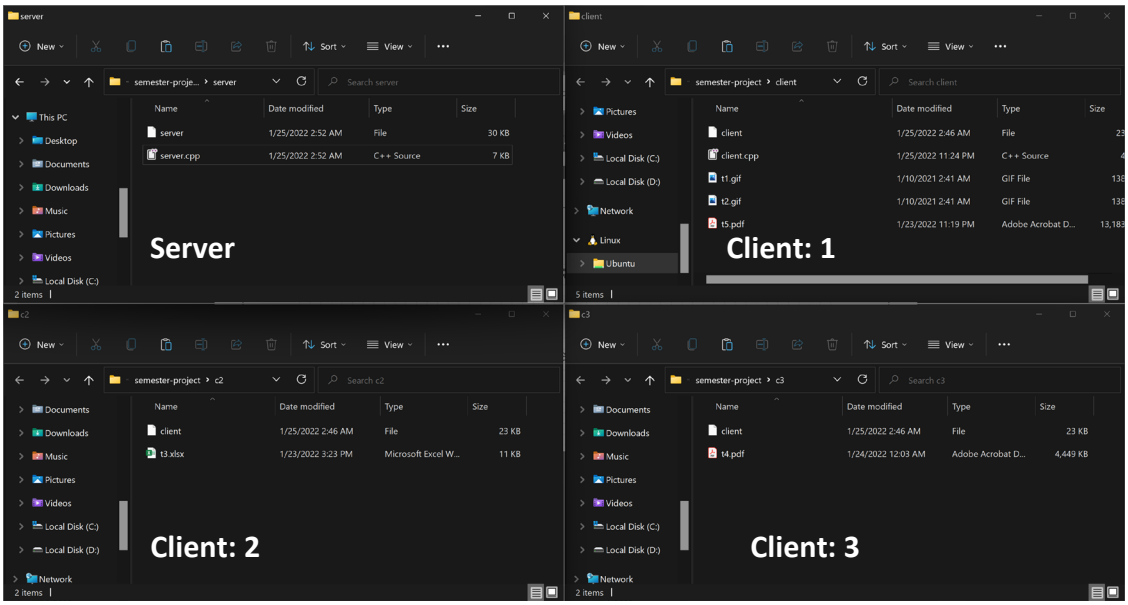
SEMESTER PROJECT

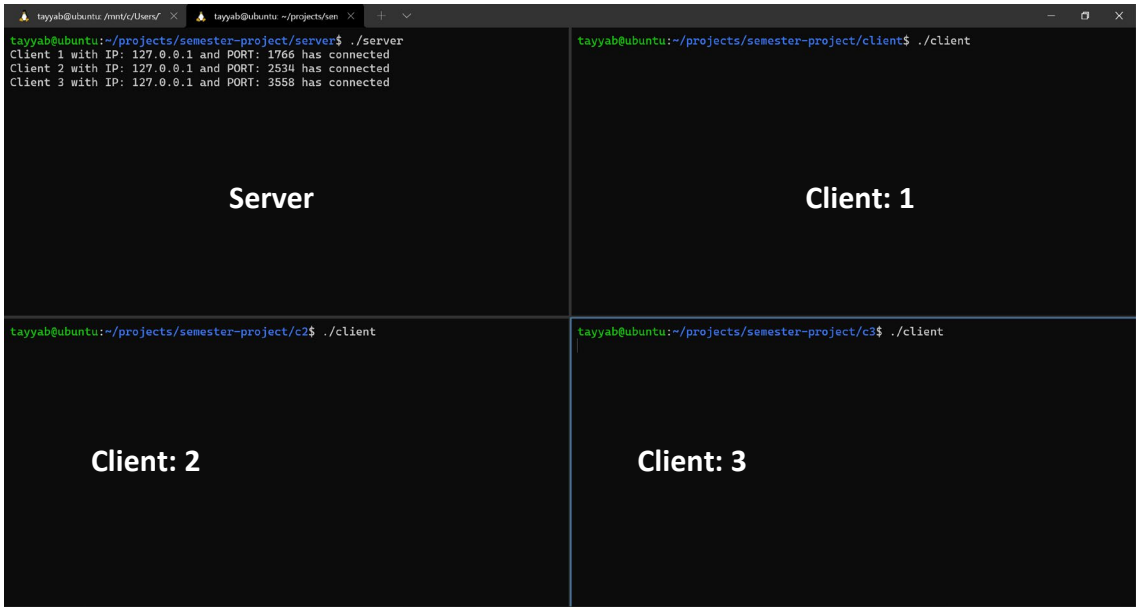
Name	Muhammad Tayyab Rashid	325735
	Abdul Rafay	282912
	Fatima Ahmad	310560
Submitted To	Sir Umar Farooq	

SEMESTER PROJECT

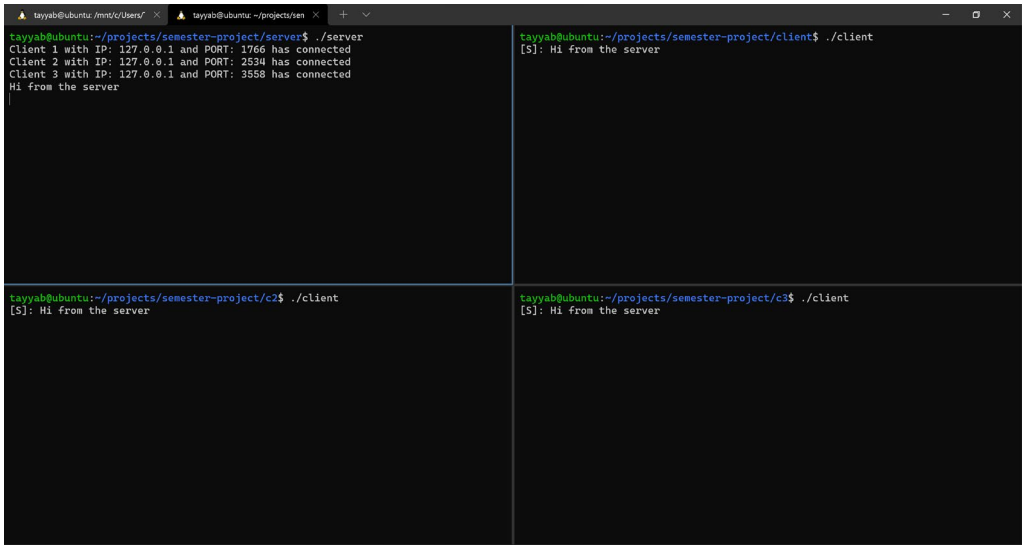
A single TCP server with three connected clients with each client having a different file to send to other clients.

Whenever a message is sent by a client, it first goes to the server which then relays that message to all the connected clients. Same is the case with file sharing. A client sends a file, it is then received by the server which then sends that same file to all other connected clients. The server can also broadcast its own messages to other clients.

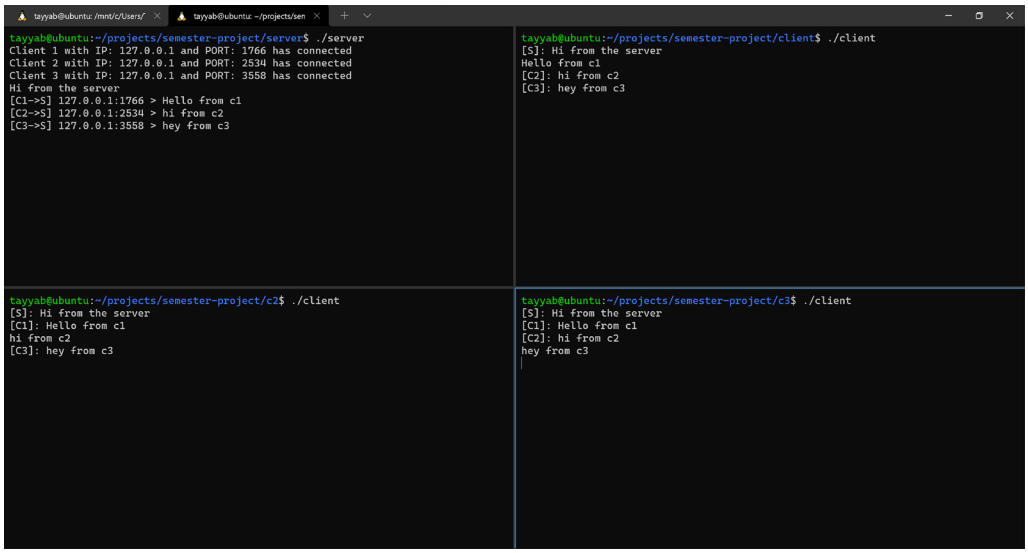




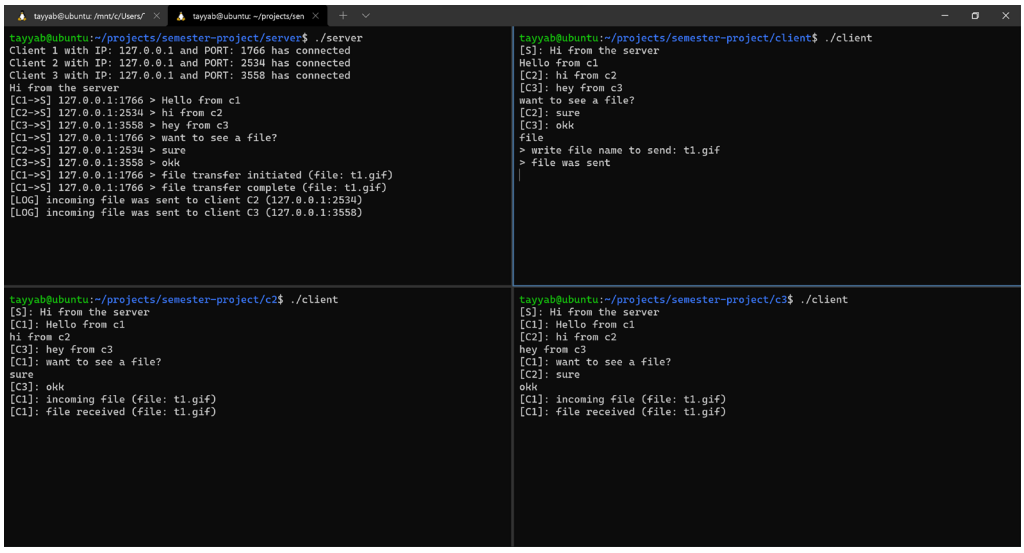
A server with three clients connected.



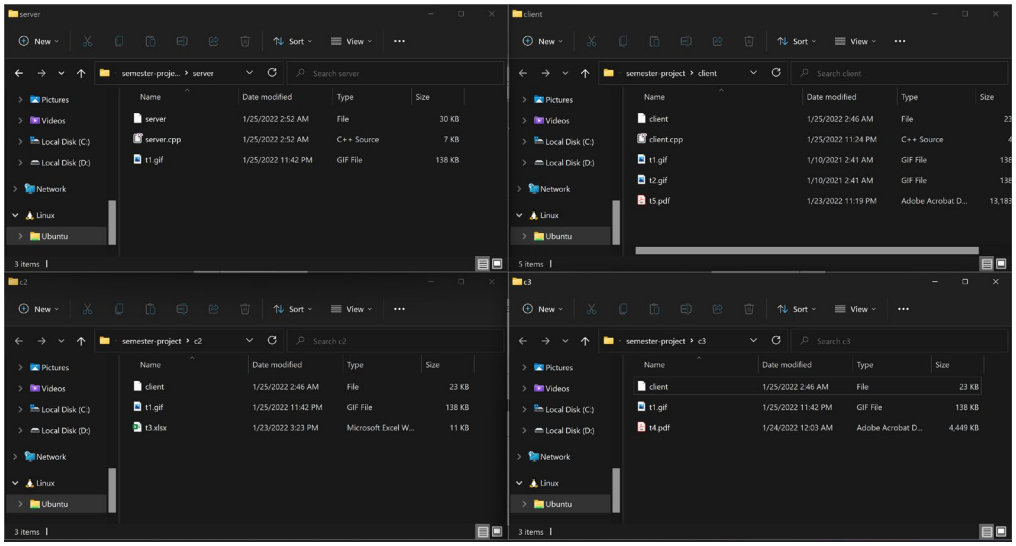
Server sends a message to all connected clients.



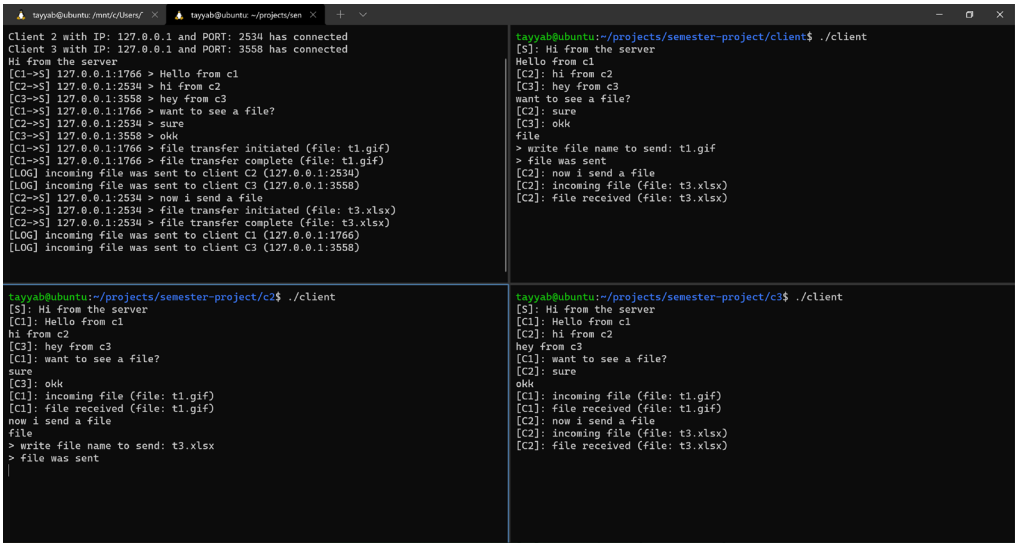
A muti-party chat application where all clients are communicating with each other.



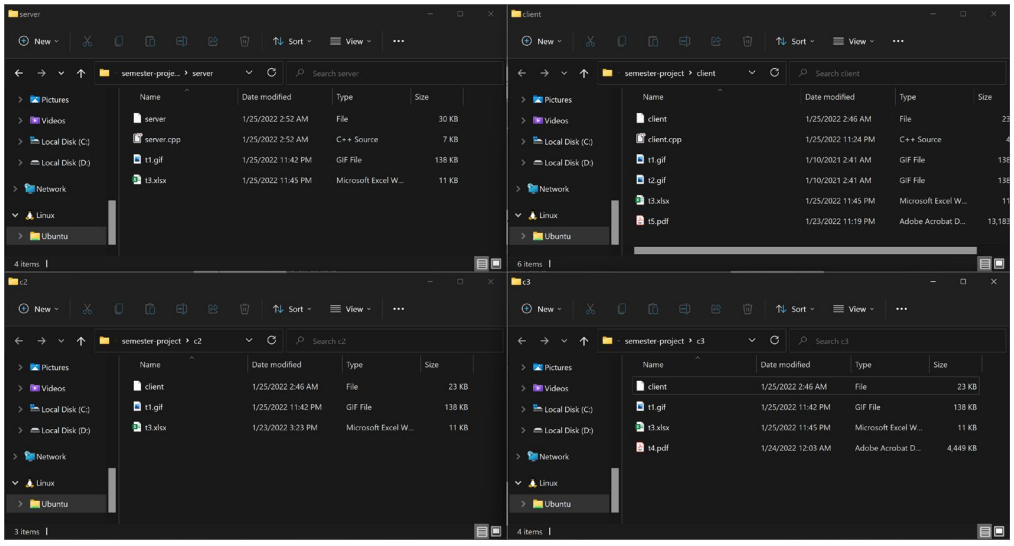
Client 1 sends a file (t1.gif).



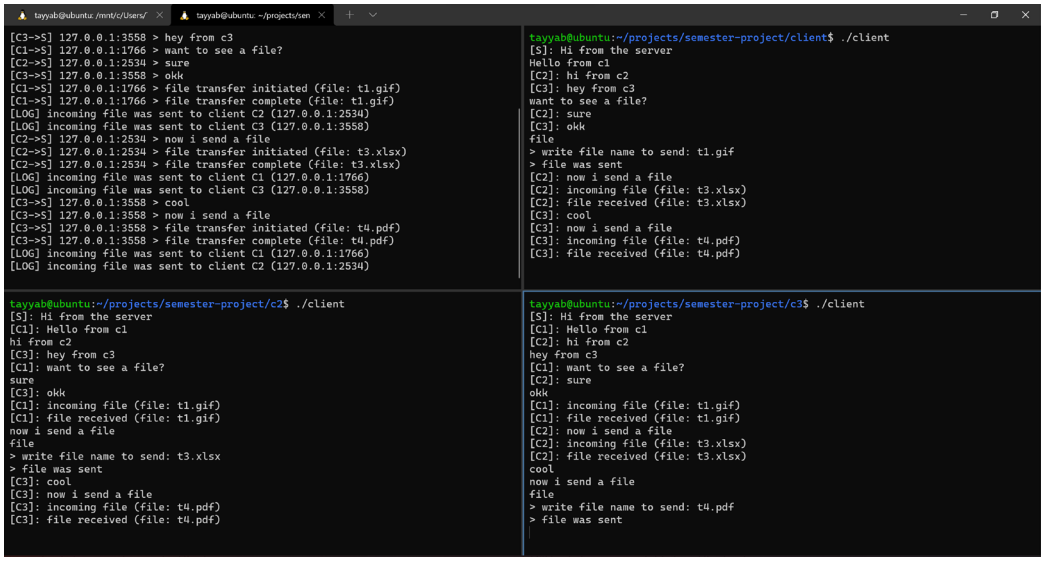
File is received by the server and all the connected clients.



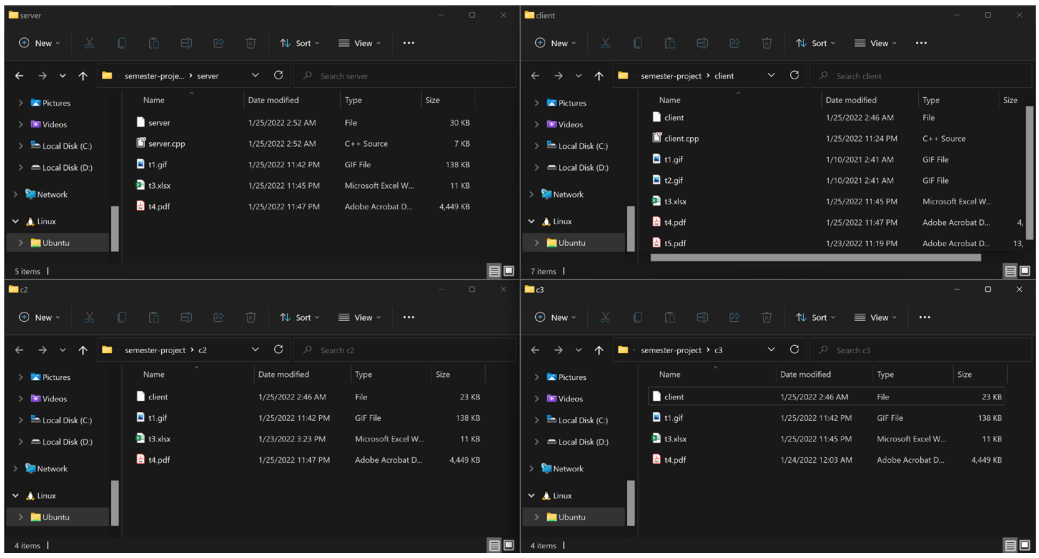
Client 2 sends a file (t3.xlsx).



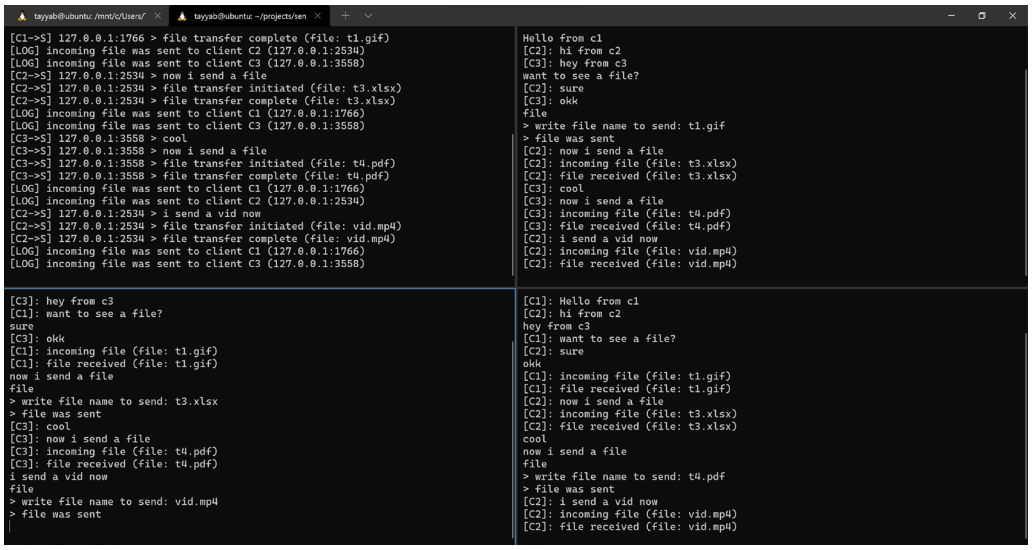
Again the file sent by C2 is received by everyone connected.



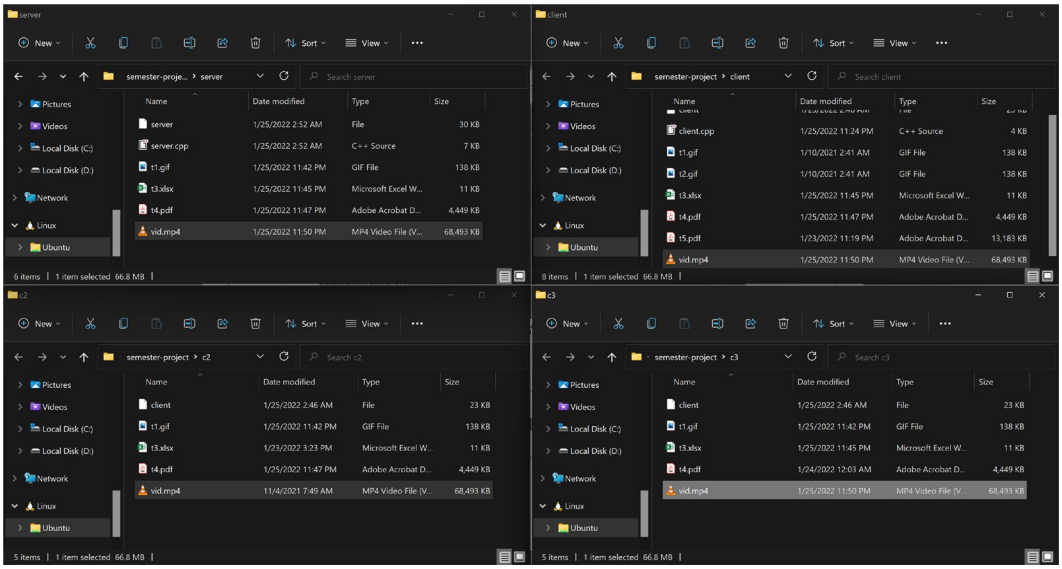
Client 3 sends a file (t4.pdf).



File is again received by everyone connected.



Client C2 sends a video file (vid.mp4 of 66.8 MB)



Everyone connected receives the video file of 66.8 MB.

And that is the working of the multi-party chat application with file sending.

CODE

SERVER.CPP:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <netinet/in.h>
#include <pthread.h>
#include <iostream>

#define PORT        8080
#define BUF_SIZ     1024
#define NAME_SIZ    10
#define MAX_CLIENTS 3

using namespace std;

// data related to the client
struct client {
    int                 index;
    int                 sockfd;
    struct sockaddr_in  addr;
    pthread_t           thread;
};

// arguments to be passed to the thread
struct arg_struct {
    struct client c;
};

int client_count = 0;
client clients[MAX_CLIENTS];

void send_name(int sockfd, string name) {
    char temp[NAME_SIZ];
    strcpy(temp, name.c_str());
    send(sockfd, (char*) &temp, strlen(temp), 0);
}

void* handle_send(void*) {
```

```
    char buf[BUF_SIZ];

    while(true) {
        memset(&buf, 0, sizeof(buf));

        string data;
        getline(cin, data);
        strcpy(buf, data.c_str());

        for(int i = 0; i < client_count; i++) {
            send_name(clients[i].sockfd, "S");
            usleep(100);
            send(clients[i].sockfd, (char*) &buf, BUF_SIZ, 0);
            usleep(100);
        }
    }

    return 0;
}

void* handle_recv (void* arguments) {
    struct arg_struct *args = (struct arg_struct *) arguments;

    int client_sockfd = args->c.sockfd;
    struct sockaddr_in clientaddr = args->c.addr;
    string cli_index = "C" + to_string(args->c.index);
    char buf[BUF_SIZ];
    int read_len, file_read_len;
    int filefd;

    while(true) {
        // temp buffer
        char data[BUF_SIZ];
        char file_name[BUF_SIZ];

        memset(buf, 0, BUF_SIZ);

        read_len = read(client_sockfd, buf, BUF_SIZ);
        if(read_len < 0) {
            perror("error occured during reading");
            close(client_sockfd);
            break;
        }

        strcpy(data, buf);

        // close client if end of file or if client sends close
        if (strcmp(data, "close") == 0 || strlen(data) == 0){
```

```
            close(client_sockfd);
            printf("[LOG] %s:%d > client %s left\n", inet_ntoa(clientaddr.sin_addr),
clientaddr.sin_port, cli_index.c_str());
            break;
        }

        // client is sending a file
        if (strcmp(data, "file") == 0) {
            read_len = read(client_sockfd, buf, BUF_SIZ);
            strcpy(file_name, buf);
            strcpy(data, buf);

            // create file
            filefd = open(file_name, O_WRONLY | O_CREAT | O_EXCL, 0700);
            if(!filefd) {
                perror("file open error: ");
                continue;
            }

            printf("[%s->S] %s:%d > file transfer initiated (file: %s)\n",
cli_index.c_str(), inet_ntoa(clientaddr.sin_addr), clientaddr.sin_port, file_name);

            // file save
            while(true) {
                memset(&buf, 0, BUF_SIZ);

                file_read_len = read(client_sockfd, buf, BUF_SIZ);
                write(filefd, buf, file_read_len);

                // when less than 1024, means last packet of the file
                if(file_read_len < 1024) {
                    printf("[%s->S] %s:%d > file transfer complete (file: %s)\n",
cli_index.c_str(), inet_ntoa(clientaddr.sin_addr), clientaddr.sin_port, file_name);
                    break;
                }
            }

            close(filefd);
            memset(&buf, 0, BUF_SIZ);
            memset(&data, 0, BUF_SIZ);

            // server will now be sending file to other clients

            string dat;

            for(int i = 0; i < client_count; i++) {
                // dont send to the same client
                if (clients[i].sockfd == client_sockfd) continue;
```

```
                int cli_sockfd = clients[i].sockfd;
                struct sockaddr_in cli_addr = clients[i].addr;

                send_name(cli_sockfd, cli_index);
                usleep(1000);

                dat = "file";
                strcpy(buf, dat.c_str());

                // tell the client that a file is being sent
                send(cli_sockfd, &buf, strlen(buf), 0);

                usleep(1000);

                memset(&buf, 0, BUF_SIZ);
                memset(&data, 0, BUF_SIZ);

                // tell server the file name
                send(cli_sockfd, &file_name, strlen(file_name), 0);

                usleep(1000);

                // load the file as readonly
                filefd = open(file_name, O_RDONLY);
                if(!filefd) {
                    perror("error");
                    continue;
                }

                // send the entire file
                while(true) {
                    memset(&buf, 0, BUF_SIZ);

                    // read file into the buffer
                    read_len = read(filefd, buf, BUF_SIZ);
                    // send file to the server
                    send(cli_sockfd, &buf, read_len, 0);

                    // reached the end of file
                    if(read_len == 0) {
                        printf("[LOG] incoming file was sent to client C%d
(%s:%d)\n",  clients[i].index, inet_ntoa(cli_addr.sin_addr), cli_addr.sin_port);
                        break;
                    }
                }
            }
```

```
            continue;
        }

        // client sent a normal message

        printf("[%s->S] %s:%d > %s\n", cli_index.c_str(), inet_ntoa(clientaddr.sin_addr),
clientaddr.sin_port, data);

        for(int i = 0; i < client_count; i++) {
            // dont send to the same client again
            if (clients[i].sockfd == client_sockfd) continue;

            send_name(clients[i].sockfd, cli_index);
            usleep(1000);
            send(clients[i].sockfd, (char*) &buf, strlen(buf), 0);
        }
    }

    return 0;
}

int main() {
    int server_sockfd, client_sockfd;
    int client_len;
    struct sockaddr_in serveraddr, clientaddr;
    char buf[BUF_SIZ];

    client_len = sizeof(clientaddr);

    // create socket
    if((server_sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket error : ");
        exit(0);
    }

    // populate serveraddr struct
    bzero(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family       = AF_INET;
    serveraddr.sin_addr.s_addr  = htonl(INADDR_ANY);
    serveraddr.sin_port         = htons(PORT);

    // bind socket
    if(bind(server_sockfd, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) > 0) {
        perror("bind error : ");
        exit(0);
    }

    // listen for clients
```

```
    if(listen(server_sockfd, MAX_CLIENTS) != 0) {
        perror("listen error : ");
    }

    pthread_t send_th;
    pthread_create(&send_th, NULL, handle_send, NULL);

    while(true) {
        // limit the maximum number of clients
        if (client_count >= MAX_CLIENTS) continue;

        // accept a new client socket
        client_sockfd = accept(server_sockfd, (struct sockaddr *) &clientaddr, (socklen_t*)
&client_len);
        if (client_sockfd < 0) {
            perror("socket accept failure");
            exit(1);
        }

        printf("Client %d with IP: %s and PORT: %d has connected\n", client_count + 1,
inet_ntoa(clientaddr.sin_addr), clientaddr.sin_port);

        // populate the client thread
        struct client c;
        c.index = client_count + 1;
        c.sockfd = client_sockfd;
        c.addr = clientaddr;

        // add the client to the global array
        clients[client_count] = c;
        client_count++;

        // args to be passed to the thread
        struct arg_struct args;
        args.c = c;

        // handle data receival from client in a separate thread
        pthread_t recv_th;
        pthread_create(&recv_th, NULL, handle_recv, (void*) &args);

        c.thread = recv_th;
    }

    close(server_sockfd);
    return 0;
}
```

CLIENT.CPP:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <iostream>

#define PORT    8080
#define IP      "127.0.0.1"
#define BUF_SIZ 1024

using namespace std;

void* handle_recv(void* conn) {
    long sockfd = (long) conn;
    int filefd;
    char buf[BUF_SIZ];
    char data[BUF_SIZ];
    char cli_name[BUF_SIZ];
    int n, read_len, file_read_len;

    while(true) {
        memset(&buf, 0, sizeof(buf));
        memset(&cli_name, 0, sizeof(cli_name));

        // first receive the name of the client or server
        n = recv(sockfd, (char*) &cli_name, sizeof(cli_name), 0);

        // verify that name or data was received
        if(n == 0 || strlen(cli_name) == 0) {
            cout << "server went offline" << endl;
            exit(1);
        }

        // now receive the actual data
        n = recv(sockfd, (char*) &buf, sizeof(buf), 0);

        // verify that the data was received
        if(n == 0 || strlen(buf) == 0) {
```

```
            cout << "server went offline" << endl;
            exit(1);
        }

        // recv file from the server
        if (strcmp(buf, "file") == 0) {
            read_len = read(sockfd, buf, BUF_SIZ);
            strcpy(data, buf);

            // create file
            filefd = open(data, O_WRONLY | O_CREAT | O_EXCL, 0700);
            if(!filefd) {
                perror("file open error: ");
                continue;
            }

            printf("[%s]: incoming file (file: %s)\n", cli_name, data);

            // file save
            while(true) {
                memset(&buf, 0, BUF_SIZ);

                file_read_len = read(sockfd, buf, BUF_SIZ);
                write(filefd, buf, file_read_len);

                // when less than 1024, means last packet of the file
                if(file_read_len < 1024) {
                    printf("[%s]: file received (file: %s)\n", cli_name, data);
                    break;
                }
            }

            close(filefd);

            continue;
        }

        // normal message
        printf("[%s]: %s\n", cli_name, buf);
    }
}

int main() {
    intptr_t            sockfd;
    int                 filefd;
    char                buf[BUF_SIZ];
    int                 file_name_len, read_len;
    struct sockaddr_in  serv_addr;
```

```
// create socket
if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket creation failed");
    close(sockfd);
    exit(1);
}

// populate address struct
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(IP);
serv_addr.sin_port = htons(PORT);

// connect to the server socket
if((connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))) == -1) {
    perror("failed to connect");
    close(sockfd);
    exit(1);
}

pthread_t th;
pthread_create(&th, NULL, handle_recv, (void*) sockfd);

while(true) {
    string data;
    getline(cin, data);

    if (data == "close") {
        printf("exiting\n");
        break;
    }

    memset(&buf, 0, BUF_SIZ);
    strcpy(buf, data.c_str());

    // file transfer
    if (data == "file") {
        // tell the server that a file is being sent
        send(sockfd, &buf, strlen(buf), 0);

        memset(&buf, 0, BUF_SIZ);

        // get the file name from the client to send
        printf("> write file name to send: ");
        scanf("%s", buf);

        // tell server the file name
        send(sockfd, &buf, strlen(buf), 0);
```

```
            // load the file as readonly
            filefd = open(buf, O_RDONLY);
            if(!filefd) {
                perror("error");
                continue;
            }

            // send the entire file
            while(true) {
                memset(&buf, 0, BUF_SIZ);

                // read file into the buffer
                read_len = read(filefd, buf, BUF_SIZ);
                // send file to the server
                send(sockfd, &buf, read_len, 0);

                // reached the end of file
                if(read_len == 0) {
                    printf("> file was sent\n");
                    break;
                }
            }

            continue;
        }

        // normal chat
        send(sockfd, &buf, strlen(buf), 0);
    }

    close(sockfd);

    return 0;
}
```