اونيۏرسيتي ملیسیا ڤهڠ

UNIVERSITI MALAYSIA PAHANG

BCN3033

Network Programming

PROJECT 2

Title of Project:

Chat Application In Different Operating Systems (OS)

Name	Muhammad Nur Aiman Bin Ali
Student ID	CA21062

Lecturer :

Dr Ahmad Firdaus bin Zainal Abidin

Course Outcome	**Marks (weight score | percentage)**
CO1	/40
CO2	/5
CO3	/5
	/40

Table of Contents

A chat application between different operating system (OS)

- **Chat application in Ubuntu (Server)**

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

#define PORT 8888
#define BUFFER_SIZE 1024

void cleanup(int sockfd)
{
    close(sockfd);
}

int receive_file(int sockfd, char *filename) {
    FILE *file = fopen(filename, "wb");
    if (file == NULL) {
        perror("File opening error");
        return 1;
    }

    char buffer[BUFFER_SIZE];
    ssize_t num_bytes;
    while ((num_bytes = recv(sockfd, buffer, BUFFER_SIZE, 0)) > 0) {
        fwrite(buffer, 1, num_bytes, file);
    }

    fclose(file);
    return 0;
}

int send_file(int sockfd, char *filename) {
    FILE *file = fopen(filename, "rb");
    if (file == NULL) {
        perror("File opening error");
        return 1;
    }

    char buffer[BUFFER_SIZE];
    ssize_t num_bytes;
    while ((num_bytes = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
        send(sockfd, buffer, num_bytes, 0);
    }
```

```
    fclose(file);
    return 0;
}

int main()
{
    int sockfd, newsockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;
    char buffer[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation error");
        return 1;
    }

    printf("\n|-----------------------------------------------------
----------|\n");
    printf("|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE
|\n");
    printf("|-----------------------------------------------------
--------|\n\n");

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Binding error");
        cleanup(sockfd);
        return 1;
    }

    if (listen(sockfd, 5) < 0) {
        perror("Listening error");
        cleanup(sockfd);
        return 1;
    }
    printf("Server listening on port %d\n", PORT);

    client_len = sizeof(client_addr);
    newsockfd = accept(sockfd, (struct sockaddr *)&client_addr,
&client_len);
    if (newsockfd < 0) {
        perror("Accepting connection error");
        cleanup(sockfd);
        return 1;
```

```
    }
    printf("Client connected\n");

    while (1) {
        memset(buffer, 0, sizeof(buffer));
        ssize_t num_bytes = recv(newsockfd, buffer, BUFFER_SIZE, 0);
        if (num_bytes <= 0) {
            printf("Client disconnected\n");
            break;
        }
        printf("Client: %s", buffer);

        if (strncmp(buffer, "exit", 4) == 0) {
            printf("Client requested to close the connection\n");
            break;
        } else if (strncmp(buffer, "sendfile", 8) == 0) {
            char filename[256];
            sscanf(buffer + 9, "%s", filename);
            printf("File transfer requested: %s\n", filename);

            if (receive_file(newsockfd, filename) == 0) {
                printf("File received successfully: %s\n",
filename);
            } else {
                printf("Error receiving file: %s\n", filename);
            }
        } else if (strncmp(buffer, "getfile", 7) == 0) {
            char filename[256];
            sscanf(buffer + 8, "%s", filename);
            printf("File transfer requested: %s\n", filename);

            if (send_file(newsockfd, filename) == 0) {
                printf("File sent successfully: %s\n", filename);
            } else {
                printf("Error sending file: %s\n", filename);
            }
        } else {
            memset(buffer, 0, sizeof(buffer));
            printf("Server: ");
            fgets(buffer, BUFFER_SIZE, stdin);

            send(newsockfd, buffer, strlen(buffer), 0);
        }
    }

    cleanup(newsockfd);
    cleanup(sockfd);
    return 0;
}
```

- **Chat application in Ubuntu (Client)**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>

#define PORT 8888
#define BUFFER_SIZE 1024

void send_file(int sockfd, const char *filename) {


}

int main()
{
    int sockfd;
    struct sockaddr_in server_addr;
    struct hostent *server;
    char buffer[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation error");
        return 1;
    }

    printf("\n|-----------------------------------------------------
----------|\n");
    printf("|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE
|\n");
    printf("|-----------------------------------------------------
--------|\n\n");

printf("\n**********************************************************
*******");
    printf("\n* Welcome to the virtual space for conversation and
connection. *");
    printf("\n*  Here you can talk about anything and everything
with others  *");
    printf("\n*    who are interested in the same topics. Enjoy your
stay!    *");
```

```
printf("\n**********************************************************
*******\n");

    server = gethostbyname("192.168.56.1");
    if (server == NULL) {
        fprintf(stderr, "Failed to resolve server host\n");
        close(sockfd);
        return 1;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    bcopy((char *)server->h_addr, (char
*)&server_addr.sin_addr.s_addr, server->h_length);

    if (connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Connection error");
        close(sockfd);
        return 1;
    }
    printf("Connected to server\n");

    while (1) {
        memset(buffer, 0, sizeof(buffer));
        printf("Client: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        buffer[strcspn(buffer, "\n")] = '\0';

        if (strncmp(buffer, "sendfile", 8) == 0) {

            char *filename = buffer + 9;
            send_file(sockfd, filename);
            continue;
        }

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

        if (strncmp(buffer, "exit", 4) == 0) {
            printf("Disconnected from server\n");
            break;
        }

        memset(buffer, 0, sizeof(buffer));
        recv(sockfd, buffer, BUFFER_SIZE, 0);
        printf("Server: %s\n", buffer);
    }

    close(sockfd);
```

```
    return 0;
}
```

- **Chat application in Window (Server)**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8888
#define BUFFER_SIZE 1024

void cleanup(SOCKET sockfd)
{
    closesocket(sockfd);
    WSACleanup();
}

void send_file(SOCKET sockfd, FILE *file)
{
    char data[BUFFER_SIZE] = {0};
    int bytes_read;

    while ((bytes_read = fread(data, 1, BUFFER_SIZE, file)) > 0)
    {
        if (send(sockfd, data, bytes_read, 0) < 0)
        {
            fprintf(stderr, "Error sending file.\n");
            break;
        }
    }

    // Signal end of file
    send(sockfd, NULL, 0, 0);
}

void receive_file(SOCKET sockfd, FILE *file)
{
    char data[BUFFER_SIZE] = {0};
    int bytes_received;

    while (1)
    {
        bytes_received = recv(sockfd, data, BUFFER_SIZE, 0);
        if (bytes_received < 0)
        {
            fprintf(stderr, "Error receiving file.\n");
            break;
        }
        else if (bytes_received == 0)
        {
```

```
            printf("File transfer complete.\n");
            break;
        }

        fwrite(data, 1, bytes_received, file);
    }
}

int main()
{
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        fprintf(stderr, "Failed to initialize Winsock.\n");
        return 1;
    }

    SOCKET sockfd, newsockfd;
    struct sockaddr_in server_addr, client_addr;
    int client_len;
    char buffer[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == INVALID_SOCKET) {
        perror("Socket creation error");
        cleanup(sockfd);
        return 1;
    }

    printf("\n|-----------------------------------------------------
----------|\n");
    printf("|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE
|\n");
    printf("|-----------------------------------------------------
--------|\n\n");

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) == SOCKET_ERROR) {
        perror("Binding error");
        cleanup(sockfd);
        return 1;
    }

    if (listen(sockfd, 5) == SOCKET_ERROR) {
        perror("Listening error");
        cleanup(sockfd);
        return 1;
```

```
    }
    printf("Server listening on port %d\n", PORT);

    client_len = sizeof(client_addr);
    newsockfd = accept(sockfd, (struct sockaddr*)&client_addr,
&client_len);
    if (newsockfd == INVALID_SOCKET) {
        perror("Accepting connection error");
        cleanup(sockfd);
        return 1;
    }
    printf("Client connected\n");

    while (1) {
        memset(buffer, 0, sizeof(buffer));
        if (recv(newsockfd, buffer, BUFFER_SIZE, 0) <= 0) {
            printf("Client disconnected\n");
            break;
        }
        printf("Client: %s", buffer);

        if (strncmp(buffer, "exit", 4) == 0) {
            printf("Client requested to close the connection\n");
            break;
        }

        if (strncmp(buffer, "sendfile", 8) == 0) {
            char filename[BUFFER_SIZE];
            memset(filename, 0, sizeof(filename));
            recv(newsockfd, filename, BUFFER_SIZE, 0);

            FILE *file = fopen(filename, "rb");
            if (file == NULL) {
                printf("File %s not found.\n", filename);
                send(newsockfd, "File not found", strlen("File not
found"), 0);
            } else {
                send(newsockfd, "File found", strlen("File found"),
0);
                send_file(newsockfd, file);
                fclose(file);
            }
        } else if (strncmp(buffer, "receivefile", 11) == 0) {
            char filename[BUFFER_SIZE];
            memset(filename, 0, sizeof(filename));
            recv(newsockfd, filename, BUFFER_SIZE, 0);

            FILE *file = fopen(filename, "wb");
            if (file == NULL) {
```

```
                fprintf(stderr, "Error creating file %s\n",
filename);
                send(newsockfd, "Error creating file", strlen("Error
creating file"), 0);
            } else {
                send(newsockfd, "Ready to receive", strlen("Ready to
receive"), 0);
                receive_file(newsockfd, file);
                fclose(file);
            }
        } else {
            memset(buffer, 0, sizeof(buffer));
            printf("\nServer: ");
            fgets(buffer, BUFFER_SIZE, stdin);

            send(newsockfd, buffer, strlen(buffer), 0);
        }
    }

    cleanup(newsockfd);
    cleanup(sockfd);
    return 0;
}
```

- **Chat application in Window (Client)**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>

#define PORT 8888
#define BUFFER_SIZE 1024

void cleanup(SOCKET sockfd)
{
    closesocket(sockfd);
    WSACleanup();
}

int main()
{
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        fprintf(stderr, "Failed to initialize Winsock.\n");
        return 1;
    }

    SOCKET sockfd;
    struct sockaddr_in server_addr;
    struct hostent* server;
    char buffer[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == INVALID_SOCKET) {
        perror("Socket creation error");
        cleanup(sockfd);
        return 1;
    }

    printf("\n|-----------------------------------------------------
----------|\n");
    printf("|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE
|\n");
    printf("|-----------------------------------------------------
--------|\n\n");

printf("\n**********************************************************
*******");
    printf("\n* Welcome to the virtual space for conversation and
connection. *");
    printf("\n*  Here you can talk about anything and everything
with others  *");
```

```
    printf("\n*    who are interested in the same topics. Enjoy your
stay!    *");

printf("\n**********************************************************
*******\n");

    server = gethostbyname("192.168.56.1");
    if (server == NULL) {
        fprintf(stderr, "Failed to resolve server host\n");
        cleanup(sockfd);
        return 1;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    memcpy((char*)&server_addr.sin_addr.s_addr, (char*)server-
>h_addr, server->h_length);

    if (connect(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) == SOCKET_ERROR) {
        fprintf(stderr, "Connection error: %d\n",
WSAGetLastError());
        cleanup(sockfd);
        return 1;
    }
    printf("Connected to server\n");

    while (1) {
        memset(buffer, 0, sizeof(buffer));
        printf("Client: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        buffer[strcspn(buffer, "\n")] = '\0';

        if (strncmp(buffer, "sendfile", 8) == 0) {

            send(sockfd, "file", strlen("file"), 0);
        } else {

            send(sockfd, buffer, strlen(buffer), 0);
        }

        if (strncmp(buffer, "exit", 4) == 0) {
            printf("Disconnected from server\n");
            break;
        }
        memset(buffer, 0, sizeof(buffer));
        int bytesReceived = recv(sockfd, buffer, BUFFER_SIZE, 0);
        if (bytesReceived <= 0) {
            if (bytesReceived == 0) {
                printf("Disconnected from server\n");
```

```
            } else {
                fprintf(stderr, "Receive error: %d\n",
WSAGetLastError());
            }
            break;
        }

        if (strncmp(buffer, "file", 4) == 0) {

        } else {
            printf("Server: %s\n", buffer);
        }
    }
    cleanup(sockfd);
    return 0;
}
```

Explanation of all lines/paragraph of codes.

- **Ubuntu (Server)**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8888
#define BUFFER_SIZE 1024
```

Here, we define the port number and buffer size constants as well as the relevant header files.

```
void cleanup(SOCKET sockfd)
{
    closesocket(sockfd);
    WSACleanup();
}
```

The cleaning function uses the close function to close a socket after receiving a socket file descriptor (sockfd) as an argument.

```
int receive_file(int sockfd, char *filename) {
    FILE *file = fopen(filename, "wb");
    if (file == NULL) {
        perror("File opening error");
        return 1;
    }

    char buffer[BUFFER_SIZE];
    ssize_t num_bytes;
    while ((num_bytes = recv(sockfd, buffer, BUFFER_SIZE, 0)) > 0) {
        fwrite(buffer, 1, num_bytes, file);
    }

    fclose(file);
    return 0;
}
```

This function is responsible for receiving a file from the client and saving it locally on the server side.

Parameters:

- `sockfd`: Socket file descriptor for communication.

- `filename`: Name of the file to be received and saved.

Process:

- Opens the specified file (`filename`) in write binary mode (`wb`).

- Receives data in chunks using `recv` and writes the received data into the file using `fwrite`.

- Continues receiving data until the end of the file transmission (`recv` returns 0) or encounters an error.

- Finally, closes the file and returns 0 if the file was received successfully, or 1 in case of an error.

```c
int send_file(int sockfd, char *filename) {
    FILE *file = fopen(filename, "rb");
    if (file == NULL) {
        perror("File opening error");
        return 1;
    }

    char buffer[BUFFER_SIZE];
    ssize_t num_bytes;
    while ((num_bytes = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
        send(sockfd, buffer, num_bytes, 0);
    }

    fclose(file);
    return 0;
}
```

This function sends a file from the server to the client.

- Parameters:

- `sockfd`: Socket file descriptor for communication.

- `filename`: Name of the file to be sent to the client.

- Process:

- Opens the specified file (`filename`) in read binary mode (`rb`).

- Reads data from the file in chunks using `fread` and sends the data through the socket to the client using `send`.

- Continues sending data until the end of the file is reached or an error occurs.

- Finally, closes the file and returns 0 if the file was sent successfully, or 1 in case of an error.

```c
int main()
{
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        fprintf(stderr, "Failed to initialize Winsock.\n");
        return 1;
    }

    SOCKET sockfd, newsockfd;
    struct sockaddr_in server_addr, client_addr;
    int client_len;
    char buffer[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == INVALID_SOCKET) {
        perror("Socket creation error");
        cleanup(sockfd);
        return 1;
    }
```

Starting now is the key task. We set variables for the client and server socket file descriptors (sockfd and newsockfd), the client and server socket address structures (server_addr and client_addr), the client address structure length (client_len), and the buffer array (buffer).

When making a socket, the socket function is used. The first option, AF_INET, designates an IPv4 address family; the second, SOCK_STREAM, a TCP socket; and the third, 0, the

automatic selection of the protocol. A non-zero status is returned by the programme if the socket construction fails, and perror is used to output an error message.

```
36      printf("\n|-----------------------------------------------------------------|\n");
37      printf("|           GREETINGS! WELCOME TO VIRTUAL CHAT SPACE           |\n");
38      printf("|-----------------------------------------------------------------|\n\n");
```

These lines just print a greeting to the console.

```
40      server_addr.sin_family = AF_INET;
41      server_addr.sin_port = htons(PORT);
42      server_addr.sin_addr.s_addr = INADDR_ANY;
43
```

Here, we configure the server address configuration. The constants AF_INET designates IPv4 as the address family, htons(PORT) changes the port number defined by PORT to the proper network byte order, and INADDR_ANY instructs the server to bind to any and all available network interfaces.

```
43
44      if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
45          perror("Binding error");
46          cleanup(sockfd);
47          return 1;
48      }
```

The socket (sockfd) is bound to the server address structure by calling the bind function. If the binding fails, cleanup(sockfd) is called to terminate the connection, an error message is written using perror, and the programme exits with a non-zero status.

```
50      if (listen(sockfd, 5) == SOCKET_ERROR) {
51          perror("Listening error");
52          cleanup(sockfd);
53          return 1;
54      }
55      printf("Server listening on port %d\n", PORT);
```

The server starts to listen for incoming connections on the socket when the listen function is called. The queue's maximum allowed number of pending connections is indicated by the second argument, 5, which is given. If the program's attempt to listen is unsuccessful, an error message is printed, the socket is closed, and a non-zero status is returned. The server is now listening on the chosen port, according to a message that is printed if the operation is successful.

```
    printf("Server listening on port %d\n", PORT);

    client_len = sizeof(client_addr);
    newsockfd = accept(sockfd, (struct sockaddr*)&client_addr, &client_len);
    if (newsockfd == INVALID_SOCKET) {
        perror("Accepting connection error");
        cleanup(sockfd);
        return 1;
    }
    printf("Client connected\n");
```

The accept function is invoked in order to accept an incoming connection request after initialising the client address structure's size. Once a client connects, it becomes blocked. If the connection request is unsuccessful, the socket is closed, an error message is shown, and the programme returns with a non-zero status. If successful, a message confirming a client connection is printed.

```
    while (1) {
        memset(buffer, 0, sizeof(buffer));
        ssize_t num_bytes = recv(newsockfd, buffer, BUFFER_SIZE, 0);
        if (num_bytes <= 0) {
            printf("Client disconnected\n");
            break;
        }
        printf("Client: %s", buffer);

        if (strncmp(buffer, "exit", 4) == 0) {
            printf("Client requested to close the connection\n");
            break;
        } else if (strncmp(buffer, "sendfile", 8) == 0) {
            char filename[256];
            sscanf(buffer + 9, "%s", filename);
            printf("File transfer requested: %s\n", filename);

            if (receive_file(newsockfd, filename) == 0) {
                printf("File received successfully: %s\n", filename);
            } else {
                printf("Error receiving file: %s\n", filename);
            }
        } else if (strncmp(buffer, "getfile", 7) == 0) {
            char filename[256];
            sscanf(buffer + 8, "%s", filename);
            printf("File transfer requested: %s\n", filename);

            if (send_file(newsockfd, filename) == 0) {
                printf("File sent successfully: %s\n", filename);
            } else {
                printf("Error sending file: %s\n", filename);
            }
        } else {
            memset(buffer, 0, sizeof(buffer));
            printf("Server: ");
            fgets(buffer, BUFFER_SIZE, stdin);

            send(newsockfd, buffer, strlen(buffer), 0);
        }
    }
```

This is the program's main loop. Once the link is established, it never ends.The buffer is cleaned up using memset inside the loop. Then, data is obtained from the client and put in the buffer by calling the recv function. The loop is broken, and the client has disconnected if the amount of bytes received is less than or equal to 0.

The client's message that was received is shown on the server's console. The server publishes a message and breaks the loop to cut off the connection if the message it receives is "exit".Memset is used to clear the buffer once more. The server prints "Server:" to request input. To read a line of data from the console and save it in the buffer, the fgets function is used.Finally, the server uses the send function to communicate the buffer's contents to the client.

If the message is "sendfile", the server extracts the filename and initiates the file receiving process using `receive_file`. If the message is "getfile", the server extracts the filename and

initiates the file sending process using `send_file`. For any other messages (neither file transfer commands nor exit commands), the server continues handling them as regular messages.

```
85
86          cleanup(newsockfd);
87          cleanup(sockfd);
88          return 0;
89      }
90
```

The cleanup method is run after exiting the loop in order to close the client socket (newsockfd) and the server socket (sockfd). In order to signify successful programme running, the main function returns 0.

- **Ubuntu (Client)**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8888
#define BUFFER_SIZE 1024
```

Here, we define the port number and buffer size constants as well as the relevant header files.

```c
void send_file(int sockfd, const char *filename) {

}
```

`send_file()` is a function intended to handle the logic of sending a file over a socket. This function takes the socket file descriptor (`sockfd`) and the name of the file (`filename`) to be sent as parameters.

```c
int main()
{
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        fprintf(stderr, "Failed to initialize Winsock.\n");
        return 1;
    }

    SOCKET sockfd;
    struct sockaddr_in server_addr;
    struct hostent* server;
    char buffer[BUFFER_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == INVALID_SOCKET) {
        perror("Socket creation error");
        cleanup(sockfd);
        return 1;
    }
```

The primary task starts. We declare variables for the buffer array (buffer), the server address structure (server_addr), a pointer to struct hostent for server information (server), the socket file descriptor (sockfd), and the server.

To create a socket, the socket function is used. The first option, AF_INET, designates IPv4 as the address family, the second, SOCK_STREAM, designates a TCP socket, and the third, 0, says that the protocol should be determined automatically. If the socket construction doesn't succeed, perror is used to produce an error message, and the programme exits with a non-zero status.

```c
    printf("\n|-------------------------------------------------------------|\n");
    printf("|           GREETINGS! WELCOME TO VIRTUAL CHAT SPACE          |\n");
    printf("|-------------------------------------------------------------|\n\n");
    printf("\n*************************************************************");
    printf("\n* Welcome to the virtual space for conversation and connection. *");
    printf("\n*  Here you can talk about anything and everything with others  *");
    printf("\n*    who are interested in the same topics. Enjoy your stay!    *");
    printf("\n*************************************************************\n");
```

These lines print a greeting to the console that includes details about the virtual chat room.

```
    server = gethostbyname("192.168.56.1");
    if (server == NULL) {
        fprintf(stderr, "Failed to resolve server host\n");
        cleanup(sockfd);
        return 1;
    }
```

In order to obtain host information based on the specified server name or IP address, the gethostbyname method is used. This instance makes use of the IP address "192.168.56.1". An error message is produced, the socket is closed, and the programme returns with a non-zero status if the function is unable to resolve the server host.

```
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    memcpy((char*)&server_addr.sin_addr.s_addr, (char*)server->h_addr, server->h_length);
```

Here, we configure the server address configuration. The port number provided by the constant PORT is converted to the proper network byte order by htons(PORT), where AF_INET indicates that the address family is IPv4. The server's IP address is copied from the server->h_addr field to the server_addr.sin_addr.s_addr field using the bcopy function.

```
    if (connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
        fprintf(stderr, "Connection error: %d\n", WSAGetLastError());
        cleanup(sockfd);
        return 1;
    }
    printf("Connected to server\n");
```

The connect function is used to connect to the server using the socket sockfd and the server address structure server_addr. If the connection fails, the socket is closed, an error message is written using perror, and the programme exits with a non-zero status. If successful, a message is printed stating that the client and server are linked.

```
    while (1) {
        memset(buffer, 0, sizeof(buffer));
        printf("Client: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        buffer[strcspn(buffer, "\n")] = '\0';

        if (strncmp(buffer, "sendfile", 8) == 0) {

            char *filename = buffer + 9;
            send_file(sockfd, filename);
            continue;
        }

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

        if (strncmp(buffer, "exit", 4) == 0) {
            printf("Disconnected from server\n");
            break;
        }

        memset(buffer, 0, sizeof(buffer));
        recv(sockfd, buffer, BUFFER_SIZE, 0);
        printf("Server: %s\n", buffer);
    }
```

The `main()` function includes a check within the while loop to handle a specific command, "sendfile". If the user enters a command starting with "sendfile", the program extracts the filename from the command string and calls the `send_file()` function, passing the socket file descriptor (`sockfd`) and the extracted filename. After sending the file or regular messages, the code continues to receive messages from the server as before. This allows the program to differentiate between regular messages and the "sendfile" command. When the user inputs "sendfile filename", it triggers the `send_file()` function, which you'll need to implement to handle the file transfer logic over the socket.

```
88          cleanup(sockfd);
89          return 0;
90      }
```

After breaking out of the loop, the client closes the socket by calling close(sockfd).The main function returns 0 to indicate successful program execution.

- **Window (Server)**

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

#define PORT 8888
#define BUFFER_SIZE 1024

void cleanup(int sockfd)
{
    close(sockfd);
}
```

The definition of the constants for the port number and buffer size are included in the code's first line. Additionally, a cleanup function is provided that shuts the socket with the help of a socket file descriptor.

```c
void send_file(SOCKET sockfd, FILE *file)
{
    char data[BUFFER_SIZE] = {0};
    int bytes_read;

    while ((bytes_read = fread(data, 1, BUFFER_SIZE, file)) > 0)
    {
        if (send(sockfd, data, bytes_read, 0) < 0)
        {
            fprintf(stderr, "Error sending file.\n");
            break;
        }
    }


    send(sockfd, NULL, 0, 0);
}
```

This function is responsible for sending a file through the socket connection. It reads the file data in chunks of `BUFFER_SIZE`, sends each chunk over the socket, and repeats until the entire file is sent. It uses `fread` to read data from the file, and `send` to transmit the data over the socket. It sends a signal to indicate the end of the file transfer.

```
34  void receive_file(SOCKET sockfd, FILE *file)
35  {
36      char data[BUFFER_SIZE] = {0};
37      int bytes_received;
38
39      while (1)
40      {
41          bytes_received = recv(sockfd, data, BUFFER_SIZE, 0);
42          if (bytes_received < 0)
43          {
44              fprintf(stderr, "Error receiving file.\n");
45              break;
46          }
47          else if (bytes_received == 0)
48          {
49              printf("File transfer complete.\n");
50              break;
51          }
52
53          fwrite(data, 1, bytes_received, file);
54      }
55  }
```

This function is responsible for receiving a file through the socket connection. It continuously receives data chunks from the socket using `recv` and writes them to the file using `fwrite`. The function stops when it receives an empty packet indicating the end of the file transfer.

```
17  int main()
18  {
19      int sockfd, newsockfd;
20      struct sockaddr_in server_addr, client_addr;
21      socklen_t client_len;
22      char buffer[BUFFER_SIZE];
23
24      sockfd = socket(AF_INET, SOCK_STREAM, 0);
25      if (sockfd < 0) {
26          int main::sockfd eation error");
27          return 1;
28      }
```

The socket file descriptors (sockfd and newsockfd), the server address structure (server_addr), the client address structure (client_addr), the length of the client address (client_len), and a buffer for message storage (buffer) are all declared at the beginning of the main function.

After that, a socket is created by calling the socket function. An IPv4 address family is specified by the AF_INET parameter, a TCP socket is specified by SOCK_STREAM, and 0 indicates that the protocol should be determined automatically. If the socket creation fails, the programme returns with a non-zero status and prints an error message using the perror command.

```
30      printf("\n|-----------------------------------------------------------------|\n");
31      printf("|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE            |\n");
32      printf("|-----------------------------------------------------------------|\n\n");
```

These lines print a welcome message on the console.

```
34      server_addr.sin_family = AF_INET;
35      server_addr.sin_port = htons(PORT);
36      server_addr.sin_addr.s_addr = INADDR_ANY;
37
38      if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
39          perror("Binding error");
40          cleanup(sockfd);
41          return 1;
42      }
```

The IP address is set to INADDR_ANY in the server address structure (server_addr), allowing the server to bind to any accessible network interface. The address family is set to IPv4 and the port number is translated to network byte order using htons.

The socket (sockfd) is then linked to the server address structure by using the bind function. If the binding fails, the programme returns with a non-zero status, an error message is printed, and the socket is closed.

```
44      if (listen(sockfd, 5) < 0) {
45          perror("Listening error");
46          cleanup(sockfd);
47          return 1;
48      }
49      printf("Server listening on port %d\n", PORT);
```

The socket is placed in a passive listening state by calling the listen function. The maximum number of connections that can be queued for acceptance is determined by the second argument, 5, which is given. If the listening attempt is unsuccessful, the socket is closed, an error message is shown, and the programme returns with a non-zero status. If everything goes well, a message stating that the server is listening on the chosen port is printed.

```
51      client_len = sizeof(client_addr);
52      newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);
53      if (newsockfd < 0) {
54          perror("Accepting connection error");
55          cleanup(sockfd);
56          return 1;
57      }
58      printf("Client connected\n");
```

An incoming client connection is accepted by calling the accept function. It waits until a client connects before returning a new socket file descriptor (newsockfd) that shows the connection has been made. If the acceptance is rejected, the socket is closed, an error message is printed, and the programme returns a non-zero status. A notification saying a client has connected is printed if the connection is successful.

```
60      while (1) {
61          memset(buffer, 0, sizeof(buffer));
62          ssize_t num_bytes = recv(newsockfd, buffer, BUFFER_SIZE, 0);
63          if (num_bytes <= 0) {
64              printf("Client disconnected\n");
65              break;
66          }
67          printf("Client: %s", buffer);
68
69          if (strncmp(buffer, "exit", 4) == 0) {
70              printf("Client requested to close the connection\n");
71              break;
72          }
73
74          memset(buffer, 0, sizeof(buffer));
75          printf("Server: ");
76          fgets(buffer, BUFFER_SIZE, stdin);
77
78          send(newsockfd, buffer, strlen(buffer), 0);
79      }
```

This is the main loop of the programme. It runs continually up until the user writes "exit."Memset is used inside the loop to clear the buffer. "Client:" is printed by the client to signal input. Using the fgets method, a line of input from the console is read and saved in the buffer. The trailing newline is removed from the input by replacing it with a null character.

If the message it gets is "exit", the server publishes a message and ends the loop to break the connection.The buffer is once more cleared using Memset. To solicit input, the server prints "Server:". The fgets method is utilised to read a line of input from the server's console and save it in the buffer.The buffer's contents are then sent back to the client by the server using the send method.

```
81          cleanup(newsockfd);
82          cleanup(sockfd);
83          return 0;
84      }
```

After ending the loop, the server calls the cleanup function to terminate the server and client sockets (newsockfd and sockfd, respectively). The main function returns 0 upon successful programme execution.

- **Window (Client)**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>

#define PORT 8888
#define BUFFER_SIZE 1024

int main()
{
    int sockfd;
    struct sockaddr_in server_addr;
    struct hostent *server;
    char buffer[BUFFER_SIZE];
```

First, the code includes the required header files. It defines a constant BUFFER_SIZE to indicate the size of the message buffer as well as a constant PORT to specify the port number of the server.The socket file descriptor (sockfd), server address structure (server_addr), server information (server), and message buffer (buffer) variables are initialised in the main function.

```c
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation error");
        return 1;
    }
```

To create a socket, the socket function is used. The AF_INET parameter defines the IPv4 address family, SOCK_STREAM denotes a TCP socket, and 0 denotes that the protocol should be selected at random. If the socket construction doesn't succeed, perror is used to produce an error message, and the programme exits with a non-zero status.

```c
    printf("\n|----------------------------------------------------------------|\n");
    printf("|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE            |\n");
    printf("|----------------------------------------------------------------|\n\n");
    printf("\n**********************************************************************");
    printf("\n* Welcome to the virtual space for conversation and connection. *");
    printf("\n*  Here you can talk about anything and everything with others  *");
    printf("\n*    who are interested in the same topics. Enjoy your stay!    *");
    printf("\n**********************************************************************\n");
```

The console is greeted with a welcome message that includes details about the virtual chat room in these lines.

```c
    server = gethostbyname("192.168.56.1");
    if (server == NULL) {
        fprintf(stderr, "Failed to resolve server host\n");
        close(sockfd);
        return 1;
    }
```

The gethostbyname method is used to locate the server's IP address. It is hardcoded as "192.168.56.1" in this instance. If the programme cannot resolve the server, an error message is printed, the socket is closed, and a non-zero status is returned.

```
43      server_addr.sin_family = AF_INET;
44      server_addr.sin_port = htons(PORT);
45      bcopy((char *)server->h_addr, (char *)&server_addr.sin_addr.s_addr, server->h_length);
46
47      if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
48          perror("Connection error");
49          close(sockfd);
50          return 1;
51      }
52      printf("Connected to server\n");
```

With the address family set to IPv4, the port number (which is converted to network byte order using htons), and the server's IP address retrieved from the server structure, the server address structure server_addr is configured.

The server address structure and socket file descriptor sockfd are then passed to the connect function, which is subsequently used to connect to the server. The socket is closed, an error notice is printed, and the programme returns with a non-zero status if the connection fails. If successful, a message confirming the client's connection to the server is printed.

```
62      while (1) {
63          memset(buffer, 0, sizeof(buffer));
64          printf("Client: ");
65          fgets(buffer, BUFFER_SIZE, stdin);
66          buffer[strcspn(buffer, "\n")] = '\0';
67
68          if (strncmp(buffer, "sendfile", 8) == 0) {
69
70              send(sockfd, "file", strlen("file"), 0);
71          } else {
72
73              send(sockfd, buffer, strlen(buffer), 0);
74          }
75
76          if (strncmp(buffer, "exit", 4) == 0) {
77              printf("Disconnected from server\n");
78              break;
79          }
80          memset(buffer, 0, sizeof(buffer));
81          int bytesReceived = recv(sockfd, buffer, BUFFER_SIZE, 0);
82          if (bytesReceived <= 0) {
83              if (bytesReceived == 0) {
84                  printf("Disconnected from server\n");
85              } else {
86                  fprintf(stderr, "Receive error: %d\n", WSAGetLastError());
87              }
88              break;
89          }
90
91          if (strncmp(buffer, "file", 4) == 0) {
92
93          } else {
94              printf("Server: %s\n", buffer);
95          }
96      }
97      cleanup(sockfd);
```

The client program's main loop is found here. This section captures user input to determine if the command starts with "sendfile". If it does, it sends a metadata signal ("file") to the server before initiating file transfer. If not, it assumes it's a regular message and sends the input message to the server.

When the user input starts with "sendfile", this section sends the "file" metadata to indicate that a file will be sent. However, the actual code to read the file and send its contents in chunks is a placeholder. You'd need to implement file reading and sending logic here.

Upon receiving data from the server, this section checks if the received data starts with "file" as metadata. If it does, it assumes that a file is being sent. However, this part is a placeholder for the file receiving logic. You need to implement code to receive file data in chunks and write it to a file.

Until the user enters "exit" as a message, it keeps on forever.Memset is used inside the loop to clean the buffer. Printing "Client:" is the client's way of asking for input. Using fgets, the user's input is retrieved and stored in the buffer. The last character in the input is changed from a newline to a null character to eliminate it.The socket file descriptor sockfd is used to send the message in the buffer to the server by calling the send function. Using strlen, the message's length can be calculated.

If "exit" is received as the message, the client prints a message and exits the loop to cut off the connection.Memset is used to clear the buffer once more. The recv function, which reads data from the sockfd socket and stores it in the buffer, is used by the client to receive a message from the server. BUFFER_SIZE controls the maximum number of bytes that can be read.The client then prints the message it received from the server on the console.

```
72      close(sockfd);
73      return 0;
74  }
```

The client closes the socket using close after exiting the loop and returns 0 to signify completed programme execution.

Prepare the information of the steps in demonstrating the chat application

- **Window**

How To Run (Server):

1. The chat application for windows starts with compiling the server.c file into a .exe file using the command "gcc -pthread server. c -o server -lws2_32".

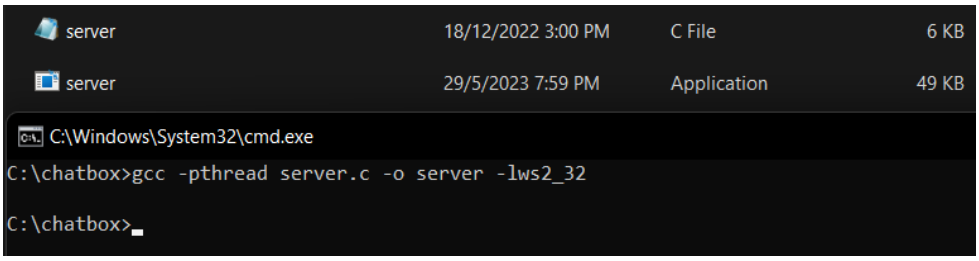

Figure 1 Shows how to compile server file

2. Then run the server.exe file with the command "server.exe ". The server port can differ according to the preferences but in this case we already assigned it. In this tutorial, we use 8888 as the server port number.

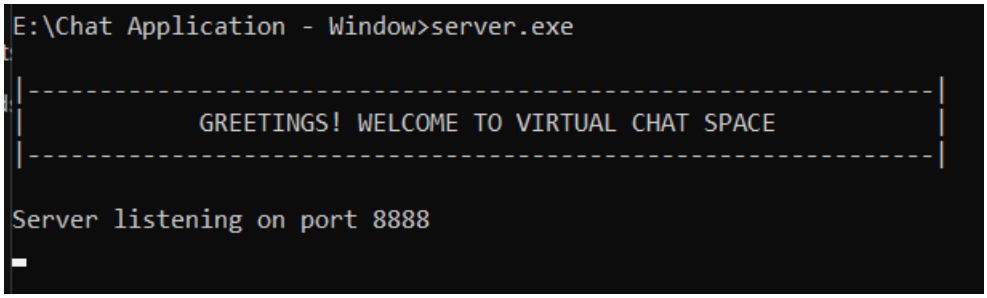

Figure 2 Figure 15: Shows the command to execute server.exe

3. Then the server wait for client to join.

How To Run (Client):

1. First, we must compile the client.c programming file into a .exe file using the command "gcc -pthread client. c -o client -lws2_32".

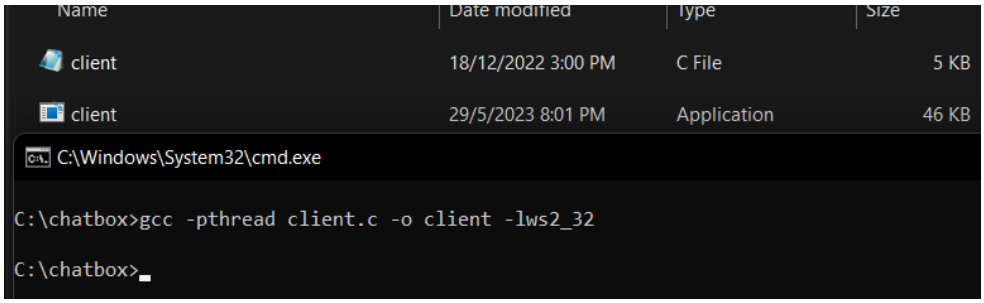

Figure 3 Shows how to compile client file

2. Then run the command "client.exe". The client port can different according to the preferences but in this case we already assigned it in this example the port number is 8888.

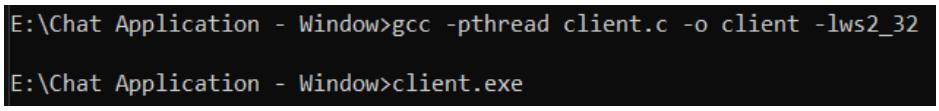

Figure 4 Shows the command to execute client.exe

3. Then, show interface and you can start sending messages to your friends.

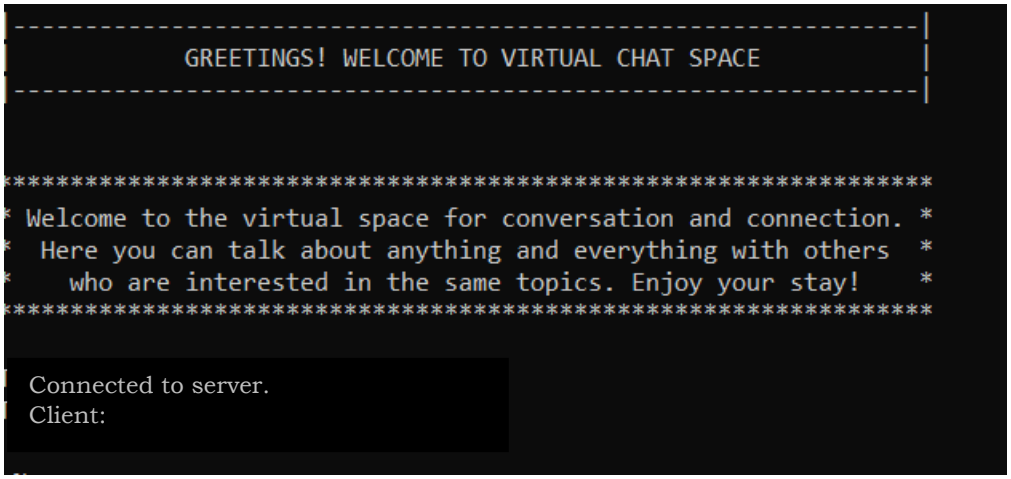

Figure 5 Shows the client interface after successfully connected

- **Ubuntu**

How To Run (Server):

1. Firstly, we are required to open terminal in Ubuntu to run this chat application.

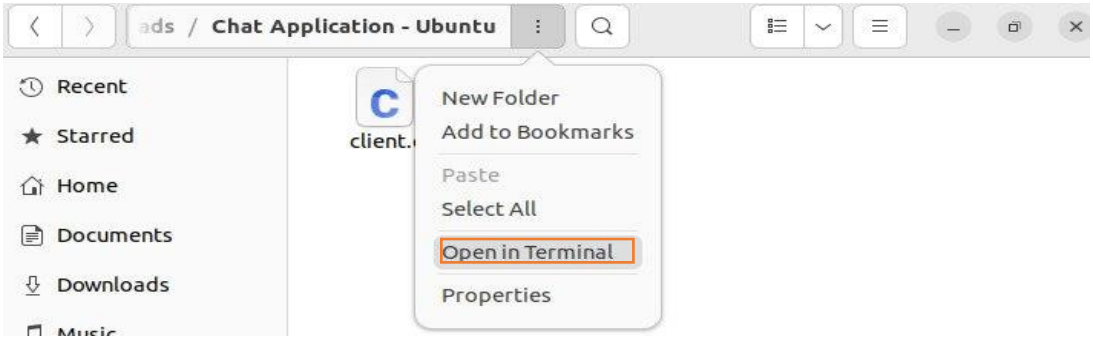

Figure 6 Show the drop-down menu for terminal

2. Then, start it by compiling the file server and client programming file using command "gcc -pthread -server.c -o server"

```
zaki@zaki-VirtualBox:~/Downloads/Chat Application - Ubuntu$ gcc -pthread server
.c -o server
```

Figure 7 Shows the command for compiling the server in chat app

3. Proceed with the command ". /server" in the new terminal for client to enter. The server port can different according to the preferences but in this case, we already assigned it in this example the port number is 8888.

```
zaki@zaki-VirtualBox:~/Downloads/Chat Application - Ubuntu$ ./server

|-------------------------------------------------------------|
|            GREETINGS! WELCOME TO VIRTUAL CHAT SPACE         |
|-------------------------------------------------------------|

Server listening on port 8888
```

Figure 8 Shows the command for compiling and start the server in chat app

4. Then, we just need to wait for the client to join by using same port number.

How To Run (Client):

1. Firstly, we are required to open terminal in Ubuntu to run this chat application.

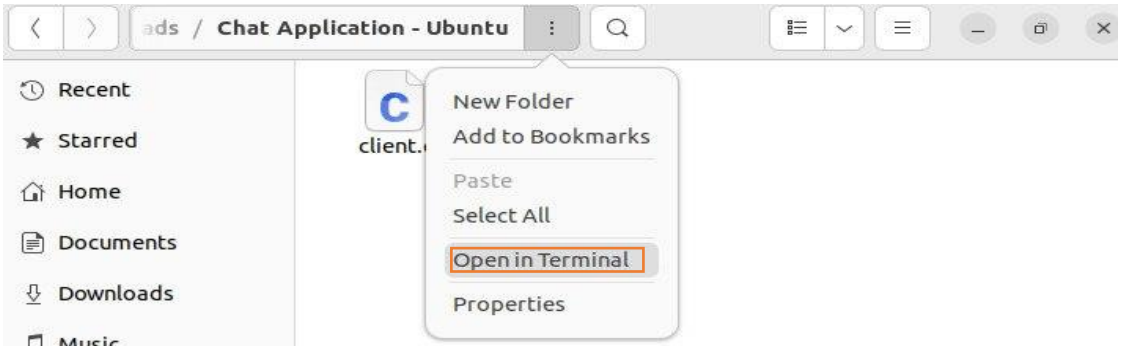

Figure 9 Show the drop-down menu for terminal

2. Then, start it by compiling the client.c file using command “gcc -pthread -client.c -o client”.

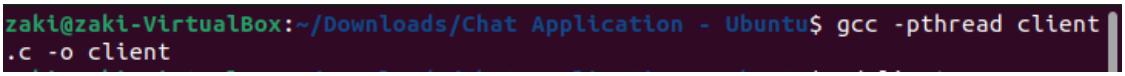

Figure 10 Shows the command for compiling the client in chat app

Proceed with the command “./client” in the new terminal for client to enter. The server port can different according to the preferences but in this case, we already assigned it in this example the port number is 8888.

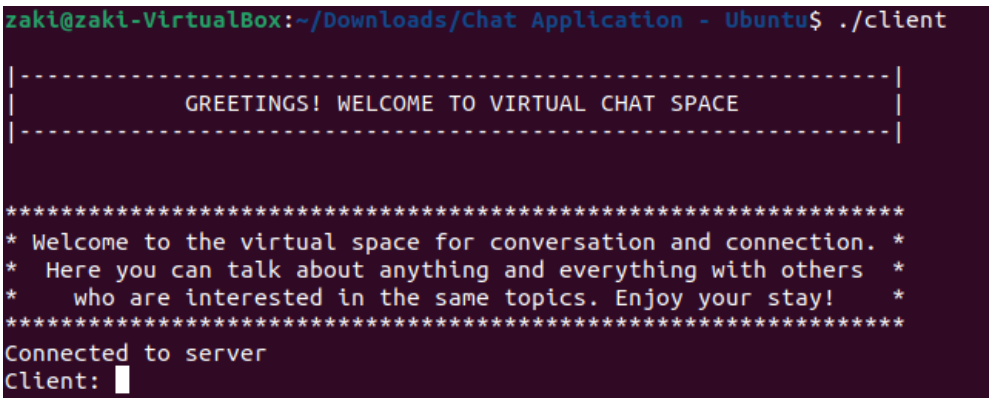

Figure 11 Show the client Interface

3. Finally, other clients can join this chat application with the same port number.

TWO (2) questions that are related to the student's chat application code

Demonstration video

Google drive :

https://drive.google.com/drive/u/0/folders/16YUa6lPWlkvIdZdLfmPjZHrj9c7vnkKW

Youtube :

https://youtu.be/wQ8zbka11sk

References

Malmberg, L. (2017). Development of a Client-Server Chat Application.

Al-Rawi, A., & Al-Rousan, M. (2018). A Secure Chat Application Using Linux and Windows. International Journal of Computer Science and Information Security, 16(1), 88-92.

Mataraarachchi, S. C., & Wedasinghe, N. (2018). Data Security System for Chat Applications Using Cryptography, Steganography and Image Processing.

Sengupta, A., Dhar, T., Das, S. K., & Ray, U. K. (2017). Cross Platform Chat Application Using ID Based Encryption. In Computational Intelligence, Communications, and Business Analytics: First International Conference, CICBA 2017, Kolkata, India, March 24–25, 2017, Revised Selected Papers, Part I (pp. 495-510). Springer Singapore.

McCune, M. The complete solutions guide for every Linux/Windows system administrator! This complete Linux/Windows integration guide offers detailed coverage of dual-boot issues, data compatibility, and networking. It also handles topics such as implementing Samba file/print services for Windows workstations and providing cross-platform database access. Running Linux and Windows in the same.

D Vhandale, A., N Gandhak, S., A Karhale, S., R Prasad, S., & A Bachwani, S. (2022). An overview of real-time chat application - IJRTI. https://www.ijrti.org/papers/IJRTI2206316.pdf

DARKO, S. A. (2014). *Network Security: A Cryptographically Approach Ofdeveloping an Instant Messaging Client-Server Chat Application Using Encryption and Decryption Algorithm on Anetwork* , 9–19. https://doi.org/https://www.academia.edu/8567863/Secured_chat_application_Network_Security_

Developing of middleware and Cross Platform Chat Application. (2021, November). https://thesai.org/Downloads/Volume12No11/Paper_10-Developing_of_Middleware_and_Cross_Platform_Chat.pdf

Using Internet Sockets. Beej's Guide to Network Programming. (n.d.). https://beej.us/guide/bgnet/

Arora, H. (2011, December 11). *C socket programming for linux with a server and client example code*. The Geek Stuff. https://www.thegeekstuff.com/2011/12/c-socket-programming/

FZE, B. B. (2023, March 21). *Simple client server chatting application information technology essay*. UKEssays. https://www.ukessays.com/essays/information-technology/simple-client-server-chatting-application-information-technology-essay.php#:~:text=Chat%20server%20is%20a%20standalone,system%20using%20loop%20back%20network.

GeeksforGeeks. (2022, November 18). *TCP server-client implementation in C*. GeeksforGeeks. https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

Stevewhims. (n.d.). *Running the Winsock client and server code sample - win32 apps*. Win32 apps | Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/winsock/finished-server-and-client-code