



BCN3033
NETWORK PROGRAMMING
PROJECT 2

TITLE: CHAT APPLICATION IN DIFFERENT OS

NAME: MUHAMMAD AFIQ BIN SHAMSUDIN
MATRIC NO.: CA21083

LECTURER:
DR. AHMAD FIRDAUS BIN ZAINAL ABIDIN

Course outcome	Marks (weight score percentage)
CO1	/40
CO2	/5
CO3	/5
	/40

TABLE OF CONTENTS

1.1 CHAT APPLICATION	3
Windows Coding.....	3
Explanation	5
Ubuntu Coding.....	7
Explanation	9
1.2 STEPS IN DEMONSTRATING	11
1.3 VIDEOS	19
1.4 SLIDES	19
1.4 REFERENCES.....	20

1.1 CHAT APPLICATION

Windows Coding

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

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];

    printf("==== Simple Chat Server =====\n");

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == INVALID_SOCKET) {
        perror("Socket creation error");
        WSACleanup();
        return 1;
    }
    printf("Server socket created\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");
        closesocket(sockfd);
        WSACleanup();
        return 1;
    }
}
```

```

if (listen(sockfd, 5) == SOCKET_ERROR) {
    perror("Listening error");
    closesocket(sockfd);
    WSACleanup();
    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");
    closesocket(sockfd);
    WSACleanup();
    return 1;
}
printf("Client connected\n");

printf("==== Chatting with the Client ==== \n");

while (1) {
    memset(buffer, 0, sizeof(buffer));
    recv(newsockfd, buffer, BUFFER_SIZE, 0);
    printf("Client: %s", buffer);

    if (strncmp(buffer, "exit", 4) == 0) {
        printf("Client disconnected\n");
        break;
    }

    memset(buffer, 0, sizeof(buffer));
    printf("Server: ");
    fgets(buffer, BUFFER_SIZE, stdin);

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

closesocket(newsockfd);
closesocket(sockfd);
WSACleanup();
return 0;
}

```

Explanation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <winsock2.h>
5  #pragma comment(lib, "ws2_32.lib")
6
7  #define PORT 8888
8  #define BUFFER_SIZE 1024
```

- This section includes the necessary header files and defines constants for the port number and buffer size.

```
10 int main()
11 {
12     WSADATA wsaData;
13     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
14         fprintf(stderr, "Failed to initialize Winsock.\n");
15         return 1;
16     }
17 }
```

- The main() function begins here. It initializes the Winsock library using the WSAStartup function.

```
17
18     SOCKET sockfd, newsockfd;
19     struct sockaddr_in server_addr, client_addr;
20     int client_len;
21     char buffer[BUFFER_SIZE];
22 }
```

- Declare variables for socket descriptors, server and client address structures, client length, and a buffer to hold messages.

```
24
25     sockfd = socket(AF_INET, SOCK_STREAM, 0);
26     if (sockfd == INVALID_SOCKET) {
27         perror("Socket creation error");
28         WSACleanup();
29         return 1;
30     }
31     printf("Server socket created\n");
32 }
```

- Creates a socket using the socket function with the TCP protocol (SOCK_STREAM). If the socket creation fails, it prints an error message and terminates the program.

```

33     server_addr.sin_family = AF_INET;
34     server_addr.sin_port = htons(PORT);
35     server_addr.sin_addr.s_addr = INADDR_ANY;
36

```

- Sets up the server address structure with the IP address (set to INADDR_ANY) and port number.

```

37     if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
38         perror("Binding error");
39         closesocket(sockfd);
40         WSACleanup();
41         return 1;
42     }

```

- Associates the socket with a specific IP address and port using the bind function. If the binding fails, it prints an error message and terminates the program.

```

44     if (listen(sockfd, 5) == SOCKET_ERROR) {
45         perror("Listening error");
46         closesocket(sockfd);
47         WSACleanup();
48         return 1;
49     }
50     printf("Server listening on port %d\n", PORT);

```

- Sets the socket to the listening state to accept incoming connections using the listen function. If the listening fails, it prints an error message and terminates the program.

```

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

```

- Accepts a client connection using the accept function. If the connection acceptance fails, it prints an error message and terminates the program.

Ubuntu Coding

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

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("Client socket created\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");
```

```

// Greeting
printf("Welcome to the Chat Client!\n");

while (1) {
    memset(buffer, 0, sizeof(buffer));
    printf("Client: ");
    fgets(buffer, BUFFER_SIZE, stdin);

    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", buffer);
}

close(sockfd);
return 0;
}

```


Explanation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8  #include <netdb.h>
9  #include <unistd.h>
10
11  #define PORT 8888
12  #define BUFFER_SIZE 1024
13
```

- This section includes the necessary header files and defines constants for the port number and buffer size.

```
14  int main()
15  {
16      int sockfd;
17      struct sockaddr_in server_addr;
18      struct hostent *server;
19      char buffer[BUFFER_SIZE];
20
21      sockfd = socket(AF_INET, SOCK_STREAM, 0);
22      if (sockfd < 0) {
23          perror("Socket creation error");
24          return 1;
25      }
26      printf("Client socket created\n");
27
```

- The main() function begins here. It declares variables for the socket descriptor, server address structure, server host information, and a buffer to hold messages.
- Creates a socket using the socket function with the TCP protocol (SOCK_STREAM). If the socket creation fails, it prints an error message and terminates the program.

```
27
28
29  server = gethostbyname("192.168.56.1");
30  if (server == NULL) {
31      fprintf(stderr, "Failed to resolve server host\n");
32      close(sockfd);
33      return 1;
34  }
35
```

- Retrieves the host information using the gethostbyname function. In this case, the server host is specified as "192.168.56.1". If the resolution fails, it prints an error message and terminates the program.

```
34
35  server_addr.sin_family = AF_INET;
36  server_addr.sin_port = htons(PORT);
37  bcopy((char *)server->h_addr, (char *)&server_addr.sin_addr.s_addr, server->h_length);
38
```

- Sets up the server address structure with the IP address and port number obtained from the host information.

```

39     if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
40         perror("Connection error");
41         close(sockfd);
42         return 1;
43     }
44     printf("Connected to server\n");
45

```

- Establishes a connection to the server using the connect function. If the connection fails, it prints an error message and terminates the program.

```

49     while (1) {
50         memset(buffer, 0, sizeof(buffer));
51         printf("Client: ");
52         fgets(buffer, BUFFER_SIZE, stdin);
53
54         send(sockfd, buffer, strlen(buffer), 0);
55
56         if (strcmp(buffer, "exit", 4) == 0) {
57             printf("Disconnected from server\n");
58             break;
59         }
60
61         memset(buffer, 0, sizeof(buffer));
62         recv(sockfd, buffer, BUFFER_SIZE, 0);
63         printf("Server: %s", buffer);
64     }

```

- Enters a loop that allows the client to send and receive messages with the server.
- Reads user input from the console using fgets and sends it to the server using the send function.
- If the user enters "exit", it prints a message indicating disconnection and breaks out of the loop.
- Receives messages from the server using the recv function and prints them.

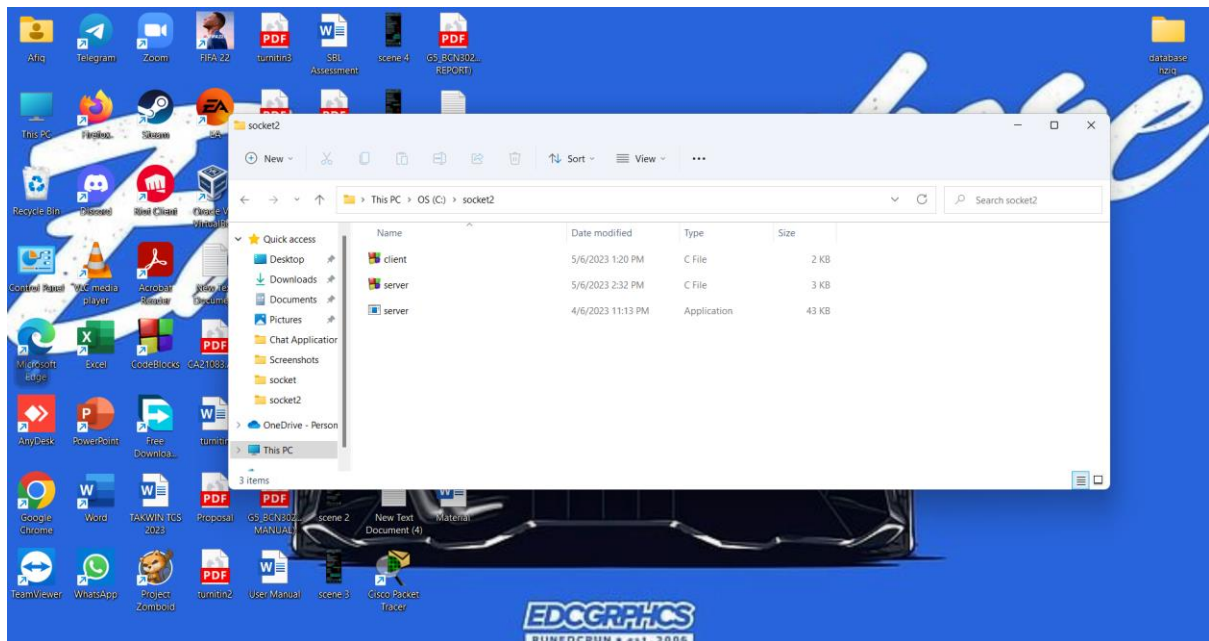
```

65
66     close(sockfd);
67     return 0;
68 }

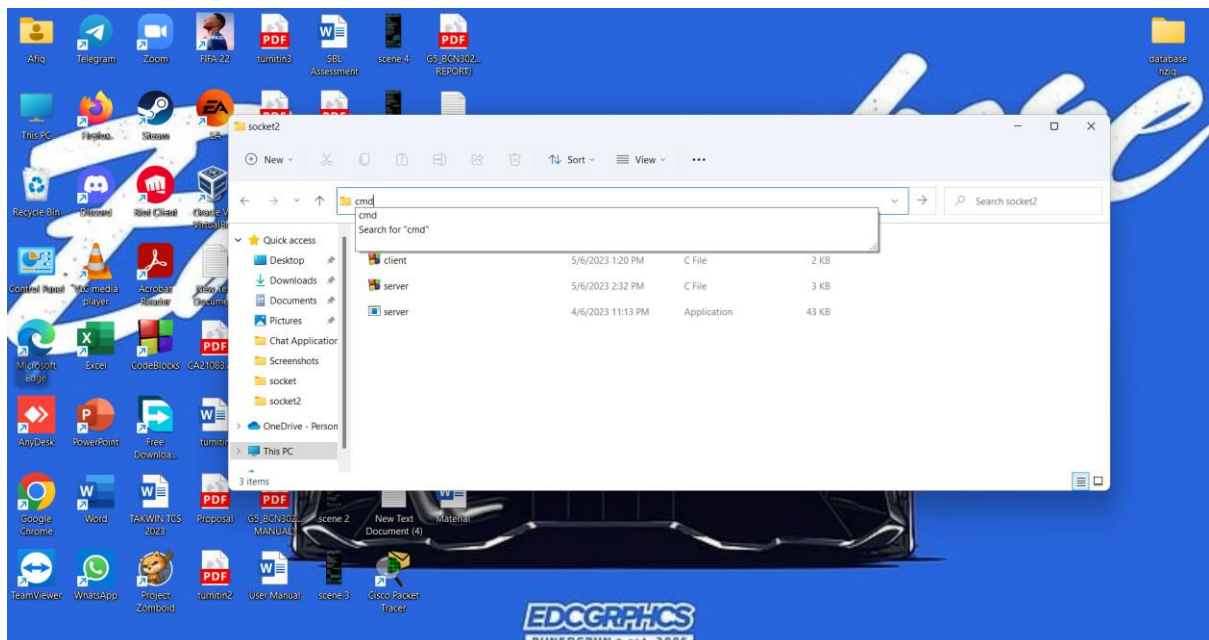
```

- Closes

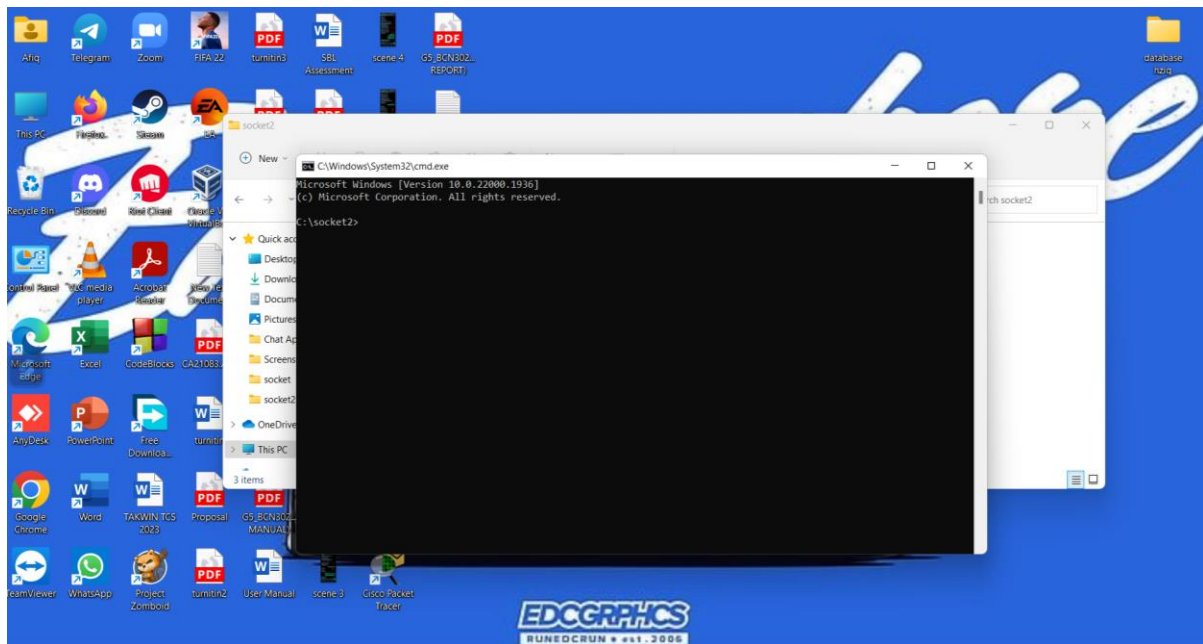
1.2 STEPS IN DEMONSTRATING



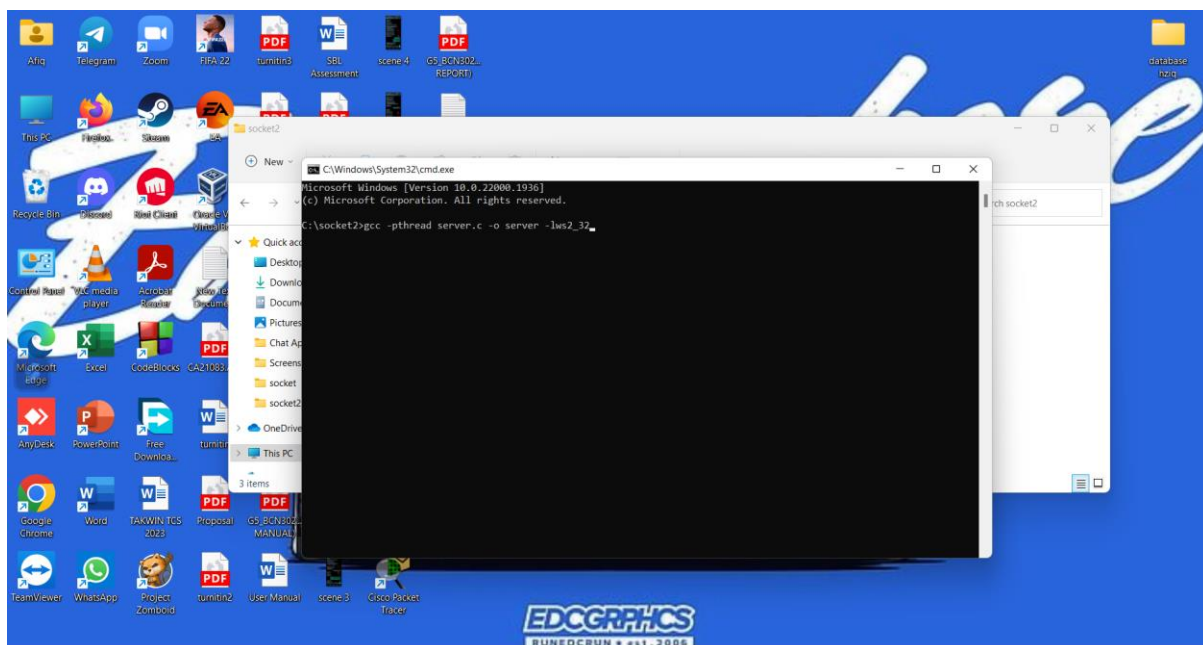
1. Open the file where the coding is saved



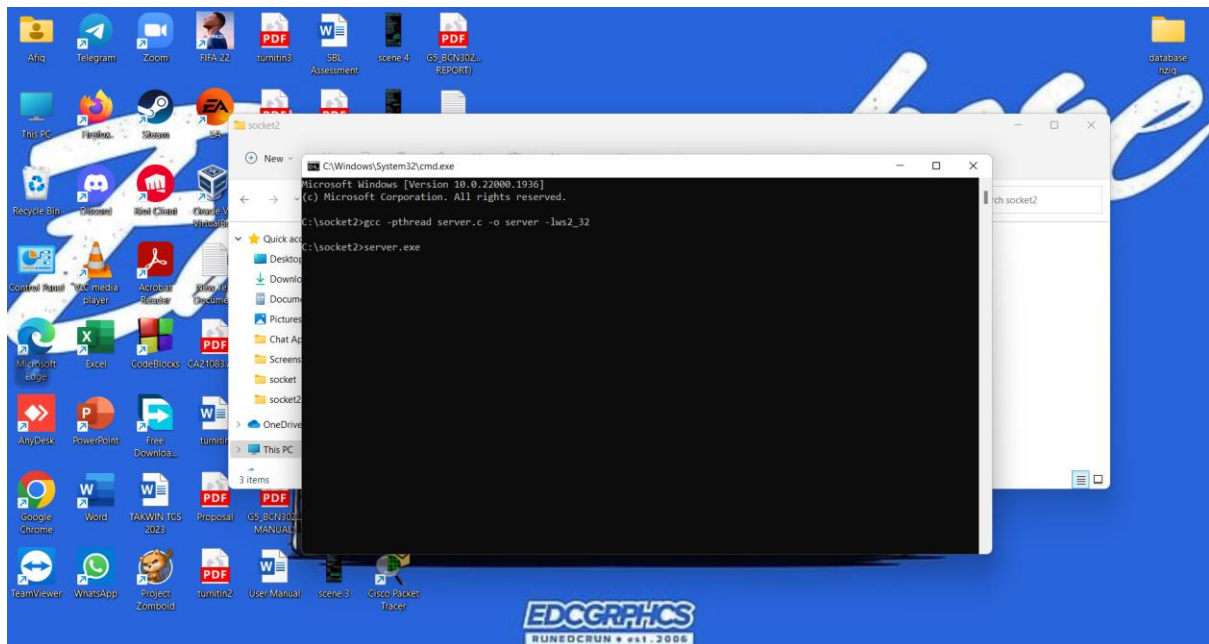
2. Open the command prompt from the file



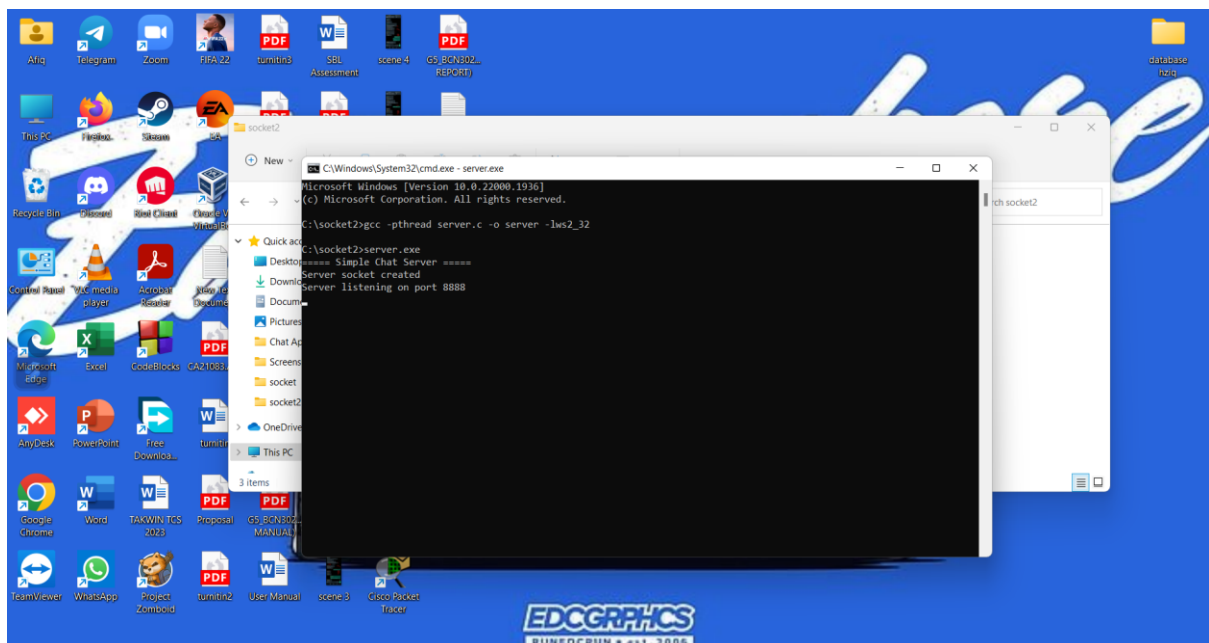
3. You can see the command prompt is opened



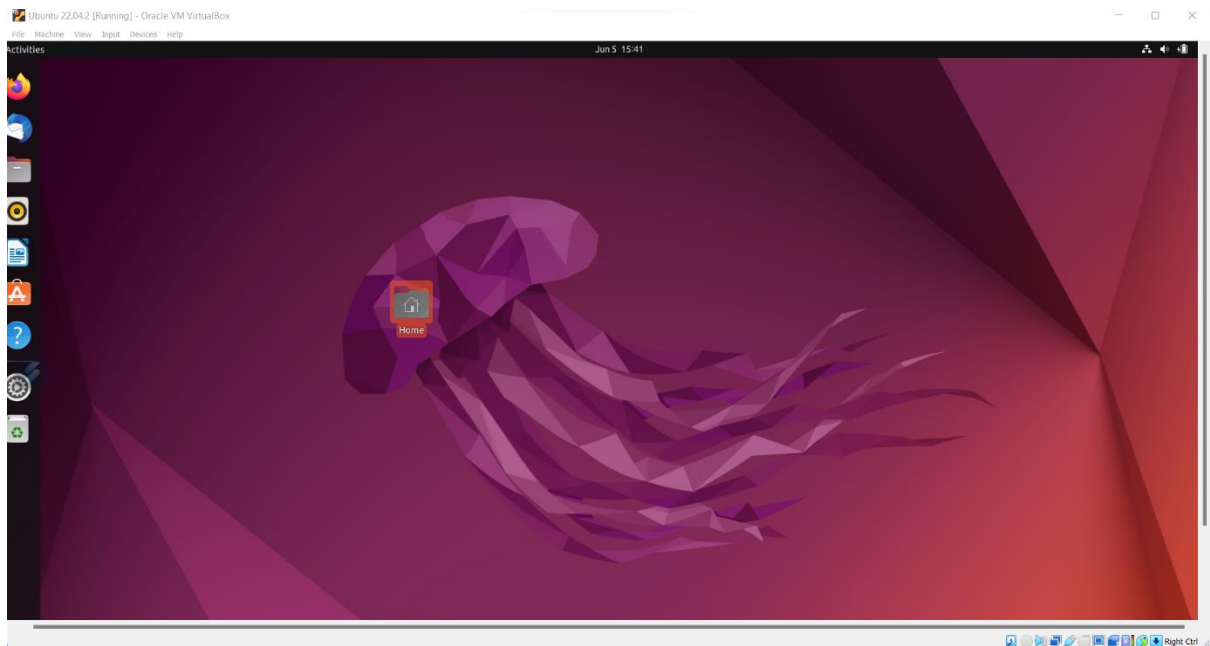
4. Enter the “gcc -pthread server.c -o server -lws2_32” command to compile the server.c coding



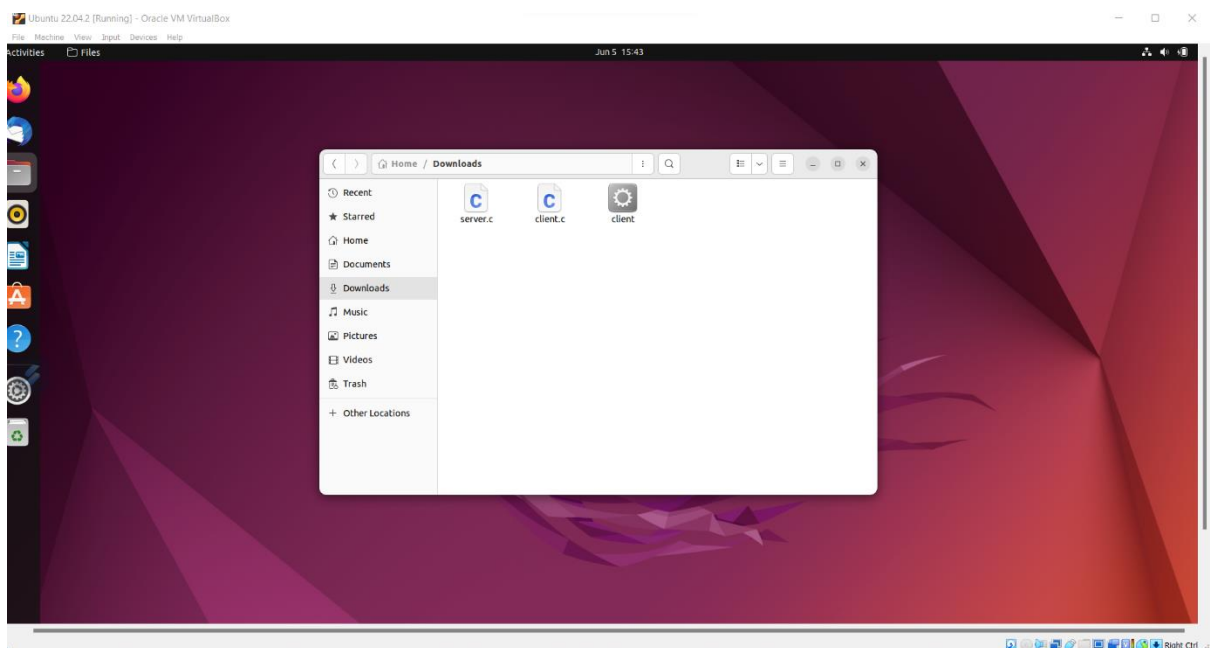
5. After done the compiling, run the coding by enter command “server.exe”



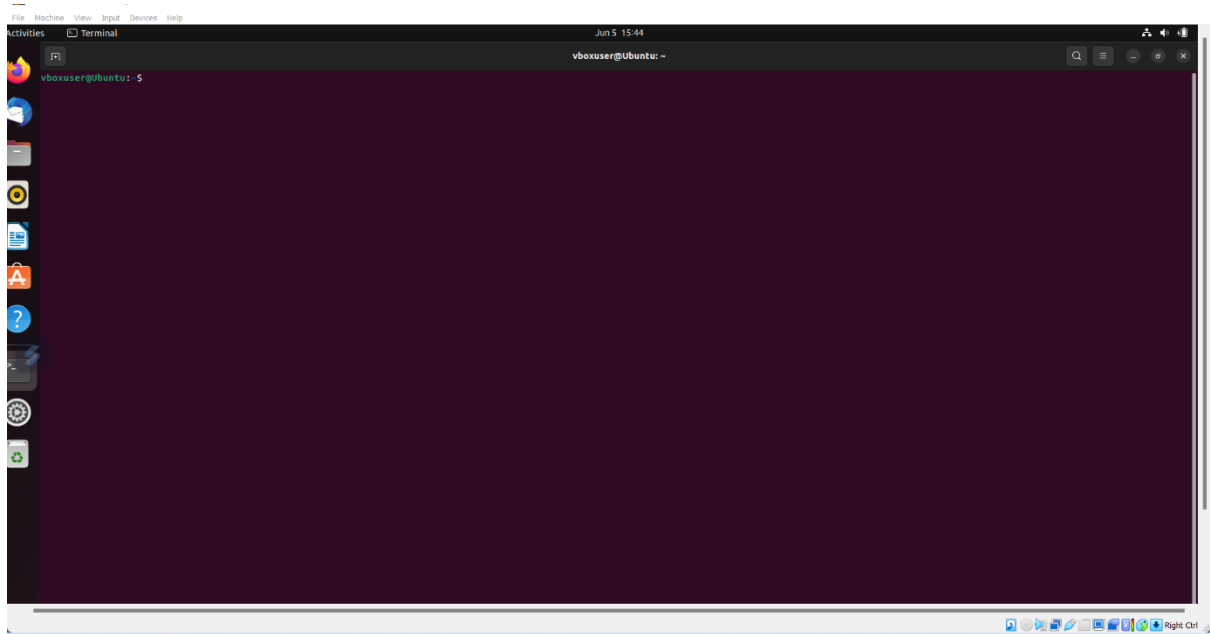
6. You can see that the coding is running properly. The server is waiting the response from the client side



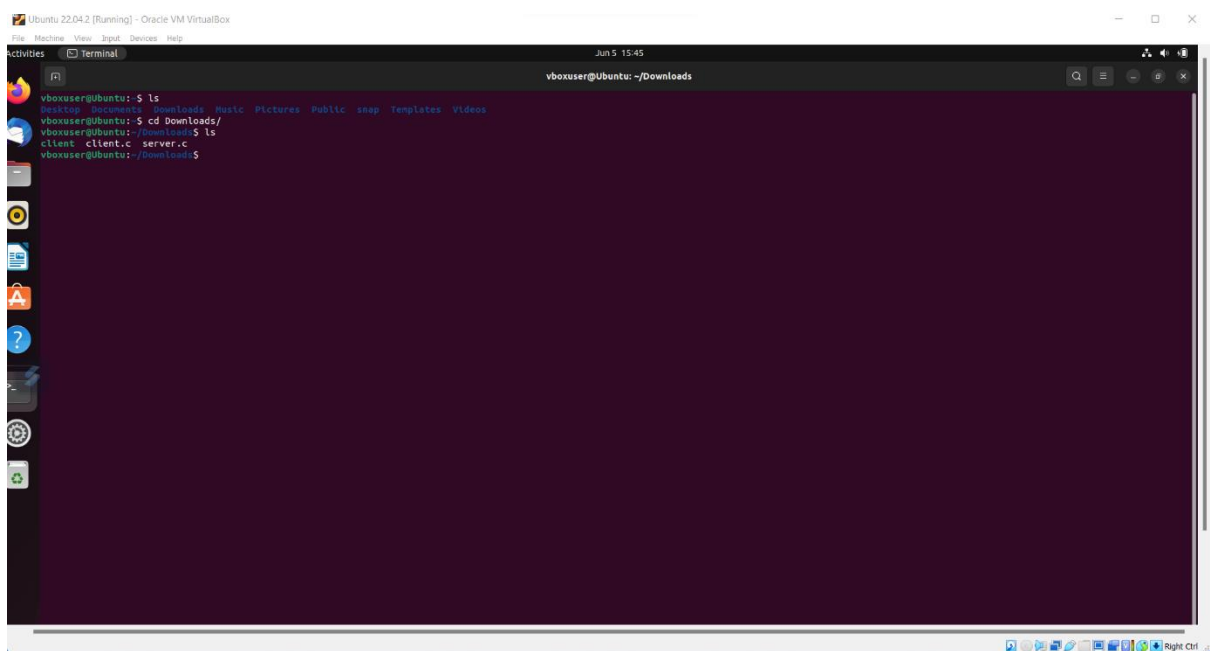
7. Open your Ubuntu application



8. Download the coding file into the Ubuntu



9. Open the terminal in Ubuntu



10. Enter command "ls" to view all the listed file. Then, change the directory to Downloads where our coding file is located.

The screenshot shows a terminal window titled "vboxuser@Ubuntu: ~/Downloads" with a dark purple background. The terminal displays the following commands and output:

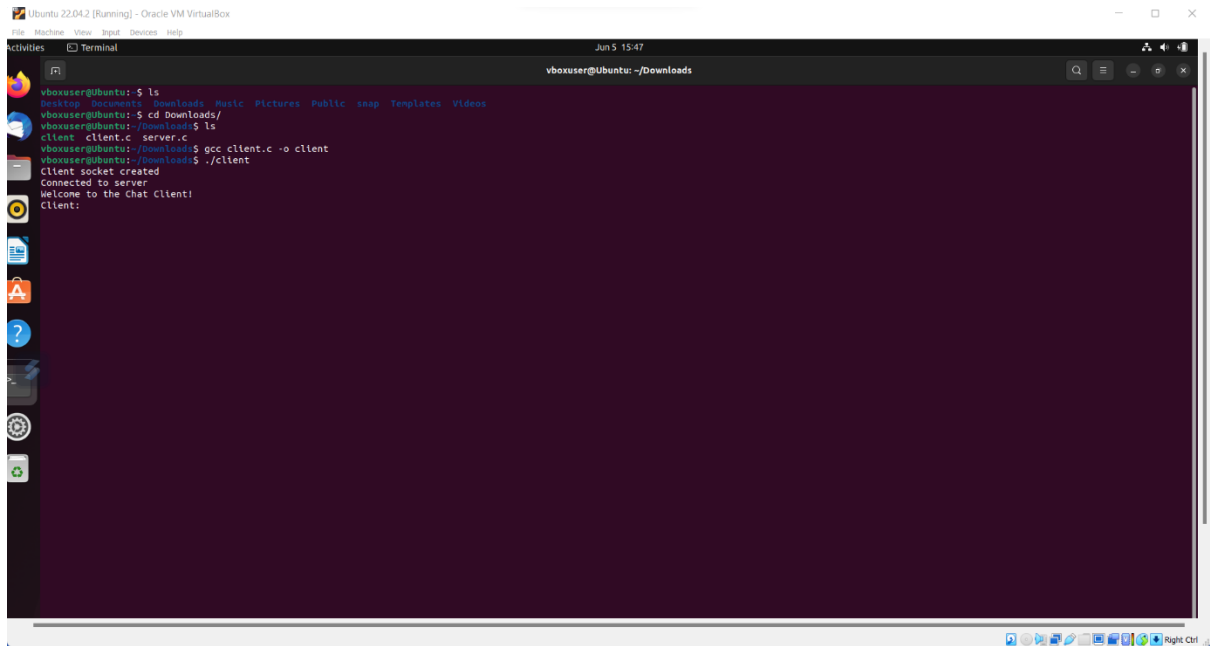
```
vboxuser@ubuntu:~$ ls
Desktop Documents Downloads Music Pictures Public snap Templates Videos
vboxuser@ubuntu:~$ cd Downloads/
vboxuser@ubuntu:~/Downloads$ ls
client client.c server.c
vboxuser@ubuntu:~/Downloads$ gcc client.c -o client
```

11. Enter the command “gcc client.c -o client” to compile the client.c coding

The screenshot shows the same terminal window as above, but with an additional command and output:

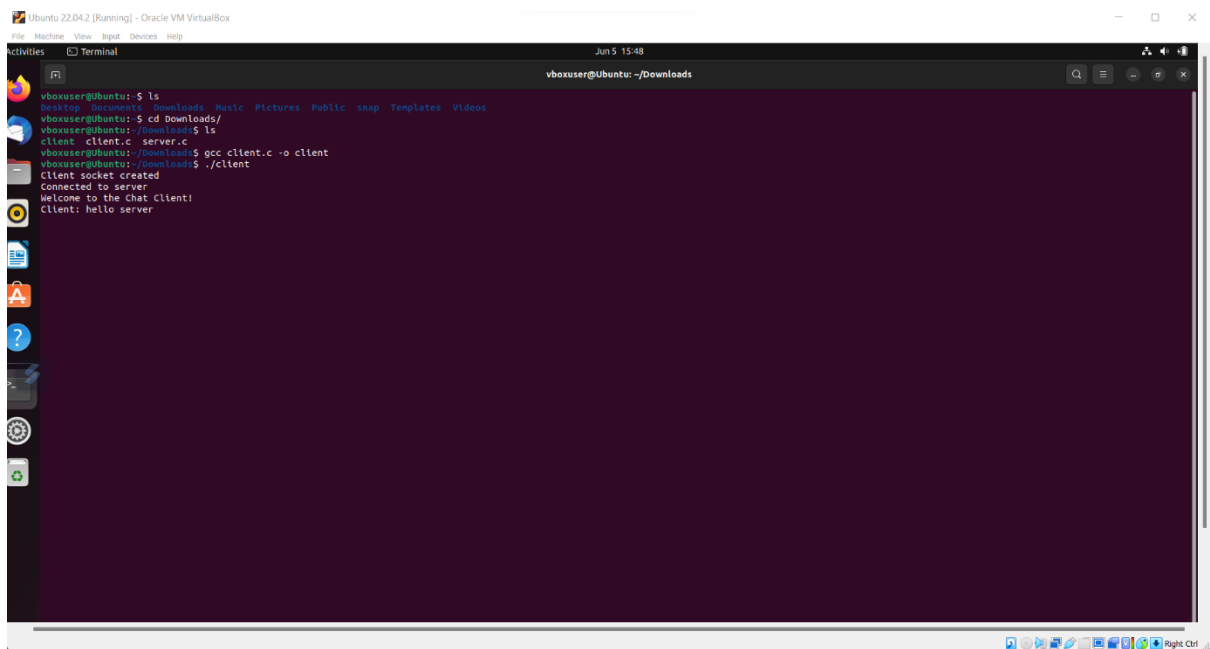
```
vboxuser@ubuntu:~$ ls
Desktop Documents Downloads Music Pictures Public snap Templates Videos
vboxuser@ubuntu:~$ cd Downloads/
vboxuser@ubuntu:~/Downloads$ ls
client client.c server.c
vboxuser@ubuntu:~/Downloads$ gcc client.c -o client
vboxuser@ubuntu:~/Downloads$ ./client
```

12. Enter command “./client” to run the coding



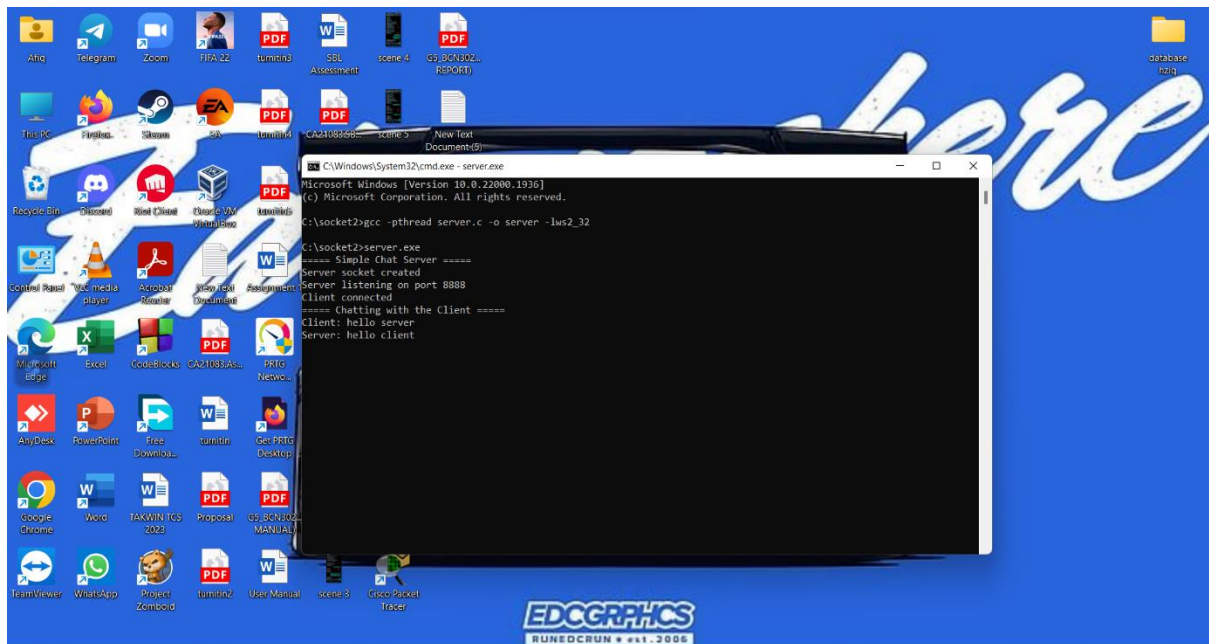
```
vboxuser@Ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
vboxuser@Ubuntu:~$ cd Downloads/
vboxuser@Ubuntu:~/Downloads$ ls
client  client.c  server.c
vboxuser@Ubuntu:~/Downloads$ gcc client.c -o client
vboxuser@Ubuntu:~/Downloads$ ./client
Client socket created
Connected to server
Welcome to the Chat Client!
Client:
```

13. You can see that the coding is run correctly and the client has connected to the server in windows



```
vboxuser@Ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
vboxuser@Ubuntu:~$ cd Downloads/
vboxuser@Ubuntu:~/Downloads$ ls
client  client.c  server.c
vboxuser@Ubuntu:~/Downloads$ gcc client.c -o client
vboxuser@Ubuntu:~/Downloads$ ./client
Client socket created
Connected to server
Welcome to the Chat Client!
Client: hello server
```

14. Now, you can create a chat with the server



15. The server received the chat from the client

1.3 VIDEOS

<https://youtu.be/WzbwvAztREE>

1.4 SLIDES

https://drive.google.com/file/d/1-d1RM0FfDxigelN9O_5x_hAx4LKvzJO6/view?usp=sharing

1.4 REFERENCES

1. Pranavdheer, (2020).:Real-time-chat-application-in-c. Github.
<https://github.com/pranavdheer/Real-time-chat-application-in-c>
2. Simple Chat Server using Sockets in C. (2018). SecuroSoft.
<https://medium.com/@securosoft/simple-chat-server-using-sockets-in-c-f72fc8b5b24e>
3. Bill Farley. (2005). Socket Chat.
<https://www.c-sharpcorner.com/article/socket-chat/>
4. Allexy. (2006). TCP/IP Chat Application Using C#.
<https://www.codeproject.com/Articles/12893/TCP-IP-Chat-Application-Using-C>
5. Karim Oumghar. (2018). Client-Server chat in C++ using sockets.
<https://simpledevcode.wordpress.com/2016/06/16/client-server-chat-in-c-using-sockets/>