Computer Networks

Lab

Assignment – 1

Report

Name: **Farrukh Riaz**

Roll: **22i-1669**

Sec: **CY-B**

1. Creating a TCP client-server connection using C code:

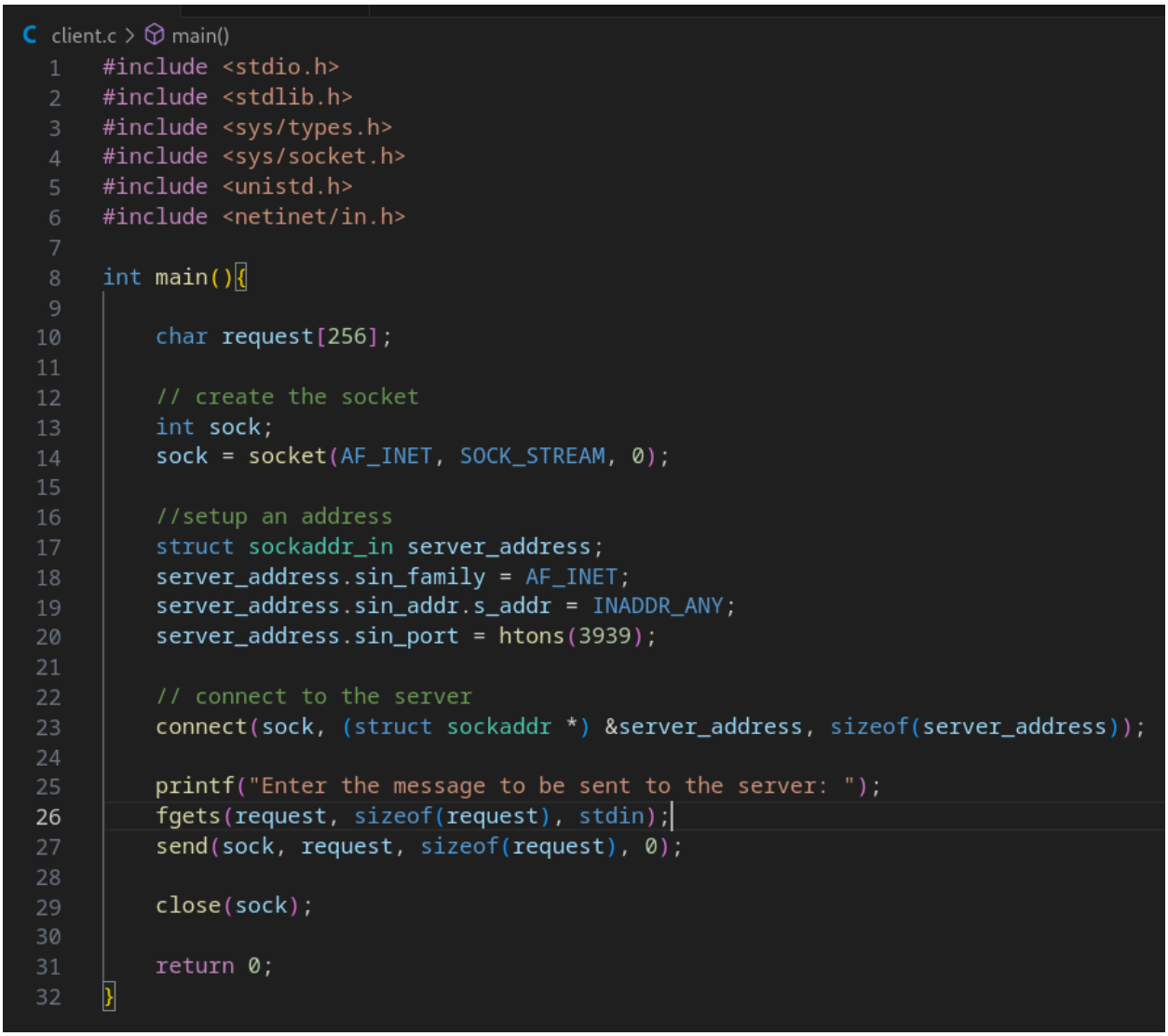
Server Side:

server.c > main()

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

int main(){
    char buf[200];

    // create the server socket
    int server_socket;
    server_socket = socket(AF_INET, SOCK_STREAM, 0);

    // define the server address
    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(3939);
    server_address.sin_addr.s_addr = INADDR_ANY;

    // bind the socket to our specified IP and port
    bind(server_socket, (struct sockaddr*) &server_address, sizeof(server_address));

    // listen for connections
    listen(server_socket, 5);

    // accept the connection
    int client_socket;
    client_socket = accept(server_socket, NULL, NULL);

    // communication
    recv(client_socket, &buf, sizeof(buf), 0);
    printf("\n%s\n", buf);

    // close the socket
    close(server_socket);

    return 0;
}
```

Creating a server agent that will listen and accept connection from clients. Created a **200-byte** buffer in which data will be received and stored from any client. Created a socket with IP info and connection type (TCP). Created a server address instance that shows entity with some identity info like IP, Port **(3939)**, etc. In bind function we are passing socket and server address instance to bind our socket and server address, further we are listening for open connection with limit of **5**. Then used the accept function to accept any client request for connection. Now we are ready for communication by only accepting any data from the client side.

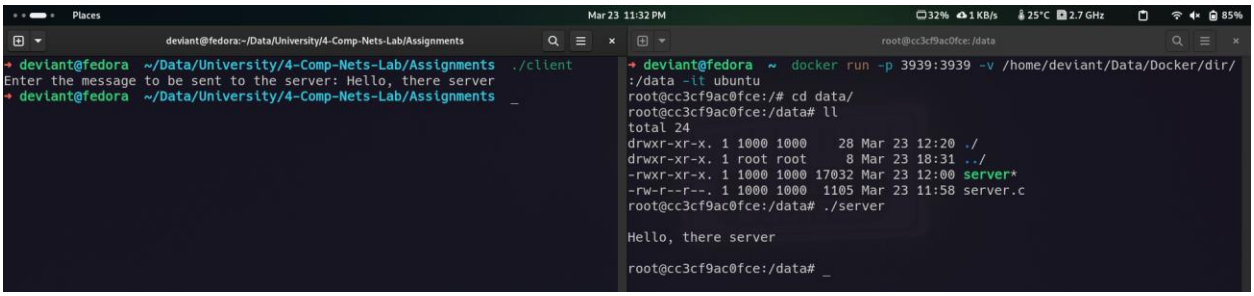
Client Side:



```c
client.c > main()
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>

int main(){

    char request[256];

    // create the socket
    int sock;
    sock = socket(AF_INET, SOCK_STREAM, 0);

    //setup an address
    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(3939);

    // connect to the server
    connect(sock, (struct sockaddr *) &server_address, sizeof(server_address));

    printf("Enter the message to be sent to the server: ");
    fgets(request, sizeof(request), stdin);
    send(sock, request, sizeof(request), 0);

    close(sock);

    return 0;
}
```

Creating client agent that will be connected to server to access the server's services. Client side is also same until connect function in which we are passing our socket and server address to connect to the specific Port and IP. Using connect function to connect to the server by passing created socket. When the server accepts the client's connection request messages can be sent to the server, after sending data to server, connection is being closed on client side.

2. TCP connection and communication between client and server:



Because we need to 2 devices on a network, so we created and docker container and mapped our container 3939 port to docker host

Command: `docker run -p 3939:3939 -v /volume_on_host/:/mount_on_container/ -it ubuntu`

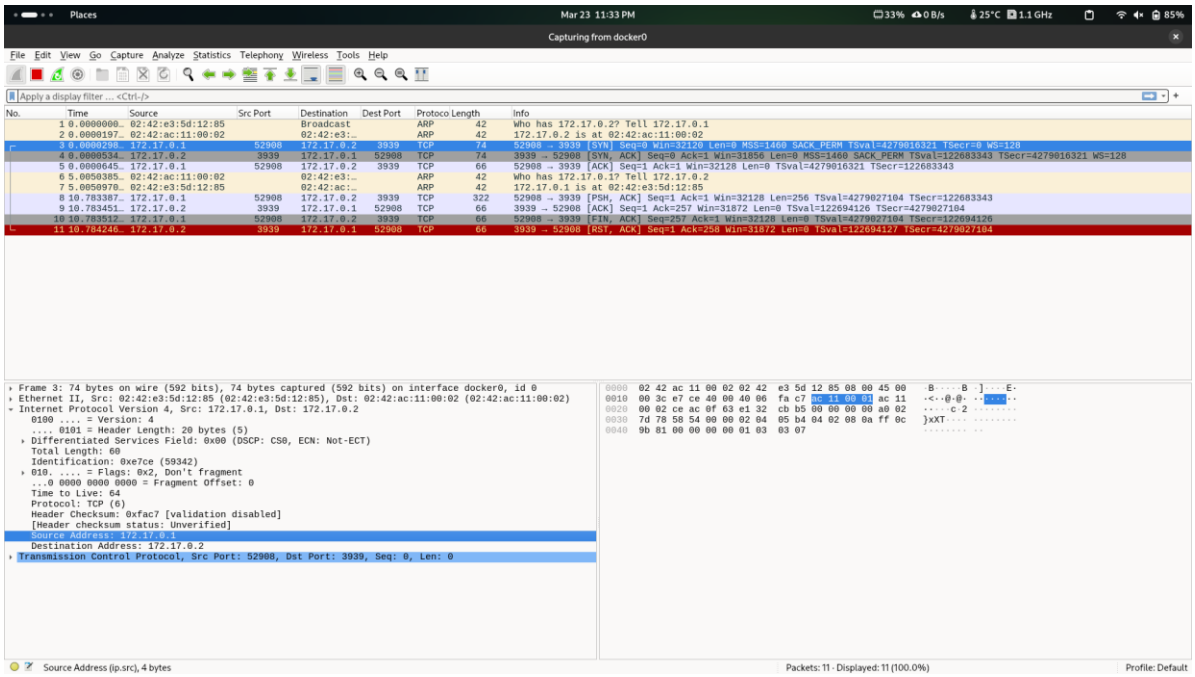
-p is mapping port between docker host and docker container.

-v is mounting volume between docker host and docker container.

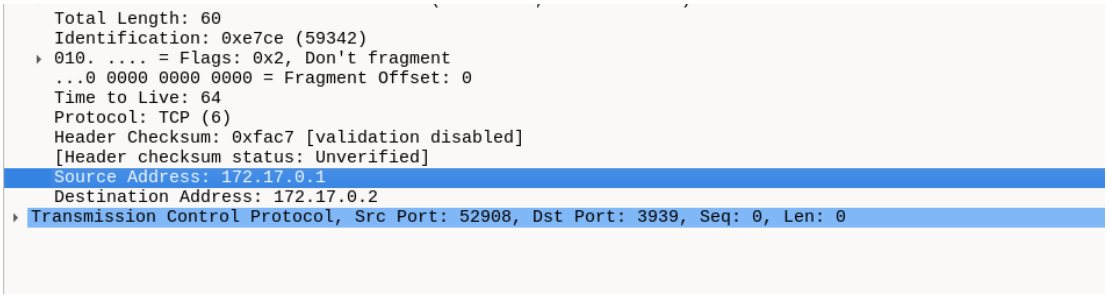
-it is running container in interactive mode; means we are using container's stdin and stdout file descriptors

Our server is running on ubuntu machine on the same network and accepting connection and data. Client sent **"Hello, there server"** message to server and server received that message and printed on the interactive terminal.

3-5. Capturing network traffic via wireshark during communications:



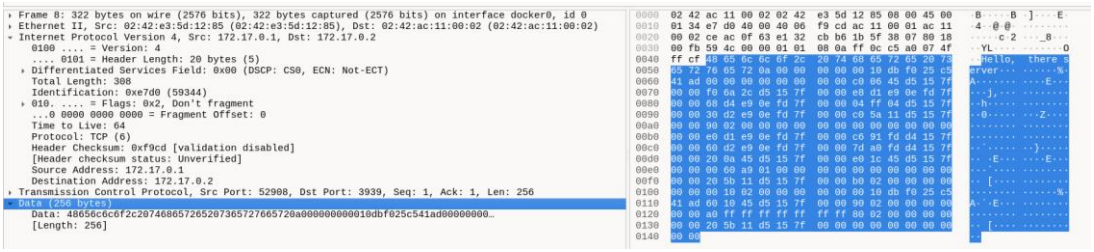
6-7. Source and Destination IP:



Source IP is **172.17.0.1**, which is the IP of our **host client**.

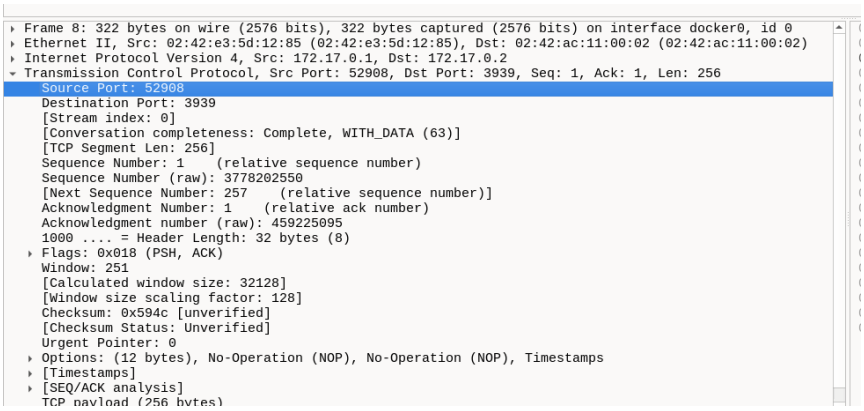
Destination IP is **172.17.0.2**, which is **server container's** IP.

8. Sniffing packet data:



As we can see on the bottom left side Data Length 256 because we created 256-byte buffer on client-side code that was sent to the server. On the right side we can see data of that buffer that was sent from client to server. In the beginning of buffer **"Hello, there server"** is written, it is the same data that we sent from client side to server.

9-10. Source and Destination Port:



Source Port 52908 which was random.

Destination Port 3939, because our server was **listening on port 3939** as we wrote in the code.

11. Modifying client and server code to make it harder to sniff the data:

Client Side:

```
23      connect(sock, (struct sockaddr *) &server_address, sizeof(server_address));
24
25      printf("Enter the message to be sent to the server: ");
26      fgets(request, sizeof(request), stdin);
27      // modify the message to be sent to the server that it is not easy to sniff
28      for(int i = 0; i < sizeof(request); i++){
29          request[i] = request[i] + 1;
30      }
31      send(sock, request, sizeof(request), 0);
32
33      close(sock);
34
35      return 0;
36  }
```

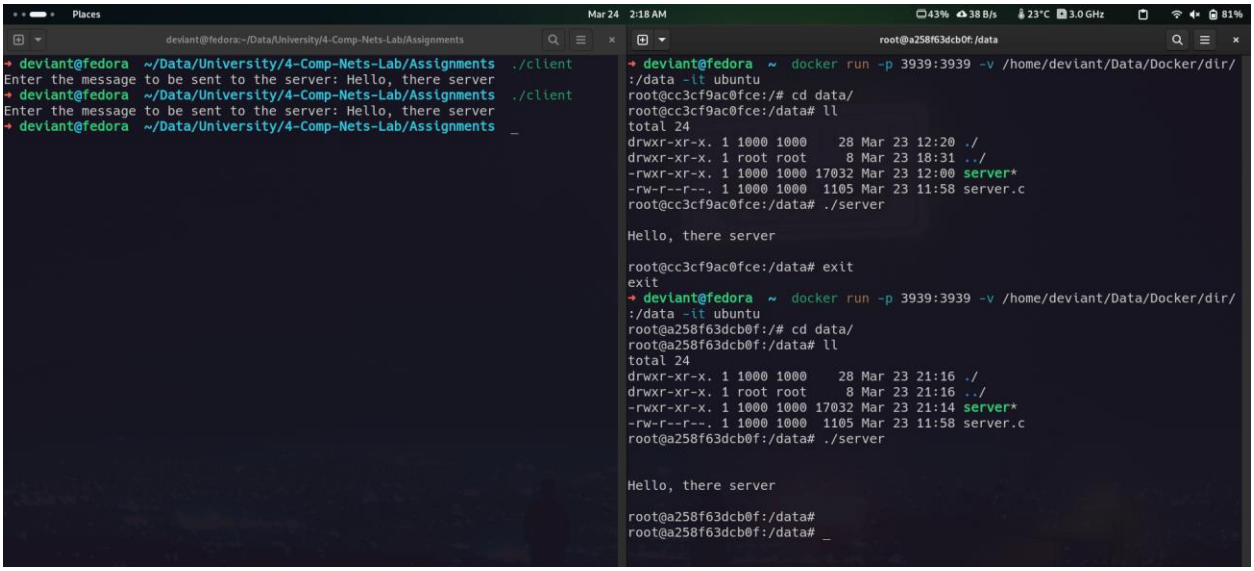
Before sending data to the server, we applied simple encryption on data to make it harder to read it on the communication channel.

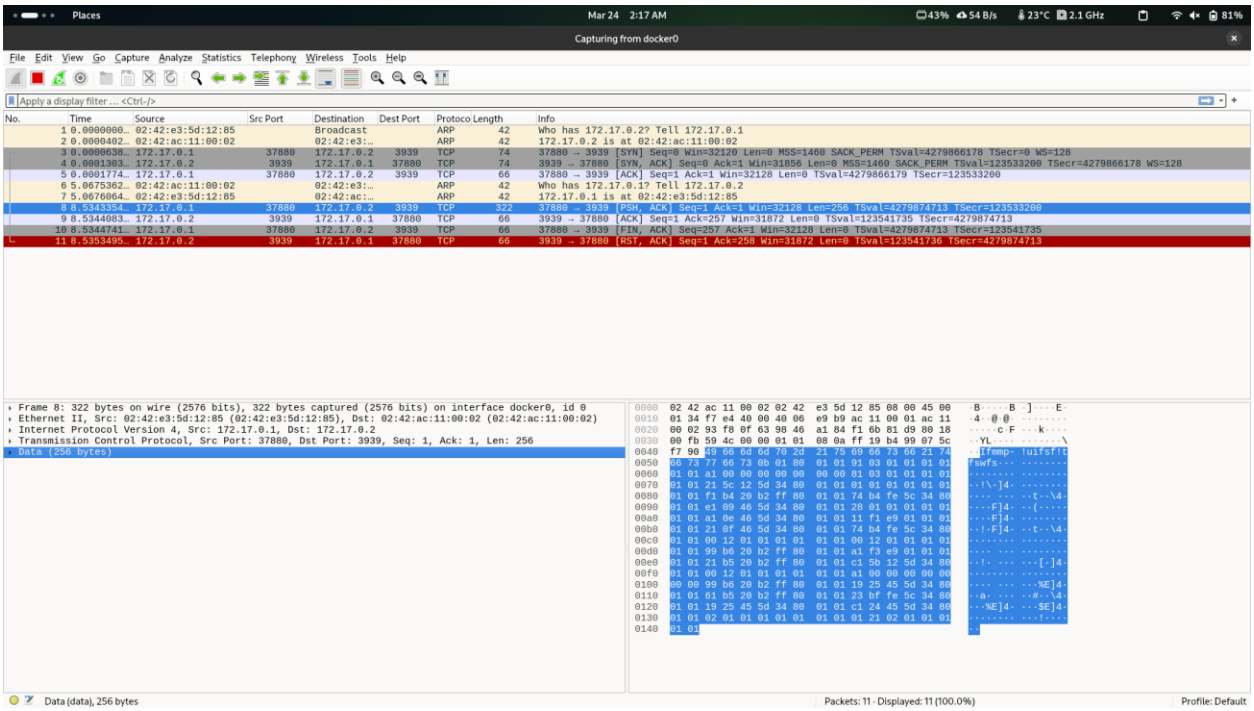
Server Side:

```
31      // communication
32      recv(client_socket, &buf, sizeof(buf), 0);
33      for(int i=0; i<sizeof(buf); i++){
34          buf[i] = buf[i] - 1;
35      }
36      printf("\n%s\n", buf);
37
38      // close the socket
39      close(server_socket);
40
41      return 0;
42  }
```

Server side reads the data sent from client, and before printing it on the screen, it applied decryption on the data.

12. Observing secure communication via wireshark:





We again captured data during TCP connection communication and tried to sniff that data which is **“lfmmp- !uifsf!t fswff”,** but this time our data is in random form, and it is not understandable.

In this way we had secure communication on the communication channel.