COMPUTER NETWORKS PRACTICE ASSIGNMENT 1

Name: Vinayak Sethi Roll No: COE18B061

Linux Network Commands

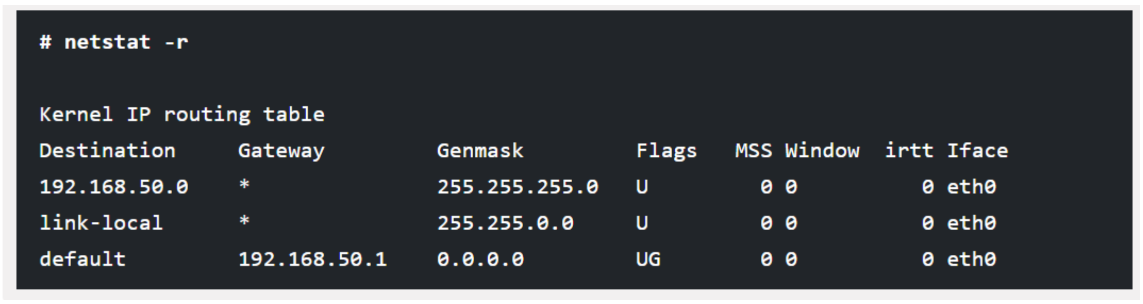
- **ifconfig:** It stands for (**interface configurator**) command is use to initialize an interface, assign **IP Address** to interface and **enable** or **disable** interface on demand. With this command you can view **IP Address** and **Hardware** / **MAC address** assign to interface and also **MTU** (**Maximum transmission unit**) size.

- **ping:** It stands for **Packet INternet Groper** command is the best way to test connectivity between **two nodes**. Whether it is **Local Area Network** (**LAN**) or **Wide Area Network** (**WAN**). Ping use **ICMP** (**Internet Control Message Protocol**) to communicate to other devices.

- **traceroute:** It is a network troubleshooting utility which shows number of hops taken to reach destination also determine packets traveling path. Below we are tracing route to global **DNS server IP Address** and able to reach destination also shows path of that packet is traveling.

- **telnet:** It connects destination host: port via a telnet protocol if connection establishes means connectivity between two hosts is working fine.

- **netstat:** It stands for (**Network Statistic**) command display connection info, routing table information etc. To displays routing table information use option as **-r**.



```
# netstat -r

Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.50.0    *               255.255.255.0   U         0 0          0 eth0
link-local      *               255.255.0.0     U         0 0          0 eth0
default         192.168.50.1    0.0.0.0         UG        0 0          0 eth0
```

- **dig:** It stands for (**domain information groper**) query **DNS** related information like **A Record**, **CNAME**, **MX Record** etc. This command mainly uses to troubleshoot **DNS** related query.

- **nslookup: nslookup** command is used to find out **DNS** related query.

- **route:** This command used to show and manipulate **ip** routing table.

- **host:** This command to find name to **IP** or **IP** to name in **IPv4** or **IPv6** and also query **DNS** records.

- **arp:** It stands for (Address Resolution Protocol) is useful to **view** / **add** the contents of the kernel's **ARP tables**.

- **ethtool:** It is used to view or modify the setting like speed and duplex of your **Network Interface Card** (**NIC**).

- **iwconfig:** This command is use to configure a **wireless network interface**.

- **hostname:** It is used to view or modify a computer's hostname.

- **scp:** It allows you to secure copy files to and from another host in the network.

- **w:** It prints a summary of the current activity on the system, including what each user is doing, and their processes. Also list the logged in users and system load average for the past 1, 5, and 15 minutes.

- **nmap:** It is a one of the powerful commands, which checks the opened port on the server.

- **Enable/Disable Network Interface:** We can enable or disable the network interface by using ifup/ifdown commands with ethernet interface parameter.
    - To **enable eth0** use **#ifup eth0**
    - To **disable eth0** use **#ifdown eth0**

- **ssh:** secure system administration and file transfers over insecure networks.

- **vnStat:** network traffic monitor.

- **wget:** retrieving files using HTTP, HTTPS, FTP and FTPS.

SOCKET API

1) socket () Function:

The first step is to call the socket function, specifying the type of communication protocol (TCP based on IPv4, TCP based on IPv6, UDP).
The function is defined as follows:

```
#include <sys/socket.h>
int socket (int family, int type, int protocol);
```

where family specifies the protocol family (AF_INET for the IPv4 protocols), type is a constant described the type of socket (SOCK_STREAM for stream sockets (TCP Protocol and SOCK_DGRAM for datagram sockets (UDP Protocol)).

The function returns a non-negative integer number, similar to a file descriptor, that we define socket descriptor or -1 on error.

2) connect () Function:

The connect () function is used by a TCP client to establish a connection with a TCP server.

The function is defined as follows:

```
#include <sys/socket.h>
int connect (int sockfd, const struct sockaddr *servaddr, socklen_t
addrlen);
```

where sockfd is the socket descriptor returned by the socket function.

The function returns 0 if the it succeeds in establishing a connection (i.e., successful TCP three-way handshake, -1 otherwise.

The client does not have to call bind () in Section before calling this function: the kernel will choose both an ephemeral port and the source IP if necessary.

3) bind () Function:

The bind () assigns a local protocol address to a socket. With the Internet protocols, the address is the combination of an IPv4 or IPv6 address (32-bit or 128-bit) address along with a 16-bit TCP port number.
The function is defined as follows:

```
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

where sockfd is the socket descriptor, servaddr is a pointer to a protocol-specific address and addrlen is the size of the address structure. Bind () returns 0 if it succeeds, -1 on error.

A process can bind a specific IP address to its socket: for a TCP client, this assigns the source IP address that will be used for IP datagrams sent on the sockets. For a TCP server, this restricts the socket to receive incoming client connections destined only to that IP address.

Normally, a TCP client does not bind an IP address to its socket. The kernel chooses the source IP socket is connected, based on the outgoing interface that is used. If a TCP server does not bind an IP address to its socket, the kernel uses the destination IP address of the incoming packets as the server's source address.
bind () allows to specify the IP address, the port, both or neither.

4) listen () Function:

The listen () function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
It is defined as follows:

```
#include <sys/socket.h>
int listen (int sockfd, int backlog);
```

where sockfd is the socket descriptor and backlog is the maximum number of connections the kernel should queue for this socket. The backlog argument provides a hint to the system of the number of outstanding connect requests that it should enqueue on behalf of the process. Once the queue is full, the system will reject additional connection requests. The backlog value must be chosen based on the expected load of the server.
The function listen () return 0 if it succeeds, -1 on error.

5) accept () Function:

The accept () is used to retrieve a connect request and convert that into a request. It is defined as follows:

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

where sockfd is a new file descriptor that is connected to the client that called the connect (). The cliaddr and addrlen arguments are used to return the protocol address of the client. The new socket descriptor has the same socket type and address family of the original socket. The original socket passed to accept () is not associated with the connection, but instead remains available to receive additional connect requests. The kernel creates one connected socket for each client connection that is accepted.

If we don't care about the client's identity, we can set the cliaddr and addrlen to NULL. Otherwise, before calling the accept function, the cliaddr parameter has to be set to a buffer large enough to hold the address and set the integer pointed by addrlen to the size of the buffer.

6) send () Function:

Since a socket endpoint is represented as a file descriptor, we can use read and write to communicate with a socket as long as it is connected. However, if we want to specify options, we need another set of functions.
For example, send () is similar to write () but allows to specify some options. send () is defined as follows:

```
#include <sys/socket.h>
ssize_t send (int sockfd, const void *buf, size_t nbytes, int flags);
```

where buf and nbytes have the same meaning as they have with write. The additional argument flags is used to specify how we want the data to be transmitted. We will not consider the possible options in this course. We will assume it equal to 0.
The function returns the number of bytes if it succeeds, -1 on error.

7) recv () Function:

The recv () function is similar to read (), but allows to specify some options to control how the data are received. We will not consider the possible options in this course. We will assume it equal to 0.
receive is defined as follows:

```
#include <sys/socket.h>
ssize_t recv (int sockfd, void *buf, size_t nbytes, int flags);
```

The function returns the length of the message in bytes, 0 if no messages are available and peer had done an orderly shutdown, or -1 on error.

8) recvfrom () Function:

This function is similar to the read () function, but three additional arguments are required. The recvfrom () function is defined as follows:

```
#include <sys/socket.h>
```

ssize_t recvfrom (int sockfd, void* buff, size_t nbytes,int flags, struct sockaddr* from,socklen_t *addrlen);

The first three arguments sockfd, buff, and nbytes, are identical to the first three arguments of read and write. sockfd is the socket descriptor, buff is the pointer to read into, and nbytes is number of bytes to read. In our examples we will set all the values of the flag's argument to 0. The recvfrom function fills in the socket address structure pointed to by from with the protocol address of who sent the datagram. The number of bytes stored in the socket address structure is returned in the integer pointed by addrlen.
The function returns the number of bytes read if it succeeds, -1 on error.

9) sendto () Function:

This function is similar to the send () function, but three additional arguments are required. The sendto () function is defined as follows:

#include <sys/socket.h>
ssize_t sendto (int sockfd, const void *buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t addrlen);

The first three arguments sockfd, buff, and nbytes, are identical to the first three arguments of recv. sockfd is the socket descriptor, buff is the pointer to write from, and nbytes is number of bytes to write. In our examples we will set all the values of the flag's argument to 0. The to argument is a socket address structure containing the protocol address (e.g., IP address and port number) of where the data is sent. addlen specified the size of this socket.
The function returns the number of bytes written if it succeeds, -1 on error.

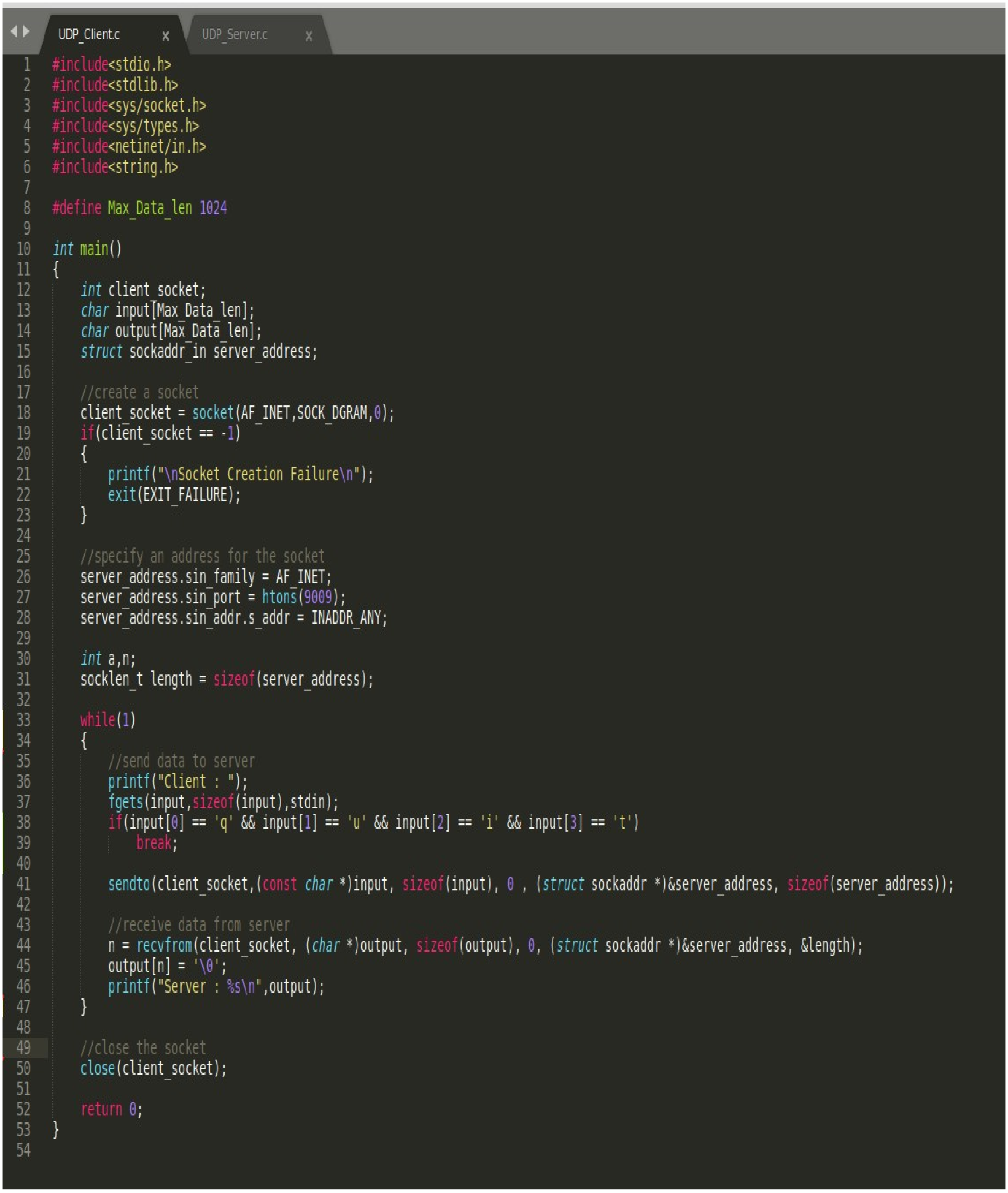
10) close () Function:

The normal close () function is used to close a socket and terminate a TCP socket. It returns 0 if it succeeds, -1 on error. It is defined as follows:

#include <unistd.h>
int close (int sockfd);

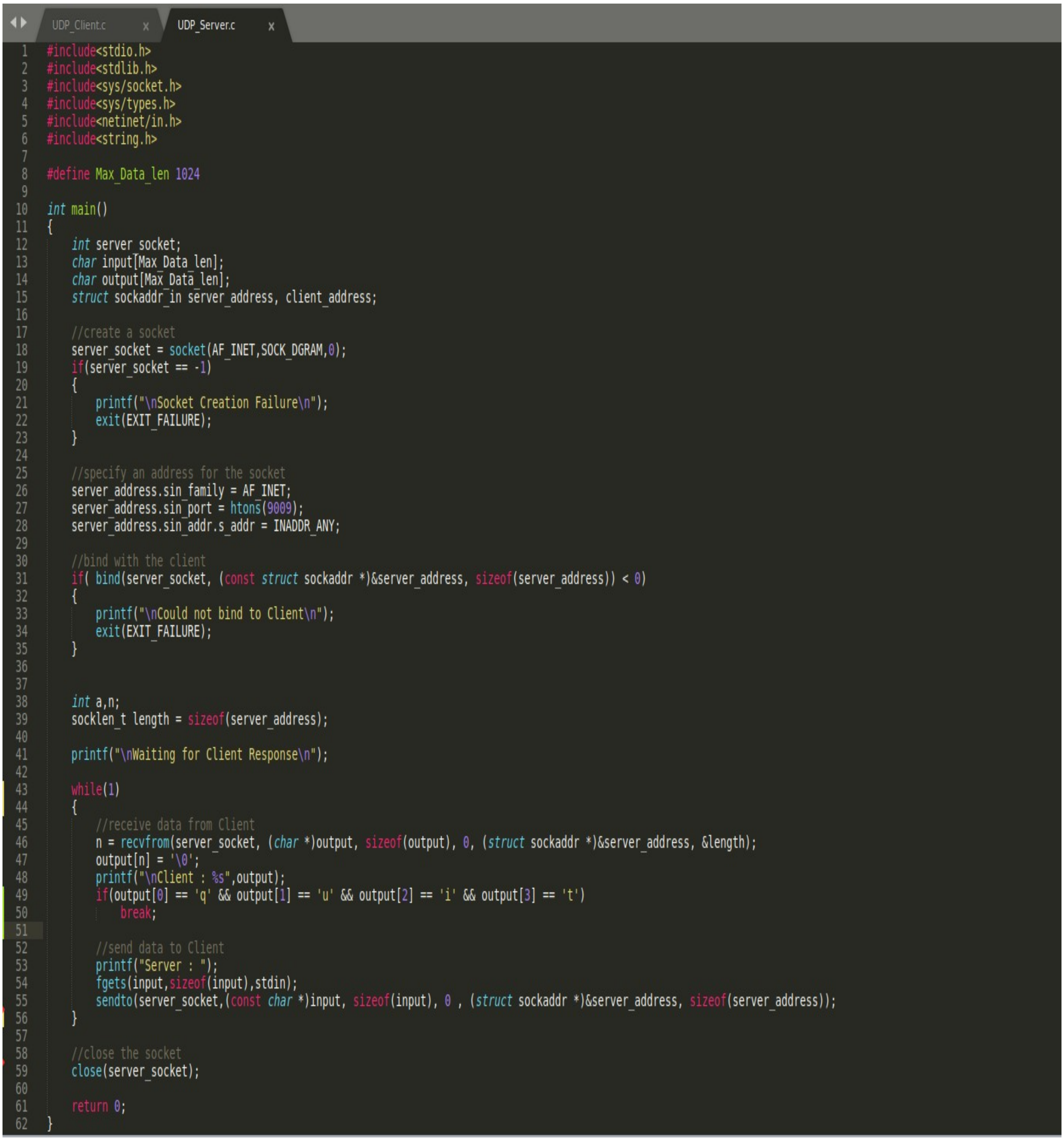
PROGRAM

File name: UDP_Client.c



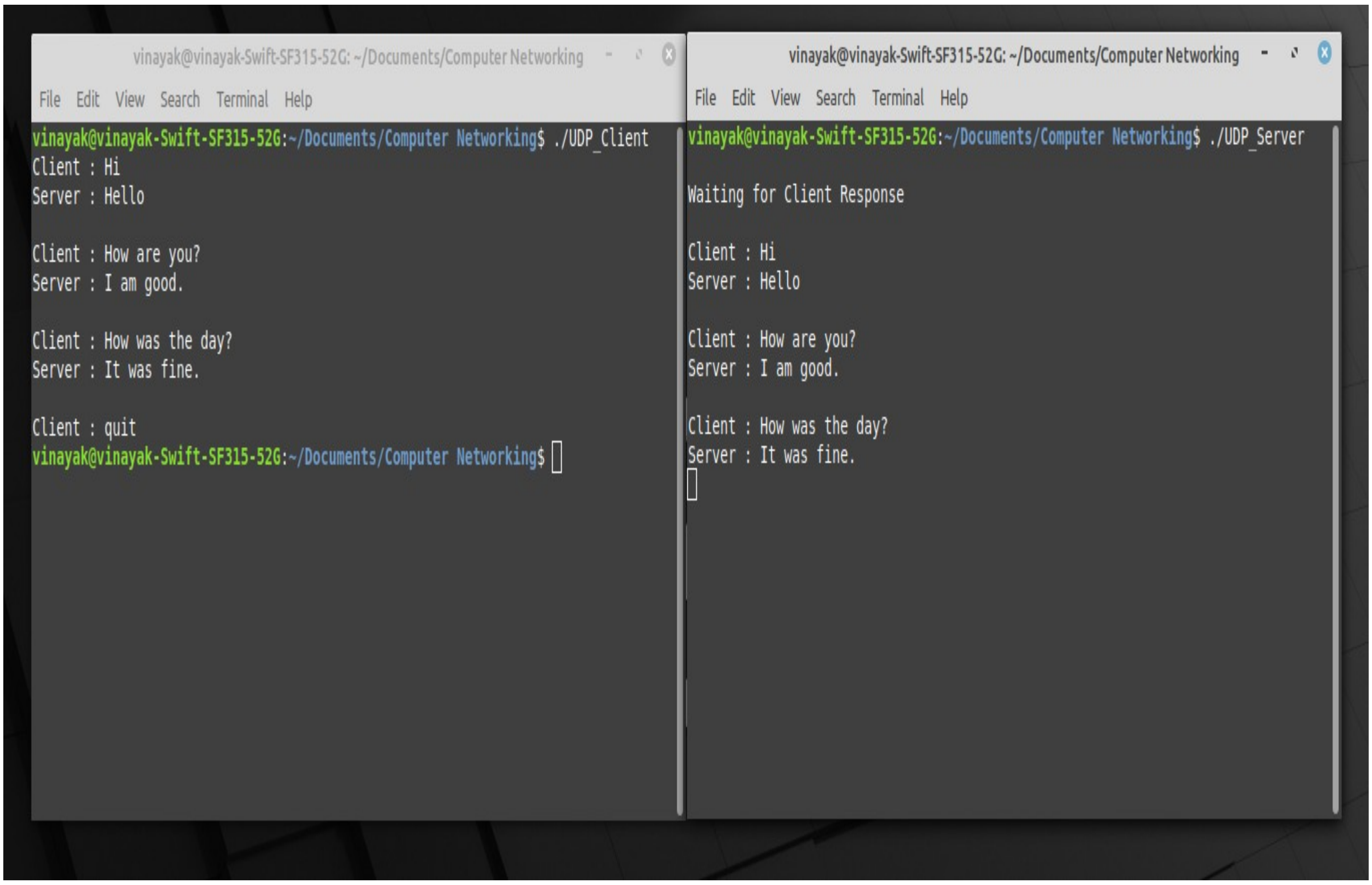
```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>

#define Max_Data_len 1024

int main()
{
    int client_socket;
    char input[Max_Data_len];
    char output[Max_Data_len];
    struct sockaddr_in server_address;

    //create a socket
    client_socket = socket(AF_INET,SOCK_DGRAM,0);
    if(client_socket == -1)
    {
        printf("\nSocket Creation Failure\n");
        exit(EXIT_FAILURE);
    }

    //specify an address for the socket
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(9009);
    server_address.sin_addr.s_addr = INADDR_ANY;

    int a,n;
    socklen_t length = sizeof(server_address);

    while(1)
    {
        //send data to server
        printf("Client : ");
        fgets(input,sizeof(input),stdin);
        if(input[0] == 'q' && input[1] == 'u' && input[2] == 'i' && input[3] == 't')
            break;

        sendto(client_socket,(const char *)input, sizeof(input), 0 , (struct sockaddr *)&server_address, sizeof(server_address));

        //receive data from server
        n = recvfrom(client_socket, (char *)output, sizeof(output), 0, (struct sockaddr *)&server_address, &length);
        output[n] = '\0';
        printf("Server : %s\n",output);
    }

    //close the socket
    close(client_socket);

    return 0;
}
```

File Name: UDP_Server.c



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

#define Max_Data_len 1024

int main()
{
    int server_socket;
    char input[Max_Data_len];
    char output[Max_Data_len];
    struct sockaddr_in server_address, client_address;

    //create a socket
    server_socket = socket(AF_INET,SOCK_DGRAM,0);
    if(server_socket == -1)
    {
        printf("\nSocket Creation Failure\n");
        exit(EXIT_FAILURE);
    }

    //specify an address for the socket
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(9009);
    server_address.sin_addr.s_addr = INADDR_ANY;

    //bind with the client
    if( bind(server_socket, (const struct sockaddr *)&server_address, sizeof(server_address)) < 0)
    {
        printf("\nCould not bind to Client\n");
        exit(EXIT_FAILURE);
    }


    int a,n;
    socklen_t length = sizeof(server_address);

    printf("\nWaiting for Client Response\n");

    while(1)
    {
        //receive data from Client
        n = recvfrom(server_socket, (char *)output, sizeof(output), 0, (struct sockaddr *)&server_address, &length);
        output[n] = '\0';
        printf("\nClient : %s",output);
        if(output[0] == 'q' && output[1] == 'u' && output[2] == 'i' && output[3] == 't')
            break;

        //send data to Client
        printf("Server : ");
        fgets(input,sizeof(input),stdin);
        sendto(server_socket,(const char *)input, sizeof(input), 0 , (struct sockaddr *)&server_address, sizeof(server_address));
    }

    //close the socket
    close(server_socket);

    return 0;
}
```

Output:


vinayak@vinayak-Swift-SF315-52G: ~/Documents/Computer Networking
File Edit View Search Terminal Help
vinayak@vinayak-Swift-SF315-52G: ~/Documents/Computer Networking
File Edit View Search Terminal Help

```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Computer Networking$ ./UDP_Client
Client : Hi
Server : Hello

Client : How are you?
Server : I am good.

Client : How was the day?
Server : It was fine.

Client : quit
vinayak@vinayak-Swift-SF315-52G:~/Documents/Computer Networking$
```

```
vinayak@vinayak-Swift-SF315-52G:~/Documents/Computer Networking$ ./UDP_Server

Waiting for Client Response

Client : Hi
Server : Hello

Client : How are you?
Server : I am good.

Client : How was the day?
Server : It was fine.
```