

EE533 Network Processor Design & Programming

Lab #1: Familiarity with VM & Sockets

Instructor: Prof. Young Cho, PhD

Name: Archit Sethi

University of Southern California

Los Angeles, CA 90007

GITHUB LINK to my REPOSITORY

Aarch0811/EE533/LAB1/

Archit's GITHUB: [Link](#)

1. Setting up two virtual machine nodes on VMware/VirtualBox

I've downloaded all the links from lab-1 pdf uploaded on USC Brightspace account. The first steps involved downloading Virtual Machine on our systems & then downloading a Linux-based OS (e.g. ubuntu).

I've used VirtualBox to work with Ubuntu & not VMware Workstation Pro. I was unable to establish my connection with internet, so I took the advise of TA Pavan to use VirtualBox.

Procedure, I followed for Part-1:

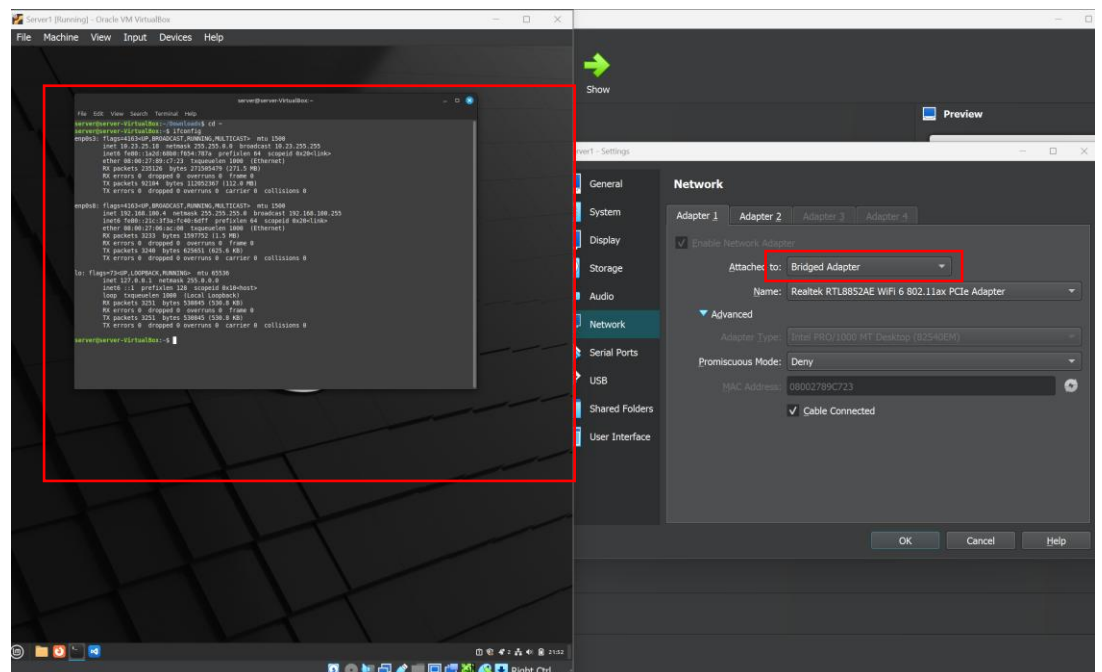
- Download & install VMware/VirtualBox on our systems.
- Create 2 Linux VMs using VMware/VirtualBox, naming them 'server' & 'client' respectively.
- Configure networking between the 2 VMs to allow communication between them.

Just, to explore a simpler windows experience in Linux-based Operating System (OS), I've used Linux Cinnamon Mint. It is lighter than Ubuntu and occupies less space.

Managed to get the complete resolution as per my screen.

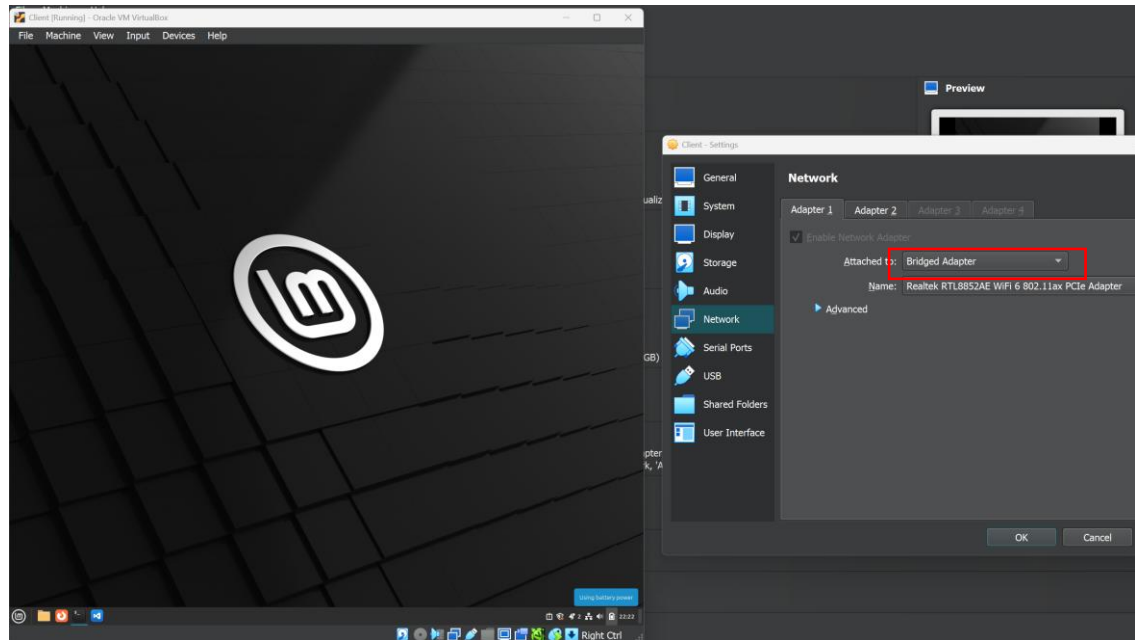
a. Server Virtual Machine

As you can see in the image below, I've added the *bridged adapter* ON, which will allow me to communicate with the client VM.

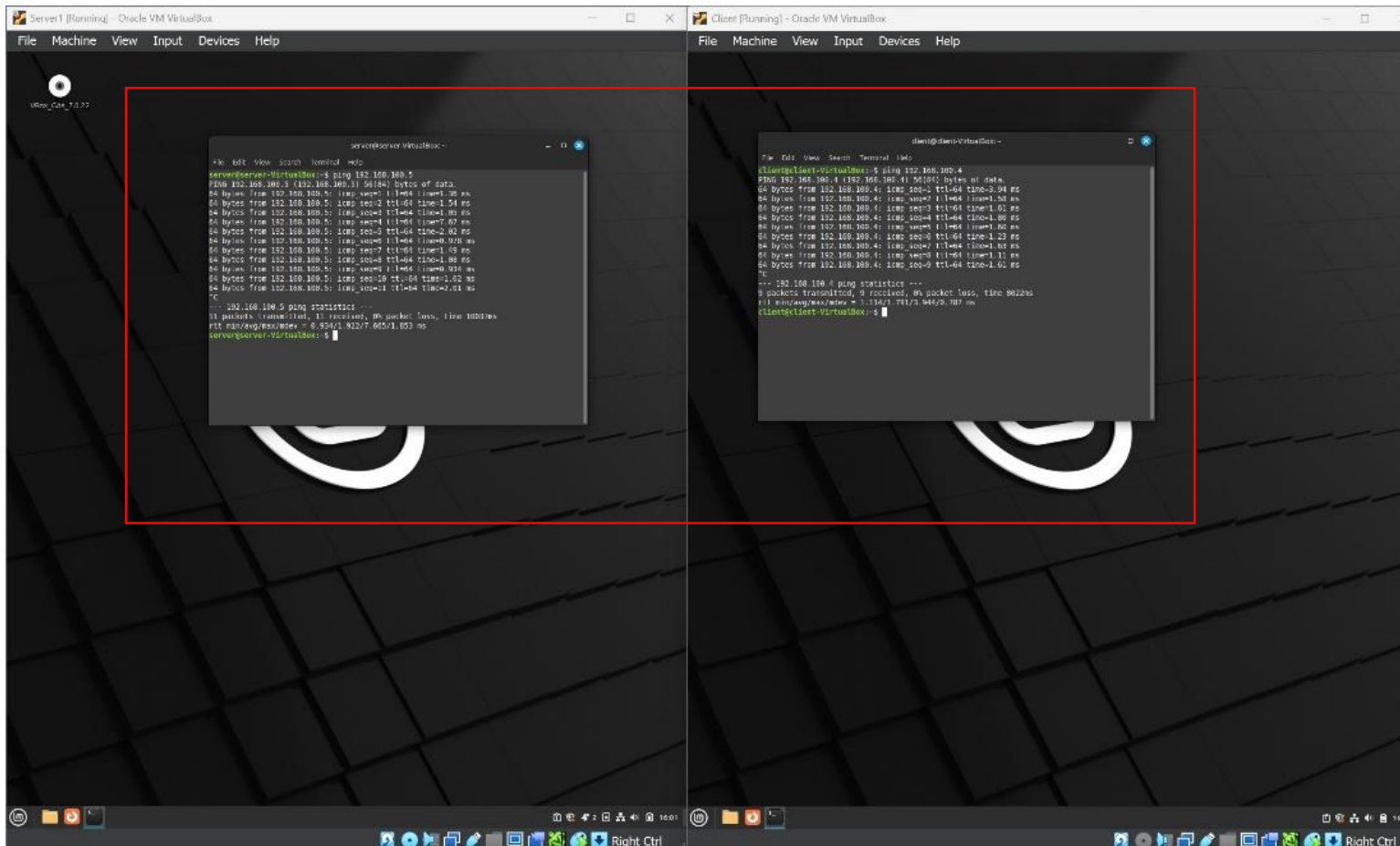


b. Client Virtual Machine

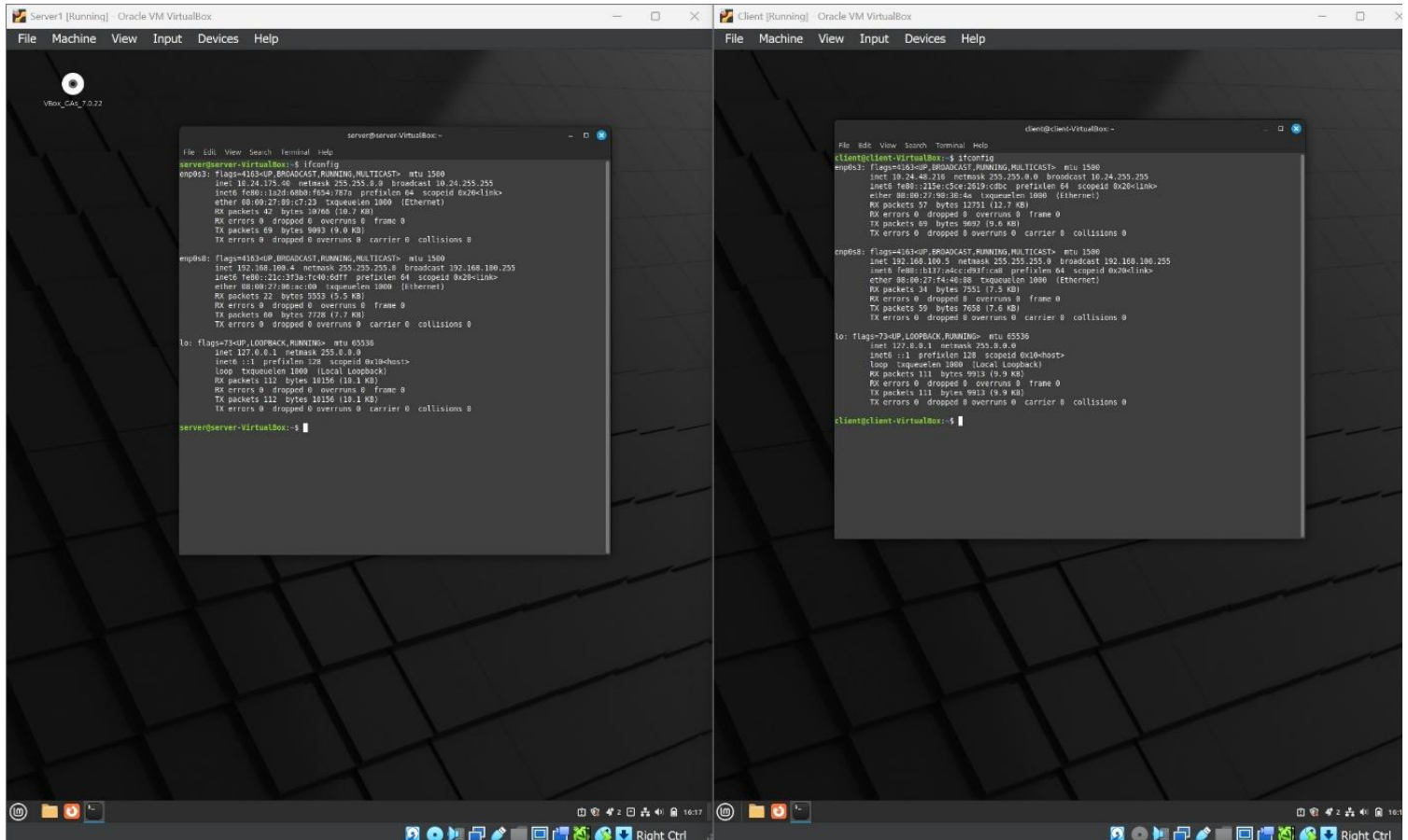
Similarly, with the client VM, I've added the *Network Adapter* to communicate its messages to the server VM.



Result, I've shown the pinned version of the 2 VMs (server & client) side-by-side with there ip addresses to display that both are working and communicating (pinging) perfectly.



Configuration settings for 2-way communication between 2 VMs: Kindly please look at the network modifications that I made in the settings of virtual machines so that there is ‘ping’ between the server & client.



2. Implement Socket Communication

a. Establish socket-based communication between the server and client.

They are follows:

NOTE: The client needs to know that the server exists and its address (server’s address), but the opposite doesn’t need to know the existence of client prior to the connection being established.

Procedure for socket creation for ‘server.c’:

- Create socket() with system call *socket()*
- Bind the socket to an address using *bind()* system call. The port number on the host machine is the address of the server socket.
- Listen for connections of a port number using *listen()* system call.
- Accept an incoming connection using *accept()* system call.
- Send and receive data using *read()* and *write()* calls

- b. Using the provided sample code on Brightspace ‘*server.c*’ & ‘*client.c*’ to implement a TCP-based communication system.**

To implement socket-based communication, we first must import the relevant libraries into our ‘*server.c*’ & ‘*client.c*’ codes.

Libraries added in the server & client codes for socket programming:

#include <stdio.h> → C programs uses input and output declarations

#include <sys/types.h> → defines no. of data types used in system calls.

#include <sys/socket.h> → includes no. of definitions of structures needed for sockets.

#include <netinet.h> → constraints and structures used for internet domain addresses.

- c. Compile the server and client programs into executables named ‘*server*’ & ‘*client*’**

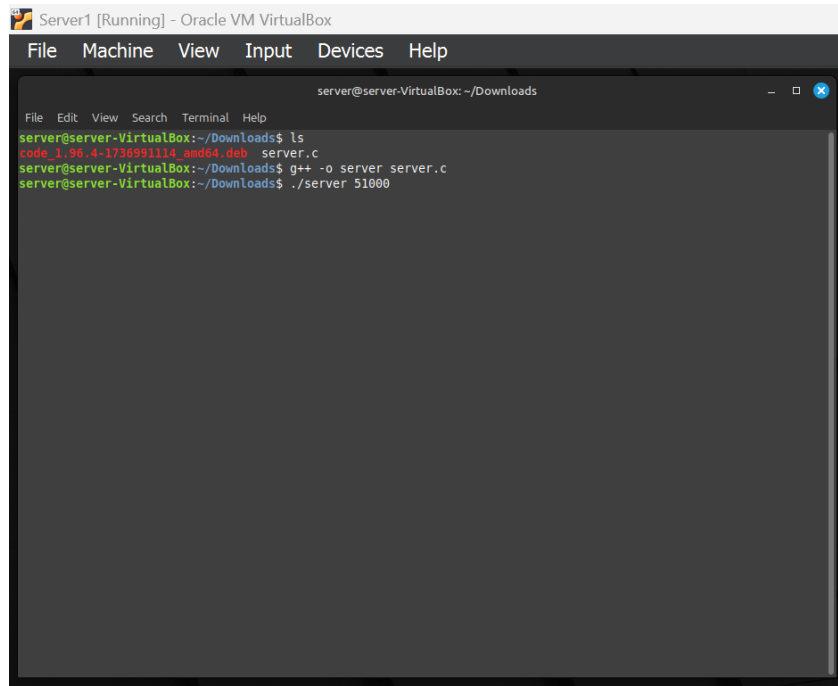
We compile our *server.c* & *client.c* using “g++ -o <executable_name> <file.c>” where file.c is my C source code (*server.c* & *client.c*) & ‘-o <executable_name>’ specifies the output name of the executable (*server* & *client*)

3. Testing the communication:

- a. Run the server program with specifies port number (like I’ve used server 51000)**

The server is specified with port number 51000. We can select any port number withing a certain range of addresses.

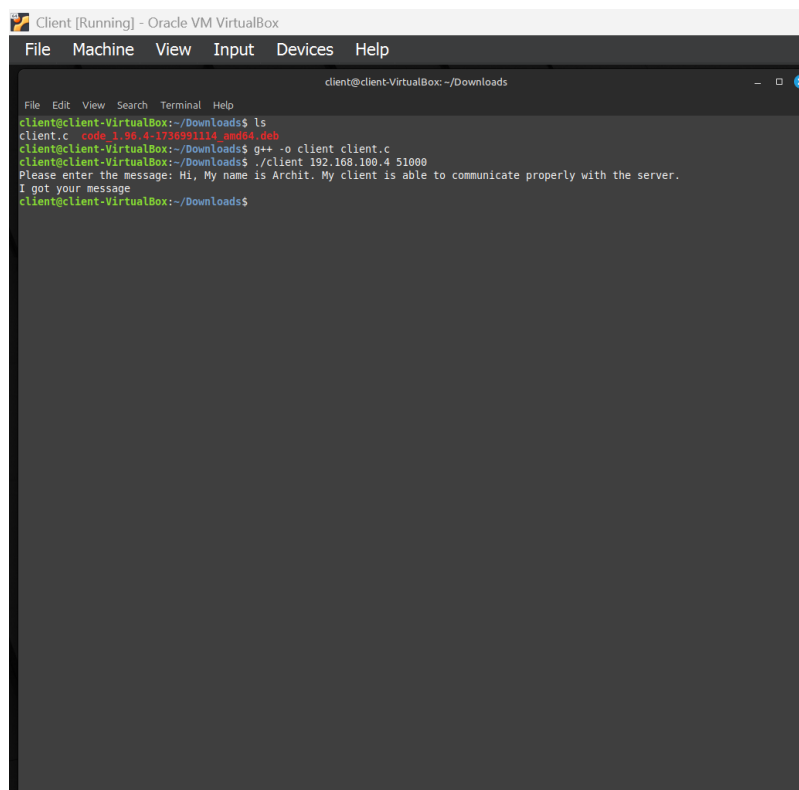
Right now the server is in listen state. Waiting a message from the client.



```
Server1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

server@server-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
server@server-VirtualBox:~/Downloads$ ls
code_1.96.4-1736991114_and64.deb server.c
server@server-VirtualBox:~/Downloads$ g++ -o server server.c
server@server-VirtualBox:~/Downloads$ ./server 51000
```

- b. **Run the client code, providing the server hostname & port number**
‘client’ executable was created and while run-time server’s IP address was specified as 192.168.100.4 with port number 51000



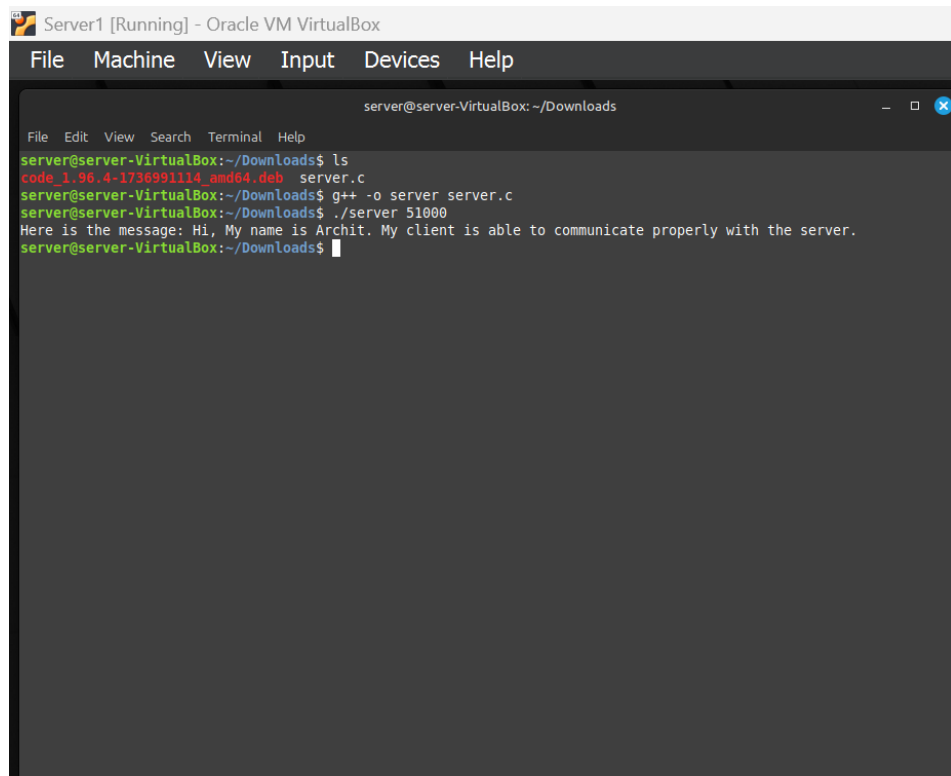
```
Client [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

client@client-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
client@client-VirtualBox:~/Downloads$ ls
code_1.96.4-1736991114_and64.deb
client.c
client@client-VirtualBox:~/Downloads$ g++ -o client client.c
client@client-VirtualBox:~/Downloads$ ./client 192.168.100.4 51000
Please enter the message: Hi, My name is Archit. My client is able to communicate properly with the server.
I got your message
client@client-VirtualBox:~/Downloads$
```

- c. **Verify that the server displays the message sent by client and sends an acknowledgement back to the client.**

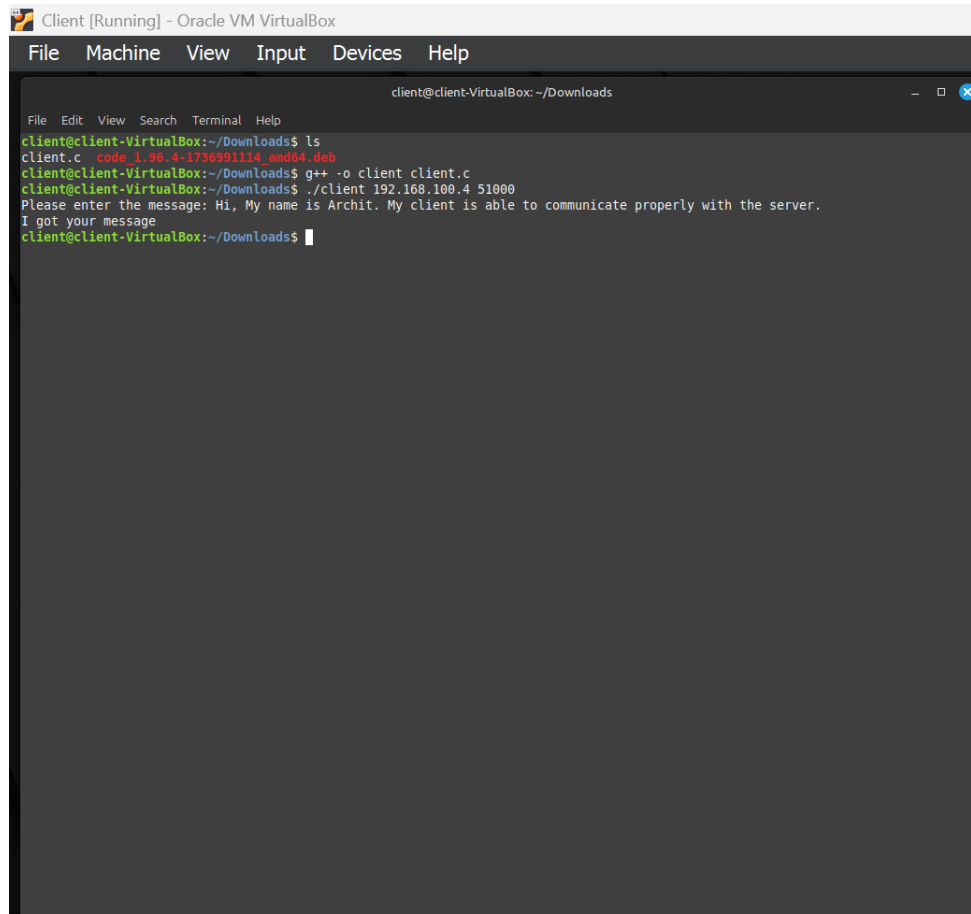
Server's response:

The server displays the message sent by client. Here I had written in the message, 'Hi, My name is Archit. My client is able to communicate properly with the server. This message is being displayed in the output window shown below.



```
Server1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
server@server-VirtualBox: ~/Downloads
server@server-VirtualBox:~/Downloads$ ls
code 1.96.4-1736991114 amd64.deb server.c
server@server-VirtualBox:~/Downloads$ g++ -o server server.c
server@server-VirtualBox:~/Downloads$ ./server 51000
Here is the message: Hi, My name is Archit. My client is able to communicate properly with the server.
server@server-VirtualBox:~/Downloads$
```

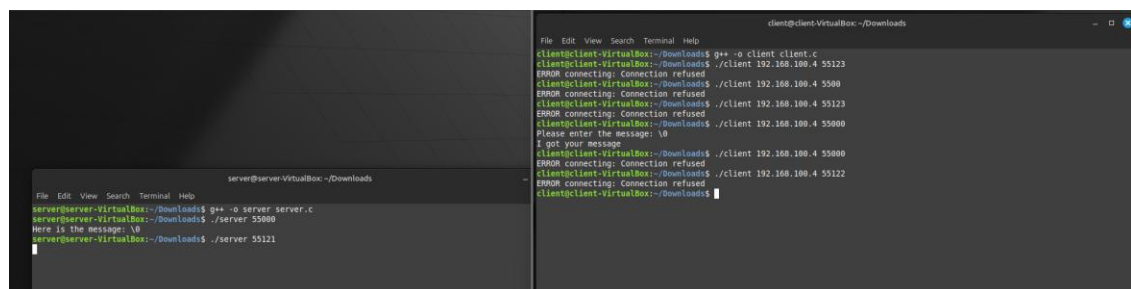
Client's response: Client acknowledges send by the server. It displays 'I got your message'.



```
Client [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
client@client-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
client@client-VirtualBox:~/Downloads$ ls
client.c  code_1.96.4-1736991114_amd64.deb
client@client-VirtualBox:~/Downloads$ g++ -o client client.c
client@client-VirtualBox:~/Downloads$ ./client 192.168.100.4 51000
Please enter the message: Hi, My name is Archit. My client is able to communicate properly with the server.
I got your message
client@client-VirtualBox:~/Downloads$
```

Let us play a little with the codes given to me. I'll try to connect the client to a port number which doesn't exist (e.g. 55122). The host port number is 55121.

It throws an exception by displaying the message as "ERROR connecting: Connection refused"



```
server@server-VirtualBox:~/Downloads
File Edit View Search Terminal Help
server@server-VirtualBox:~/Downloads$ g++ -o server server.c
server@server-VirtualBox:~/Downloads$ ./server 55000
Here is the message: \0
server@server-VirtualBox:~/Downloads$ ./server 55121

client@client-VirtualBox:~/Downloads
File Edit View Search Terminal Help
client@client-VirtualBox:~/Downloads$ g++ -o client client.c
client@client-VirtualBox:~/Downloads$ ./client 192.168.100.4 55122
ERROR connecting: Connection refused
client@client-VirtualBox:~/Downloads$ ./client 192.168.100.4 55000
ERROR connecting: Connection refused
client@client-VirtualBox:~/Downloads$ ./client 192.168.100.4 55122
ERROR connecting: Connection refused
client@client-VirtualBox:~/Downloads$
```

4. Sample Code

- a. Source codes provided to us on USC Brightspace:
 - i. Server.c changes made by me


```
server.c
1  /* A simple server in the internet domain using TCP
2     The port number is passed as an argument */
3  #include <stdio.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7
8  void error(char *msg)
9  {
10     perror(msg);
11     exit(1);
12 }
13
14 int main(int argc, char *argv[])
15 {
16     int sockfd, newsockfd, portno, clien;
17     char buffer[256];
18     struct sockaddr_in serv_addr, cli_addr;
19     int n;
20     if (argc < 2) {
21         fprintf(stderr, "ERROR, no port provided\n");
22         exit(1);
23     }
24     sockfd = socket(AF_INET, SOCK_STREAM, 0);
25     if (sockfd < 0)
26         error("ERROR opening socket");
27     bzero((char *) &serv_addr, sizeof(serv_addr));
28     portno = atoi(argv[1]);
29     serv_addr.sin_family = AF_INET;
30     serv_addr.sin_addr.s_addr = INADDR_ANY;
31     serv_addr.sin_port = htons(portno);
32     if (bind(sockfd, (struct sockaddr *) &serv_addr,
33             sizeof(serv_addr)) < 0)
34         error("ERROR on binding");
35     listen(sockfd,5);
36     clien = sizeof(cli_addr);
37     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clien);
38     if (newsockfd < 0)
39         error("ERROR on accept");
40     bzero(buffer,256);
41     n = read(newsockfd,buffer,255);
42     if (n < 0) error("ERROR reading from socket");
43     printf("Here is the message: %s\n",buffer);
44     n = write(newsockfd,"I got your message",18);
45     if (n < 0) error("ERROR writing to socket");
46     return 0;
47 }
```

```
server.c
1  /* A simple server in the internet domain using TCP
2     The port number is passed as an argument */
3  #include <stdio.h>
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7
8  void error(char *msg)
9  {
10     perror(msg);
11     exit(1);
12 }
13
14 int main(int argc, char *argv[])
15 {
16     int sockfd, newsockfd, portno, clien;
17     char buffer[256];
18     struct sockaddr_in serv_addr, cli_addr;
19     int n;
20     if (argc < 2) {
21         fprintf(stderr, "ERROR, no port provided\n");
22         exit(1);
23     }
24     sockfd = socket(AF_INET, SOCK_STREAM, 0);
25     if (sockfd < 0)
26         error("ERROR opening socket");
27     bzero((char *) &serv_addr, sizeof(serv_addr));
28     portno = atoi(argv[1]);
29     serv_addr.sin_family = AF_INET;
30     serv_addr.sin_addr.s_addr = INADDR_ANY;
31     serv_addr.sin_port = htons(portno);
32     if (bind(sockfd, (struct sockaddr *) &serv_addr,
33             sizeof(serv_addr)) < 0)
34         error("ERROR on binding");
35     listen(sockfd,5);
36     clien = sizeof(cli_addr);
37     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clien);
38     if (newsockfd < 0)
39         error("ERROR on accept");
40     bzero(buffer,256);
41     n = read(newsockfd,buffer,255);
42     if (n < 0) error("ERROR reading from socket");
43     printf("Here is the message: %s\n",buffer);
44     n = write(newsockfd,"I got your message",18);
45     if (n < 0) error("ERROR writing to socket");
46     return 0;
47 }
```

As you can see clearly the codes provided to us on Brightspace were full of bugs. There were many errors which I had resolved. These were library issues, variable declarations, wrong parameters used, etc.,

The updated and bug-free source code has been depicted in the snippet below. It was compiled properly and was able to generate the executable file.

ii. Client.c changes made by me

As you can see clearly the codes provided to us on Brightspace were full of bugs. There were many errors which I had resolved. These were library issues, variable declarations, wrong parameters used, etc.,

The updated and bug-free source code has been depicted in the snippet below. It was compiled properly and was able to generate the executable file.

```

client_original.c > main(int, char *[])
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <netinet/in.h>
5 #include <netdb.h>
6
7 void error(char *msg)
8 {
9     perror(msg);
10    exit(0);
11 }
12
13 int main(int argc, char *argv[])
14 {
15     int sockfd, portno, n;
16
17     struct sockaddr_in serv_addr;
18     struct hostent *server;
19
20     char buffer[256];
21     if (argc < 3) {
22         fprintf(stderr, "usage %s hostname port\n", argv[0]);
23         exit(0);
24     }
25     portno = atoi(argv[2]);
26     sockfd = socket(AF_INET, SOCK_STREAM, 0);
27     if (sockfd < 0)
28         error("ERROR opening socket");
29     server = gethostbyname(argv[1]);
30     if (server == NULL) {
31         fprintf(stderr, "ERROR, no such host\n");
32         exit(0);
33     }
34     bzero((char *) &serv_addr, sizeof(serv_addr));
35     serv_addr.sin_family = AF_INET;
36     bcopy((char *)server->h_addr,
37           (char *)&serv_addr.sin_addr.s_addr,
38           server->h_length);
39     serv_addr.sin_port = htons(portno);
40     if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
41         error("ERROR connecting");
42     printf("Please enter the message: ");
43     bzero(buffer, 256);
44     fgets(buffer, 255, stdin);
45     n = write(sockfd, buffer, strlen(buffer));
46     if (n < 0)
47         error("ERROR writing to socket");
48     bzero(buffer, 256);
49     n = read(sockfd, buffer, 255);
50     if (n < 0)
51         error("ERROR reading from socket");
52     printf("%s\n", buffer);
53     return 0;
54 }

```

```

client@client-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
client@client-VirtualBox:~/Downloads$ g++ -o client_original client_original.c
client_original.c: In function 'void error(char*)':
client_original.c:10:5: error: 'exit' was not declared in this scope
10     exit(0);
    ^~~~~
client_original.c:6:1: note: 'exit' is defined in header '<cstdlib>'; did you forget to '#include <cstdlib>'?
5     #include <netdb.h>
+++  #include <cstdlib>
6
client_original.c: In function 'int main(int, char*)':
client_original.c:23:8: error: 'exit' was not declared in this scope
23     exit(0);
    ^~~~~
client_original.c:23:8: note: 'exit' is defined in header '<cstdlib>'; did you forget to '#include <cstdlib>'?
25     portno = atoi(argv[2]);
    ^~~~~
client_original.c:28:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
28     error("ERROR opening socket");
    ^~~~~~
client_original.c:32:9: error: 'exit' was not declared in this scope
32     exit(0);
    ^~~~~
client_original.c:32:9: note: 'exit' is defined in header '<cstdlib>'; did you forget to '#include <cstdlib>'?
client_original.c:34:5: error: 'bzero' was not declared in this scope
34     bzero((char *) &serv_addr, sizeof(serv_addr));
    ^~~~~~
client_original.c:36:5: error: 'bcopy' was not declared in this scope
36     bcopy((char *)server->h_addr,
    ^~~~~~
client_original.c:41:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
41     error("ERROR connecting");
    ^~~~~~
client_original.c:45:29: error: 'strlen' was not declared in this scope
45     n = write(sockfd, buffer, strlen(buffer));
    ^~~~~~
client_original.c:6:1: note: 'strlen' is defined in header '<string>'; did you forget to '#include <string>'?
5     #include <netdb.h>
+++  #include <string>
6
client_original.c:45:9: error: 'write' was not declared in this scope; did you mean 'fwrite'?
45     n = write(sockfd, buffer, strlen(buffer));
    ^~~~~~
client_original.c:47:16: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
47     error("ERROR writing to socket");
    ^~~~~~
client_original.c:49:9: error: 'read' was not declared in this scope; did you mean 'fread'?
49     n = read(sockfd, buffer, 255);
    ^~~~~~
client_original.c:51:16: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
51     error("ERROR reading from socket");
    ^~~~~~
client@client-VirtualBox:~/Downloads$

```

b. The SERVER code

```

5. /* A simple server in the internet domain using TCP */
6. #include <stdlib.h>
7. #include <string.h>
8. #include <unistd.h>
9. #include <netdb.h>
10. #include <stdio.h>
11. #include <sys/types.h>
12. #include <sys/socket.h>
13. #include <netinet/in.h>
14.
15. void error(const char *msg)
16. {
17.     perror(msg);
18.     exit(1);
19. }
20.
21. int main(int argc, char *argv[])
22. {
23.     int sockfd, newsockfd, portno;
24.     socklen_t clien;
25.     char buffer[256];
26.     struct sockaddr_in serv_addr, cli_addr;

```

```
27.     int n;
28.
29.     if (argc < 2) {
30.         fprintf(stderr, "ERROR, no port provided\n");
31.         exit(1);
32.     }
33.
34.     sockfd = socket(AF_INET, SOCK_STREAM, 0);
35.     if (sockfd < 0)
36.         error("ERROR opening socket");
37.
38.     bzero((char *)&serv_addr, sizeof(serv_addr));
39.     portno = atoi(argv[1]);
40.     serv_addr.sin_family = AF_INET;
41.     serv_addr.sin_addr.s_addr = INADDR_ANY;
42.     serv_addr.sin_port = htons(portno);
43.
44.     if (bind(sockfd, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0)
45.         error("ERROR on binding");
46.
47.     listen(sockfd, 5);
48.     clilen = sizeof(cli_addr);
49.     newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr,
&clilen);
50.     if (newsockfd < 0)
51.         error("ERROR on accept");
52.
53.     // Corrected code block
54.     bzero(buffer, 256);
55.     n = read(newsockfd, buffer, 255);
56.     if (n <= 0) {
57.         error("ERROR reading from socket");
58.     }
59.     buffer[n] = '\0'; // Null-terminate the received string
60.     printf("Here is the message: %s\n", buffer);
61.
62.     n = write(newsockfd, "I got your message", 18);
63.     if (n < 0) {
64.         error("ERROR writing to socket");
65.     }
66.
67.     // Cleanup
68.     close(newsockfd);
69.     close(sockfd);
70.     return 0;
71. }
72.
```

a. The CLIENT code

```
73.#include <stdlib.h>
74.#include <string.h>
75.#include <unistd.h>
76.#include <netdb.h>
77.#include <stdio.h>
78.#include <sys/types.h>
79.#include <sys/socket.h>
80.#include <netinet/in.h>
81.#include <netdb.h>
82.
83.void error(const char *msg)
84.{
85.    perror(msg);
86.    exit(0);
87.}
88.
89.int main(int argc, char *argv[])
90.{
91.    int sockfd, portno, n;
92.
93.    struct sockaddr_in serv_addr;
94.    struct hostent *server;
95.
96.    char buffer[256];
97.    if (argc < 3) {
98.        fprintf(stderr,"usage %s hostname port\n", argv[0]);
99.        exit(0);
100.    }
101.    portno = atoi(argv[2]);
102.    sockfd = socket(AF_INET, SOCK_STREAM, 0);
103.    if (sockfd < 0)
104.        error("ERROR opening socket");
105.    server = gethostbyname(argv[1]);
106.    if (server == NULL) {
107.        fprintf(stderr,"ERROR, no such host\n");
108.        exit(0);
109.    }
110.    bzero((char *) &serv_addr, sizeof(serv_addr));
111.    serv_addr.sin_family = AF_INET;
112.    bcopy((char *)server->h_addr,
113.        (char *)&serv_addr.sin_addr.s_addr,
114.        server->h_length);
115.    serv_addr.sin_port = htons(portno);
116.    if (connect(sockfd,(struct sockaddr
117.        *)&serv_addr,sizeof(serv_addr)) < 0)
118.        error("ERROR connecting");
```

```

119.
120.     printf("Please enter the message: ");
121.     bzero(buffer, 256);
122.     fgets(buffer, 255, stdin);
123.     buffer[strcspn(buffer, "\n")] = '\0'; // Remove trailing
        newline
124.
125.     n = write(sockfd, buffer, strlen(buffer));
126.     if (n <= 0) {
127.         error("ERROR writing to socket");
128.     }
129.
130.     bzero(buffer, 256);
131.     n = read(sockfd, buffer, 255);
132.     if (n <= 0) {
133.         error("ERROR reading from socket");
134.     }
135.     buffer[n] = '\0'; // Null-terminate the received string
136.     printf("%s\n", buffer);
137.
138.     // Close the socket
139.     close(sockfd);
140. }

```

Explanation for Server code 'server.c':

- a. Import relevant libraries essential for socket programming, input/output operations, and error handling.

```

#include <stdlib.h>
#include <string.h>           // For bzero()
#include <unistd.h>           // For close() and read()/write()
#include <netdb.h>            // For networking-related structures
#include <stdio.h>            // For printf() and perror()
#include <sys/types.h>        // For data types like socklen_t
#include <sys/socket.h>       // For socket programming functions
#include <netinet/in.h>      // For sockaddr_in structure

```

- b. There is an error handling function that whenever a system call fails, making debugging easier.

```

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

```

- c. Main function: The program expects the port number as a Command-Line Argument. If NO port is provided, it prints an error message and exits.
- d. Socket Creation: Server creates a socket, binds it to the port, and listens for incoming connections.

- i. `socket(AF_INET, SOCK_STREAM, 0):`
`AF_INET`: Specifies IPv4.
`SOCK_STREAM`: Specifies TCP (stream-based protocol).
The return value is the file descriptor for the created socket.
If `socket()` fails, it calls `error()` to terminate the program.
- e. Server Address Initialization:
 - i. `bzero`: Clears the `serv_addr` structure.
 - ii. `portno`: Converts the port argument (string) to an integer.
 - iii. `sin_family`: Specifies IPv4.
 - iv. `sin_addr.s_addr`: Specifies the server's IP address. `INADDR_ANY` allows connections from any network interface.
 - v. `htons`: Converts the port number to big-endian byte number.
- f. Binding the socket:
 - i. `bind`: Associates the socket (`sockfd`) with the server's address (`serv_addr`) and port.
 - ii. If `bind()` fails, then it prints an error and exits.
- g. Listen for connections: `listen(sockfd, 5)`
 - i. `listen`: Marks the socket as passive i.e. it will accept incoming connections.
 - ii. The 2nd parameter specifies the maximum number of pending connections.
- h. Accept a connection: When a client connects, the server accepts the connection, creating a new socket.
 - i. `Accept`: Waits for an incoming client connection.
 - ii. `cli_addr`: Stores the address of the client
 - iii. `newsockfd`: A new file descriptor for the connection with the client/
 - iv. If no connection can be accepted, it prints an error and exits.
- i. Send data to client:

```
n = write(newsockfd, "I got your message", 18);
if (n < 0) {
    error("ERROR writing to socket");
}
```

 - `write`: Sends a message back to client.
 - `n`: stores the number of bytes
- j. Close the sockets: Properly close the client and server to release resources.
 - i. `close(newsockfd)`
 - ii. `close(sockfd)`

Why we use this server program?

The `server.c` program is simple TCP server that listens for incoming connections, exchanges messages with client and terminates after handling the communication. It serves as a base for understanding how servers operate in networked environment.

Explanation for Client code 'client.c':

- a. Resolve Server Address
gethostname(argv[1]): resolves the hostname to an IP address.
Returns a struct hostent containing the server's IP information. If host is invalid, the program exits with an error.
- b. connect: Establishes a connection to server using the initialized serv_addr.
If connection fails, the program exits with error.
- c. Send a message: prompts the user to input a message. fgets: Reads up to 255 characters from stdin into buffer. Strcspn: removes the trailing newline added by fgets. write: sends the message to the server through the socket.

Why we use this client program?

For testing the server, establish TCP communication ensuring reliable, ordered, and error-checked delivery, reusable design i.e. the program can connect to any server by changing the hostname and port, making it different testing scenarios, error handling, learning and debugging. The client is critical for validating and interacting with the server in socket programming projects.

Enhanced code for handling multiple clients and doesn't die after the process is terminated:

We will use fork() off a new process to handle each new connection.

Following changes were made in the server code to make the required enhancements:

- k. Put the accept statement and the following code in an infinite loop.
- l. After a connection is established, call fork()#### to create a new process.
- m. The child process will close sockfd#### and call #dostuff#####, passing the new socket file descriptor as an argument. When the two processes have completed their conversation, as indicated by dostuff()#### returning, this process simply exits.
- n. The parent process closes newsockfd####. Because all of this code is in an infinite loop, it will return to the accept statement to wait for the next connection

The code which has been provided to us is:

```
while (1){
newsockfd = accept(sockfd,
(struct sockaddr *) &cli_addr, &clilen);
if (newsockfd < 0)
error("ERROR on accept");
pid = fork();
if (pid < 0)
```

```

error("ERROR on fork");
if (pid == 0){
close(sockfd);
dostuff(newsockfd);
exit(0);}
else
close(newsockfd);
} /* end of while */

```

Below is the snippet showing that the code is working fine for the forking methodology.

The image shows two side-by-side terminal windows from Oracle VM VirtualBox. The left window is titled "Server1 [Running] - Oracle VM VirtualBox" and shows a terminal session on a server. The right window is titled "Client [Running] - Oracle VM VirtualBox" and shows a terminal session on a client. Both windows show the execution of a program that demonstrates forking methodology, with the server listening on port 52654 and the client connecting to it.

```

server@server-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
server@server-VirtualBox:~/Downloads$ ls
code 1.96.4-173891113 amd64.deb  server  server.c  server_enhanced.c  server_zombie.c
server@server-VirtualBox:~/Downloads$ g++ -o server_enhanced server_enhanced.c
server@server-VirtualBox:~/Downloads$ ls
code 1.96.4-173891113 amd64.deb  server  server.c  server_enhanced.c
server@server-VirtualBox:~/Downloads$ ./server_enhanced
server
server@server-VirtualBox:~/Downloads$ ./server_enhanced 52654
Server has been booted up. Server is listening on Port 52654...
Here is the message: Hi, this is the enhanced version which handles multiple clients using the
forking methodology.
Here is the message: The server is waiting for new process. This is the new client.

client@client-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
client@client-VirtualBox:~/Downloads$ ls
client.c  code 1.96.4-173891113 amd64.deb
client@client-VirtualBox:~/Downloads$ g++ -o client client.c
client@client-VirtualBox:~/Downloads$ ./client 192.168.106.4 52654
usage: ./client hostname port
client@client-VirtualBox:~/Downloads$ ./client 192.168.106.4 52654
Please enter the message: Hi, this is the enhanced version which handles multiple clients using the forking methodology.
I got your message
client@client-VirtualBox:~/Downloads$ ./client 192.168.106.4 52654
Please enter the message: The server is waiting for new process. This is the new client.
I got your message
client@client-VirtualBox:~/Downloads$

```

Zombie problem: The above output has a zombie problem. Zombie problem means that if the parent runs for a long time and accepts many connections, each of these connections will create a zombie when the connection is terminated. A zombie is a process which has terminated but cannot be permitted to fully die because at some point in the future, the parent of the process might execute a wait and would want information about the death of the child. Zombies clog up the process table in the kernel, and so they should be prevented.

Unfortunately, the code which prevents zombies is not consistent across different architectures. When a child dies, it sends a SIGCHLD signal to its parent. On systems such as AIX, the following code in main() is all that is needed.


```
signal(SIGCHLD,SIG_IGN);
```

This says to ignore the SIGCHLD signal. However, on systems running SunOS, you have to use the following code:

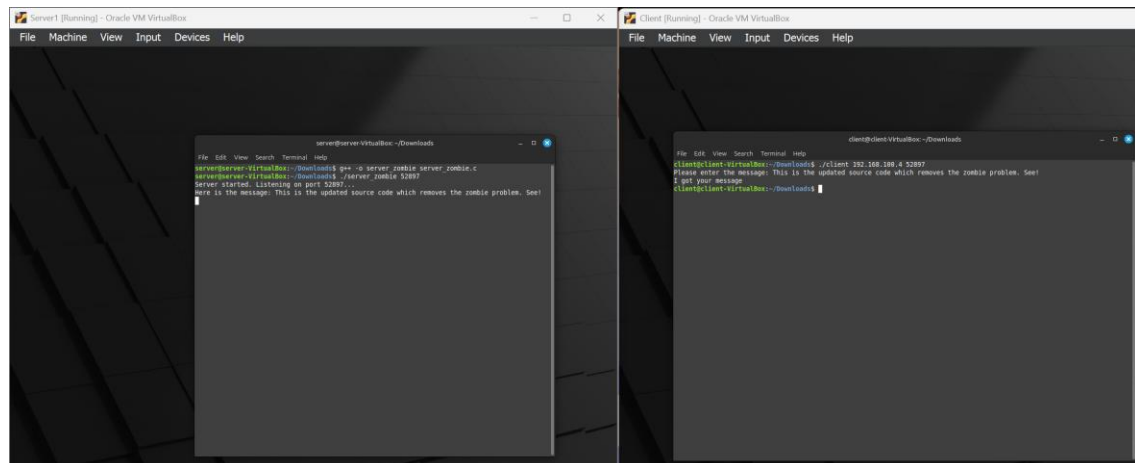
```
void *SigCatcher(int n)
{
wait3(NULL,WNOHANG,NULL);
}

...
int main()
{
...
signal(SIGCHLD,SigCatcher);
...

```

The function SigCatcher() will be called whenever the parent receives a SIGCHLD signal (i.e. whenever a child dies). This will in turn call wait3 which will receive the signal.

In the below-mentioned snippet, you can see that the server is listening on port number 52897. The client's message is displayed accordingly. Then the server is again goes in the accept state and listens for another client's message. As you can see the server has not terminated the process.



5. User-Datagram Protocol Sockets

UDP sockets can be compiled and run in exactly the same way as the server and client using the stream socket.

The major differences are depicted below in the form of code snippets:

Changes in the server.c code to form server_udp.c:

- 'clilen' is replaced with 'formlen' & cli_addr is replaced with 'from_addr'.

```
int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno;
    socklen_t fromlen; //clilen is replaced with fromlen
    char buffer[256];
    struct sockaddr_in serv_addr, from_addr; //cli_addr is replaced
    int n;
```

b. The following parts of server.c socket have been removed to make it usable for UDP protocol.

- a. accept(), connect(), listen(), read(), write() have been removed to make it usable for UDP.
- b. bind() is NOT removed.

```
// listen(sockfd, 5);
// clilen = sizeof(cli_addr);
// newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);
// if (newsockfd < 0)
//     error("ERROR on accept");

// bzero(buffer, 256);
// n = read(newsockfd, buffer, 255);
// if (n <= 0) {
//     error("ERROR reading from socket");
// }

buffer[n] = '\0'; // Null-terminate the received string
printf("Here is the message: %s\n", buffer);

// n = write(newsockfd, "I got your message", 18);
// if (n < 0) {
//     error("ERROR writing to socket");
// }

close(newsockfd);
close(sockfd);
return 0;
```

c. SOCK_STREAM was replaced with SOCK_DGRAM

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");

bzero((char *)&serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
```

Changes in the client.c code to form client_udp.c:

- a. The same changes were made in the client.c

```
int main(int argc, char *argv[])
{
    portno = atoi(argv[2]); //Converts the port number from STRING TO INTEGER

    sockfd = socket(AF_INET, SOCK_DGRAM, 0); //Creating a UDP SOCKET
    if (sockfd < 0)
        error("ERROR opening socket");

    server = gethostbyname(argv[1]); // Resolve the server's hostname
    if (server == NULL) {
        fprintf(stderr, "ERROR, no such host\n");
        exit(0);
    }

    bzero((char *)&serv_addr, sizeof(serv_addr)); // Set up the server address st

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

    //Prompt the user to enter a message
    printf("Please enter the message: ");
    bzero(buffer, 256);
    fgets(buffer, 255, stdin);
    buffer[strcspn(buffer, "\n")] = '\0'; //Remove trailing newline

    // Send the message to the server
    n = sendto(sockfd, buffer, strlen(buffer), 0,
        (struct sockaddr *)&serv_addr, sizeof(serv_addr));
    if (n < 0) {
        error("ERROR sending to socket");
    }

    socklen_t serv_len = sizeof(serv_addr); // Receive the server's response
    bzero(buffer, 256);
    n = recvfrom(sockfd, buffer, 255, 0,
        (struct sockaddr *)&serv_addr, &serv_len);
    if (n < 0) {
        error("ERROR receiving from socket");
    }
    buffer[n] = '\0'; //Null-terminate the received string

    //Print the server's response
    printf("Server response: %s\n", buffer);

    // Close the socket
    close(sockfd);
    return 0;
}
```

6. Sockets in the Unix Domain

A UNIX Domain server facilitates inter-process communication (IPC) on the same system using UNIX domain sockets. Unlike Internet domain sockets that rely on IP and port numbers, UNIX domain sockets use the file system for addressing. This ensures fast and reliable communication between processes running on the same machine.

Server_UNIX: This is the server in UNIX domain source code in c.

```

1  /* a server in the unix domain. The pathname of
2  |   the socket address is passed as an argument */
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <unistd.h>
6  #include <stdlib.h>
7  #include <sys/un.h>
8  #include <stdio.h>
9  void error(const char *);
10 int main(int argc, char *argv[])
11 {
12     int sockfd, newsockfd, servlen, n;
13     socklen_t clilen;
14     struct sockaddr_un cli_addr, serv_addr;
15     char buf[80];
16
17     if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
18         error("creating socket");
19     bzero((char *) &serv_addr, sizeof(serv_addr));
20     serv_addr.sun_family = AF_UNIX;
21     strcpy(serv_addr.sun_path, argv[1]);
22     servlen = strlen(serv_addr.sun_path) +
23             sizeof(serv_addr.sun_family);
24     if (bind(sockfd, (struct sockaddr *) &serv_addr, servlen) < 0)
25         error("binding socket");
26
27     listen(sockfd, 5);
28     clilen = sizeof(cli_addr);
29     newsockfd = accept(
30         sockfd, (struct sockaddr *) &cli_addr, &clilen);
31     if (newsockfd < 0)
32         error("accepting");
33     n = read(newsockfd, buf, 80);
34     printf("A connection has been established\n");
35     write(1, buf, n);
36     write(newsockfd, "I got your message\n", 19);
37     close(newsockfd);
38     close(sockfd);
39     return 0;
40 }
41
42 void error(const char *msg)
43 {
44     perror(msg);
45     exit(0);
46 }

```

Client_UNIX:

This is the client_UNIX source code in c.

```

/* a client in the unix domain */
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
void error(const char *);

int main(int argc, char *argv[])
{
    int sockfd, servlen, n;
    struct sockaddr_un serv_addr;
    char buffer[82];

    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sun_family = AF_UNIX;
    strcpy(serv_addr.sun_path, argv[1]);
    servlen = strlen(serv_addr.sun_path) +
              sizeof(serv_addr.sun_family);
    if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
        error("Creating socket");
    if (connect(sockfd, (struct sockaddr *)&serv_addr, servlen) < 0)
        error("Connecting");
    printf("Please enter your message: ");
    bzero(buffer, 82);
    fgets(buffer, 80, stdin);
    write(sockfd, buffer, strlen(buffer));
    n = read(sockfd, buffer, 80);
    printf("The return message was\n");
    write(1, buffer, n);
    close(sockfd);
    return 0;
}

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

```

Output with executable:

```

server@server-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
server@server-VirtualBox:~/Downloads$ ls
5259 client_UNIX.c server server_enhanced server_original.c server_udp.c server_UNIX.c server_zombie.c
client_original.c code_1.99.4-175691114_amd64.deb server.c server_enhanced.c server_udp server_UNIX server_zombie
server@server-VirtualBox:~/Downloads$ ./server_UNIX 1547
A connection has been established
Hi, this is the UNIX domain socket and it is working fine! yaya
server@server-VirtualBox:~/Downloads$

```

```

server@server-VirtualBox: ~/Downloads
File Edit View Search Terminal Help
1547 client_UNIX.c server.c server_original.c server_UNIX server_zombie.c
5259 code_1.99.4-175691114_amd64.deb server_enhanced server_udp server_UNIX.c server_zombie
client_original.c server
server@server-VirtualBox:~/Downloads$ g++ -o client_UNIX client_UNIX.c
server@server-VirtualBox:~/Downloads$ ./client_UNIX 1547
Please enter your message: Hi, this is the UNIX domain socket and it is working fine! yaya
The return message was
I got your message
server@server-VirtualBox:~/Downloads$

```

7. Github Time-stamps with History

Please find below the complete github history with time stamps to my repository for Lab-1.

Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Commits

History for EE533 / LAB1 on main

All usersAll time

Commits on Jan 18, 2025

Add files via upload

Aarch0811 authored 12 minutes ago

Verified4076824

CopyDownloadCompare

Create Readme.txt

Aarch0811 authored 13 minutes ago

Verifiedfda9ddc

CopyDownloadCompare

Add files via upload

Aarch0811 authored 1 hour ago

Verifieddaac88d

CopyDownloadCompare

Add files via upload

Aarch0811 authored 1 hour ago

Verifiedae0a40f

CopyDownloadCompare

Create Readme.txt

Aarch0811 authored 2 hours ago

Verified8cb358d

CopyDownloadCompare

Add files via upload

Aarch0811 authored 2 hours ago

Verifiedb38ffea

CopyDownloadCompare

Create Readme.txt

Aarch0811 authored 2 hours ago

Verifiedf68787c

CopyDownloadCompare

Add files via upload

Aarch0811 authored 2 hours ago

Verifiedefbb5fd

CopyDownloadCompare

Create placeholder.txt

Aarch0811 authored 2 hours ago

Verified769bc9f

CopyDownloadCompare

deleted accidental file creation

Aarch0811 authored 2 hours ago

Verifiedeb7f587

CopyDownloadCompare

Create Multi-Client Connection

Aarch0811 authored 2 hours ago

Verified6fc69b2

CopyDownloadCompare

folder deleted

Aarch0811 authored 2 hours ago

Verified7c88828

CopyDownloadCompare

Multi-Client Connection

Aarch0811 authored 2 hours ago

Verified25a1ac6

CopyDownloadCompare

Multi-Client Connection

Aarch0811 authored 2 hours ago

Verified59836ae

CopyDownloadCompare

made changes

Aarch0811 authored 2 hours ago

Verifiedda49666

CopyDownloadCompare

Add files via upload

Aarch0811 authored 2 hours ago

Verified268626a

CopyDownloadCompare

readme_file.txt

Aarch0811 authored 2 hours ago

Verified040111f

CopyDownloadCompare

End of commit history for this file