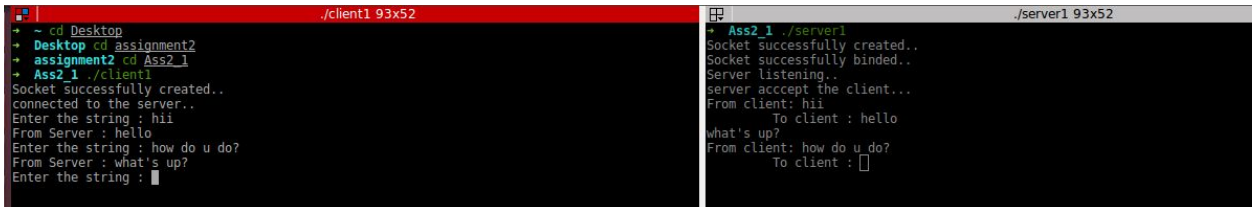
Deepika Choudhary
17114025
CS I
B.Tech IIIyr

<u>Lab Assignment 2</u>
COMPUTER NETWORK LABORATORY

This assignment aims to make us familiar with the hardware and software aspects of computer networking and extracting information related to computer networking using C/C++ programs.

Problem Statement 1 :
Write a socket program in C to connect two nodes on a network to communicate with each other, where one socket listens on a particular port at an IP, while other socket reaches out to the other to form a connection.



```
./client1 93x52
→  ~ cd Desktop
→  Desktop cd assignment2
→  assignment2 cd Ass2_1
→  Ass2_1 ./client1
Socket successfully created..
connected to the server..
Enter the string : hii
From Server : hello
Enter the string : how do u do?
From Server : what's up?
Enter the string :
```

```
./server1 93x52
→  Ass2_1 ./server1
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hii
         To client : hello
what's up?
From client: how do u do?
         To client :
```

Algorithm &Data Structures Used:

Socket creation:

`int sockfd = socket(domain, type, protocol)`

sockfd: socket descriptor, an integer (like a file-handle)
domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
type: communication type
SOCK_STREAM: TCP(reliable, connection-oriented)
SOCK_DGRAM: UDP(unreliable, connectionless)
protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on the protocol field in the IP header of a packet.

Setsockopt:

`int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);`

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in the reuse of address and port. Prevents error such as: “address already in use”.

Bind:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

After the creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

Listen:

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

Accept:

```
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

Stages for Client

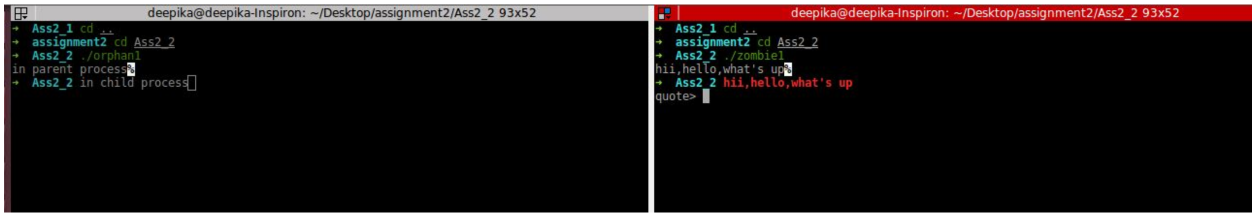
Socket connection: Exactly same as that of server’s socket creation

Connect:

int connect(int sockfd, const struct sockaddr *addr socklen_t addrlen);

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Problem Statement 2 :
Write a C program to demonstrate both Zombie and Orphan process.



```
deepika@deepika-Inspiron: ~/Desktop/assignment2/Ass2_2 93x52
→  Ass2_1 cd ..
→  assignment2 cd Ass2_2
→  Ass2_2 ./orphan1
in parent process%
→  Ass2_2 in child process
```

```
deepika@deepika-Inspiron: ~/Desktop/assignment2/Ass2_2 93x52
→  Ass2_1 cd ..
→  assignment2 cd Ass2_2
→  Ass2_2 ./zombie1
hii,hello,what's up%
→  Ass2_2 hii,hello,what's up
quote>
```

Algorithm &Data Structures Used:

Zombie Process:
A process which has finished the execution but still has an entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.In the code, the child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence doesn't call wait() and the child process's entry still exists in the process table.

Orphan Process:

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.In the following code, parent finishes execution and exits while the child process is still executing and is called an orphan process now.However, the orphan process is soon adopted by the init process, once its parent process dies.