##### **)** Implement echo server using TCP/UDP in iterative/concurrent logic.

**OBJECTIVES:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS intraction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbersing and process relation.
5. To knows the iterative and concurrent server concept.

**THEORY:**

JAVA SOCKET PROGRAMMING

1. TCP Echo Client
   * In the TCP Echo client a socket is created.
   * Using the socket a connection is made to the server using the connect() function.
   * After a connection is established, we send messages input from the user and display the data received from the server using send() and read() functions.
2. UDP Echo Client

* In the UDP Echo client a socket is created.
* Then we bind the socket.
* After the binding is successful, we send messages input from the user and display the data received from the server using sendto() and recvfrom() functions.

1. TCP Echo Server



* In the TCP Echo server, we create a socket and bind to an advertised port number.
* After binding, the process listens for incoming connections.
* Then an infinite loop is started to process the client requests for connections.
* After a connection is requested, it accepts the connection from the client machine and forks a new process.
* The new process receives data from the client using recv() function and echoes the same data using the send() function.
* Please note that this server is capable of handling multiple clients as it forks a new process for every client trying to connect to the server.

1. UDP Echo Server



* In the UDP Echo server, we create a socket and bind to an advertised port number.
* Then an infinite loop is started to process the client requests for connections.
* The process receives data from the client using recvfrom () function and echoes the same data using the sendto() function.
* Please note that this server is capable of handles multiple clients automatically as UDP is a datagram based protocol hence no exclusive connection is required to a client in this case.

#### Program:

SERVER:

#include <stdio.h> #include <stdlib.h> #include <unistd.h> int main()

{

char str[100];

int listen\_fd, comm\_fd; struct sockaddr\_in servaddr;

listen\_fd = socket(AF\_INET, SOCK\_STREAM, 0); bzero( &servaddr, sizeof(servaddr)); servaddr.sin\_family = AF\_INET; servaddr.sin\_addr.s\_addr = htons(INADDR\_ANY); servaddr.sin\_port = htons(22000);

bind(listen\_fd, (struct sockaddr \*) &servaddr, sizeof(servaddr)); listen(listen\_fd, 10);

comm\_fd = accept(listen\_fd, (struct sockaddr\*) NULL, NULL); while(1)

{

bzero( str, 100); read(comm\_fd,str,100); printf("Echoing back - %s",str); write(comm\_fd, str, strlen(str)+1);

}

}

CLIENT:

#include <stdio.h> #include <unistd.h> #include <errno.h> #include <string.h> #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h>

int main(int argc,char \*\*argv)

{

int sockfd,n;

char sendline[100]; char recvline[100];

struct sockaddr\_in servaddr; sockfd=socket(AF\_INET,SOCK\_STREAM,0); bzero(&servaddr,sizeof servaddr); servaddr.sin\_family=AF\_INET; servaddr.sin\_port=htons(22000); inet\_pton(AF\_INET,"127.0.0.1",&(servaddr.sin\_addr)); connect(sockfd,(struct sockaddr \*)&servaddr,sizeof(servaddr)); while(1)

{

bzero( sendline, 100);

bzero( recvline, 100);

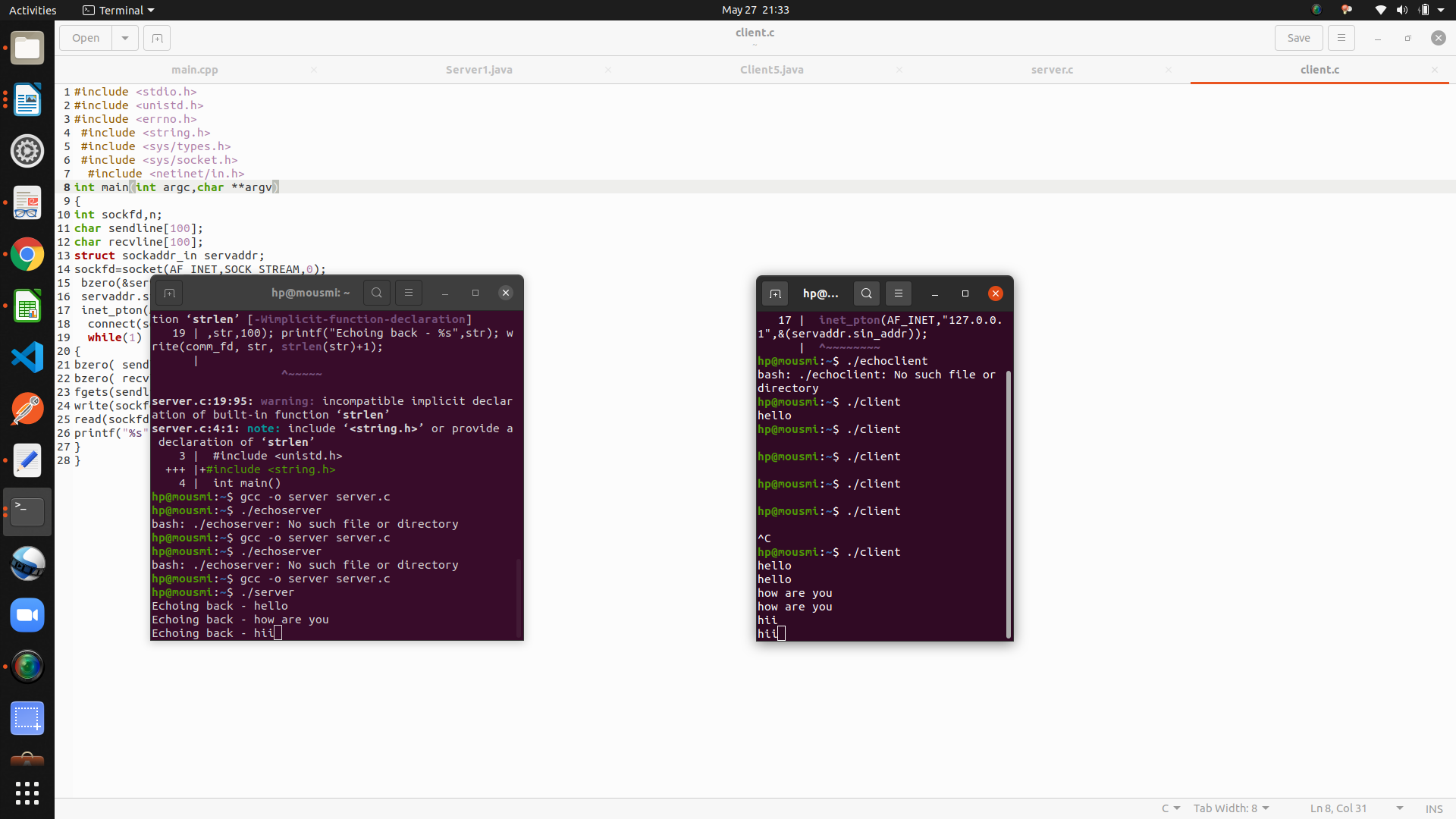
fgets(sendline,100,stdin); /\*stdin = 0 , for standard input \*/ write(sockfd,sendline,strlen(sendline)+1); read(sockfd,recvline,100);

printf("%s",recvline);

}

}

OUTPUT:-

****

**Conclusion:**

Using TCP connection between Client and server, the echo server mechanism was implemented using iterative logic.

#### Reference:

[www.cs.cf.ac.uk/Dave/C/CE.html](http://www.cs.cf.ac.uk/Dave/C/CE.html)