<u>**Machine problem 1**</u>

*Please go through the github repository for better readability and understanding of the code.

https://github.tamu.edu/baruah-dharmendra/ECEN602_Team04

The README.md file has all the implementation details and the testcases.

Readme file:

```
# TCP-Client-Server-in-C++

## Purpose:

This Project is developed as a part of Machine Problem 1 of Computer
Networks and Communication course. It is performed as a team of two where
we are supposed to implement a client and server for a simple TCP echo
service.

---
## Implementation:

 The Client Server model performs the following implementation:
1.  Start the server first with the command line: echos Port, where Port
is the port number on which the server is listening. The server supports
multiple simultaneous connections.
2.  Start the client second with a command line: echo IPAdr Port, where
IPAdris the IPv4 address of the server and Portis the port number on which
the server is listening.
3.  The client reads a line of text from its standard input and sends the
same signal back to the server.
4.  The server reads the line from its network input and echoes the line
back to the client.
5.  The client reads the echoed line and prints it on its standard input.
6.  When the client reads an EOF from its standard input i.e. ,Äú:
exit,Äù, it closes the socket and exits. When the client closes the
socket, the server receive a TCP FIN packet, and the server child
process,Äô read() command returns with a 0, after which the child process
exits.

---
## Usage
In the Server Client application multiple clients can be connected to the
server. Each time a server accepts a client connection it shows a message
with the client information. When a client sends a message to the server,
it receives it and sends it back to the client which is again printed in
the client console. On the client, just type in a message and hit enter to
send it to the server.
```

## Running

### Installation:

Clone this repository
```
git@github.tamu.edu:baruah-dharmendra/ECEN602_Team04.git
```

### Building:

For this we will need standard C++ compiler installed in the machine in which the program is run. To build it can be directly done from the make file or individually by compiling the server client.

For building without the make file:

Build the client: ``` g++ -o client tcpClient.cpp ```

Build the server: ``` g++ -o server tcpServer.cpp ```

### Execution:

In the first terminal run: ``` ./server ```

In another terminal run: ``` ./client ```

After execution to start the server type:
echos Port(ex. 4444)

```echos 4444```

If the server response following should be visible in the terminal.
```
[+]Server Socket is created.

[+]Bind to port Port(ex. 4444)

[+]Listening....
```
In client terminal, we can start the client by:
echo IPAdr(ex. 127.0.0.1) Port(ex. 4444)

```echo 127.0.0.1 4444```

The client should respond to it with the following message:
```
[+]Client Socket is created.

[+]Connected to Server.
```

Client:
```
Once the connection to the client is established the server should show
the following message:

```
Connection accepted from 127.0.0.1:47506
```
### Testing:

To test it, if at the client terminal, following input is enter:

```
Client: test1
```

Then, the message will be echoed back by the server and following message
should be displayed in the client side.

```
Client: test1
Server: test1
```

Whereas, the server terminal should display the client message as follows
to acknoledge read and write at the servers end.
```
Client: test1
```

### Exiting:

For closing the client `:exit` command is used as EOF.

Following should appear at the client and the server side.

Client terminal:

```
Disconnected from server.
```
Server terminal:

```
Disconnected from 127.0.0.1:47506
```

## Test cases

Testcase 1:

Line of text terminated by a newline
```

![](image_1.png)

Testcase 2:

Line of text the maximum line length without a newline

![](image_2.png)

Testcase 3:

Line with no characters and EOF

![](image_3.png)

Testcase 4:

Client terminated after entering text

![](image_4.png)

Testcase 5:

Three clients connected to the server

![](Image_5.png)

## Team

@dharmendrabaruah
@yehtungchi

## Effort

The entire project was completed with equal efforts from either member of the team maintaining synergy. It was carried out in the university library, where both of the members were responsible for the analysis, coding, debugging, testing and documentation of the server client application.

Client Code:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <vector>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>


using namespace std;

int main()
{

      int check = 0;
      string echo_IPAdr_port;
      char IPAdr[] = "0.0.0.0"; //"127.0.0.1"
      int Port_No; //define PORT NUMBER

      //command line: echo IPAdr Port, where IPAdr is the IPv4 address
      // of the server in dotted decimal notation and Port is the port
number
      int command_line=1;
      while (command_line)
      {
            getline(cin, echo_IPAdr_port);

            if(echo_IPAdr_port==":exit"){
                  command_line=0;
            }

            size_t pos = 0;
            string token;
            char delimiter = ' ';
            vector<string> temp;
            // split the input string
            while ((pos = echo_IPAdr_port.find(delimiter)) !=
std::string::npos){
                  token = echo_IPAdr_port.substr(0, pos);
                  temp.push_back(token);
                  echo_IPAdr_port.erase(0, pos + 1);
            }
            temp.push_back(echo_IPAdr_port);
            if (temp[0] == "echo"){
                  for(int i=0; i<temp[1].length(); i++){
                        IPAdr[i] = temp[1][i];
```

```
                }
                Port_No = stoi(temp[2]);
                break;
            }else{
                cout << "[-]Error in commandline." << endl;
            }
        }

        int clientSocket, ret;
        struct sockaddr_in serverAddr;
        char buffer[1024];
        //create a socket
        clientSocket = socket(AF_INET, SOCK_STREAM, 0);
        if (clientSocket < 0)
        {
            printf("[-]Error in connection.\n");
            exit(1);
        }
        printf("[+]Client Socket is created.\n");

        memset(&serverAddr, '\0', sizeof(serverAddr));
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(Port_No);
        // Ipv4 address ex."127.0.0.1"
        serverAddr.sin_addr.s_addr = inet_addr(IPAdr);
        //connect to the server on the socket
        ret = connect(clientSocket, (struct sockaddr *)&serverAddr, 
    sizeof(serverAddr));
        if (ret < 0)
        {
            printf("[-]Error in connection.\n");
            exit(1);
        }
        printf("[+]Connected to Server.\n");

        while (1)
        {
            //enter line of text
            printf("Client: \t");
            scanf("%s", &buffer[0]);
            send(clientSocket, buffer, strlen(buffer), 0);
            //compare to close the client if there is command :exit
            if (strcmp(buffer, ":exit") == 0)
            {
                close(clientSocket);
                printf("[-]Disconnected from server.\n");
                exit(1);
            }
            //if data from the server isn't recieved print error
            if (recv(clientSocket, buffer, 1024, 0) < 0)
            {
                printf("[-]Error in receiving data.\n");
            }
            else
            {
                  //display response
                  printf("Server: \t%s\n", buffer);
            }
      }

      return 0;
}
```

Server Code:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

using namespace std;

int main(){

      int check = 0;
      string echo_port;
      int Port_No;

      //Command line: echos Port, where Port is the port number.
      while(true){
            getline( cin, echo_port );
            if(echo_port.substr(0, 5)=="echos"){
                  Port_No = stoi(echo_port.substr(5));
                  break;
            }else{
                  cout << "[-]Error in commandline." << endl;
            }
      }

      int sockfd, ret;
      struct sockaddr_in serverAddr;

      int newSocket;
      struct sockaddr_in newAddr;

      socklen_t addr_size;

      char buffer[1024];
      pid_t childpid;
```

```
	//create a socket
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if(sockfd < 0){
		printf("[-]Error in connection.\n");
		exit(1);
	}
	printf("[+]Server Socket is created.\n");

	memset(&serverAddr, '\0', sizeof(serverAddr));

	//bind the ipaddress and port to the socket
	serverAddr.sin_family = AF_INET;
	//PORT ex. 4444
	serverAddr.sin_port = htons(Port_No);
	//server address ex. 127.0.0.1
	serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	ret = bind(sockfd, (struct sockaddr*)&serverAddr,
sizeof(serverAddr));

	if(ret < 0){
		printf("[-]Error in binding.\n");
		exit(1);
	}
	printf("[+]Bind to port %d\n", Port_No);
	//listen for the socket
	if(listen(sockfd, 10) == 0){
		printf("[+]Listening....\n");
	}else{
		printf("[-]Error in binding.\n");
	}


	while(1){
		newSocket = accept(sockfd, (struct sockaddr*)&newAddr,
&addr_size);
		if(newSocket < 0){
			exit(1);
		}
		printf("Connection accepted from %s:%d\n",
inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));

		if((childpid = fork()) == 0){
			close(sockfd);

			while(1){
				//wait for client to send data
				recv(newSocket, buffer, 1024, 0);
				if(strcmp(buffer, ":exit") == 0){
					printf("Disconnected from %s:%d\n",
inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));
					break;
				}else{
					printf("Client: %s\n", buffer);
					//echo message back to client
```

```
                        send(newSocket, buffer, strlen(buffer), 0);
                        bzero(buffer, sizeof(buffer));
                    }
                }
            }

        }
        //close the socket
        close(newSocket);

        return 0;
}
```