


```

        sizeof(opt)))
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address,
        sizeof(address)) < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
if (listen(server_fd, 3) < 0)
{
    perror("listen");
    exit(EXIT_FAILURE);
}
if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
        (socklen_t *)&addrlen)) < 0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}
valread = read(new_socket, buffer, 1024);
printf("\n%s: Message from client: \n%s\n", ctime(&t), buffer);
// cout<<typeid(valread).name()<<endl;
if (buffer[0] == '/')
{
    string input = buffer;
    input = input.substr(1, input.size() - 2);
    size_t found = input.find(" ");
    string cmd = input.substr(0, found);
    string path = input.substr(found, input.size() - 1);
    // cout << input << endl;
    // cout << found << endl;
    // cout << cmd << endl;
    // cout << path << endl;
    string command = "";
    if (cmd == "list")
    {
        command = "ls " + path + " >file.txt";
    }
    else if (cmd == "show")
    {
        command = "cat " + path + " >file.txt";
    }
    // cout<<"Command request:"<< command<<endl;

```

```

        system(command.c_str());
        std::ifstream t("file.txt");
        std::stringstream buff;
        buff << t.rdbuf();
        strcpy(msg, buff.str().c_str());
    }
    else
    {
        printf("Enter response for client:\n");
        fgets(msg, sizeof(msg), stdin);
    }

    send(new_socket, msg, strlen(msg), 0);
    printf("\n!!Reply Sent!!\n\n");

    // closing the connected socket
    close(new_socket);
    // closing the listening socket
    shutdown(server_fd, SHUT_RDWR);
    return 0;
}

```

Algorithm:

Step-1: Start

Step-2: Create a time object to display time stamps

Step-3: Define a buffer of size 1024 and msg of size 100.

Step-4: Use socket functions to create a server-side TCP socket on port 8080

Step-5: Read the value from socket to buffer

Step-6: If first character of buffer is '/' then it is a command request, hence split it into command and path and store them in cmd and path variables respectively.

Step-7: Check if cmd is "list" or "show" if it is list then concat "ls" with path and store it in command variable else concat "cat" with path and a redirection operation.

Step-8: Use system command to run the string variable command but since it doesn't return the output to a string so we can't transmit it directly through socket. Hence, we will store output in a temporary file 'file.txt' and read it to a string and transmit it through socket.

Step-9: If buffer does not begin with '/' hence is not a command rather a plain message which can be responded manually hence prompt the user and ask for response and transmit it through socket.

Step-10: Display the response sent

Step-11: End

client.c

```

// Client-side C program
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>

```

```

#include <unistd.h>
// #include <bits/stdc++.h>
#include<time.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
    time_t t;
    time(&t);
    int sock = 0, valread, client_fd;
    struct sockaddr_in serv_addr;
    //string to store message to server
    char msg[100];
    //prompting user to enter message
    printf("Enter message for server:\n");
    fgets(msg, sizeof(msg), stdin);
    char buffer[1024] = { 0 };
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
        <= 0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if ((client_fd
        = connect(sock, (struct sockaddr*)&serv_addr,
            sizeof(serv_addr)))
        < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }
    send(sock, msg, strlen(msg), 0);
    printf("\n%s: !!Message Sent!!\n\n", ctime(&t));
    valread = read(sock, buffer, 1024);
    printf("\n%s: Response from server: \n%s\n", ctime(&t), buffer);

    // closing the connected socket
    close(client_fd);
    return 0;
}

```

Algorithm:

Step-1: Start

Step-2: Create a time object to display time stamps

Step-3: Define a buffer of size 1024 and msg of size 100.

Step-4: Use socket functions to create a server-side TCP socket on port 8080

Step-5: Prompt the user for message to be sent to server along with timestamp from time object and store it in msg variable created earlier.

Step-6: Send the message stored in msg through send function.

Step-7: Get the response from the server through read function and store it in buffer we created earlier.

Step-8: Add time stamp to the message through time object.

Step-9: Display the response

Step-10: End

Input/Outputs:

Creating folder and files to be used:

```

/ For a light heart lives long. \
|                               |
\ -- Shakespeare, "Love's Labour's Lost" /
-----
      ^ ^
      (oo)\-----
      (--) \      )\ \
          ||----w |
          ||      ||

emperor-kautilya@pop-os:~/Desktop/folder$ touch {a..c}.txt
emperor-kautilya@pop-os:~/Desktop/folder$ cal > a.txt
emperor-kautilya@pop-os:~/Desktop/folder$
```

Test case-1: Simple responses

Server-side terminal:

```
emperor-kautilya@pop-os: ~/nET Prog/Lab-2
/ The ripest fruit falls first. \
|                               |
\ -- William Shakespeare, "Richard II" /
-----
      ^ ^
      (oo)\_____
      (__) \       )\/\
           ||----w |
           ||     ||

emperor-kautilya@pop-os:~/nET Prog/Lab-2$ g++ new-server.cpp -o new-server.out
emperor-kautilya@pop-os:~/nET Prog/Lab-2$ ./new-server.out

Fri Jan 27 18:44:21 2023
: Message from client:
Hello this is client

Enter response for client:
Hello client this is server

!!Reply Sent!!
```

Client-side terminal:

```
emperor-kautilya@pop-os: ~/nET Prog/Lab-2
/ Writing is turning one's worst moments \
| into money.                             |
|                                         |
\ -- J.P. Donleavy                       /
-----
      ^ ^
      (oo)\_____
      (__) \       )\/\
           ||----w |
           ||     ||

emperor-kautilya@pop-os:~/nET Prog/Lab-2$ ./client.out
Enter message for server:
Hello this is client

Fri Jan 27 18:44:48 2023
: !!Message Sent!!

Fri Jan 27 18:44:48 2023
: Response from server:
Hello client this is server
```

Test case -2: List folder content

Server-side terminal:

```
emperor-kautilya@pop-os: ~/nET Prog/Lab-2 x emperor-kautilya@pop-os: ~/nET Prog/Lab-2 x
emperor-kautilya@pop-os:~/nET Prog/Lab-2$ g++ new-server.c++ -o new-server.out
emperor-kautilya@pop-os:~/nET Prog/Lab-2$ ./new-server.out

Fri Jan 27 18:35:38 2023
: Message from client:
/list ~/Desktop/folder/
```

Client-side terminal:

```
emperor-kautilya@pop-os: ~/nET Prog/Lab-2 x emperor-kautilya@pop-os: ~/nET Prog/Lab-2 x
emperor-kautilya@pop-os:~/nET Prog/Lab-2$ ./client.out
Enter message for server:
/list ~/Desktop/folder/

Fri Jan 27 18:35:44 2023
: !!Message Sent!!

Fri Jan 27 18:35:44 2023
: Response from server:
a.txt
b.txt
c.txt
```

Test Case-3: Showcase File contents

Server-side terminal:

```
emperor-kautilya@pop-os: ~/nET Prog/Lab-2 x emperor-kautilya@pop-os: ~/nET Prog/Lab-2 x
-----
/ The ripest fruit falls first. \
|                               |
\ -- William Shakespeare, "Richard II" /
-----

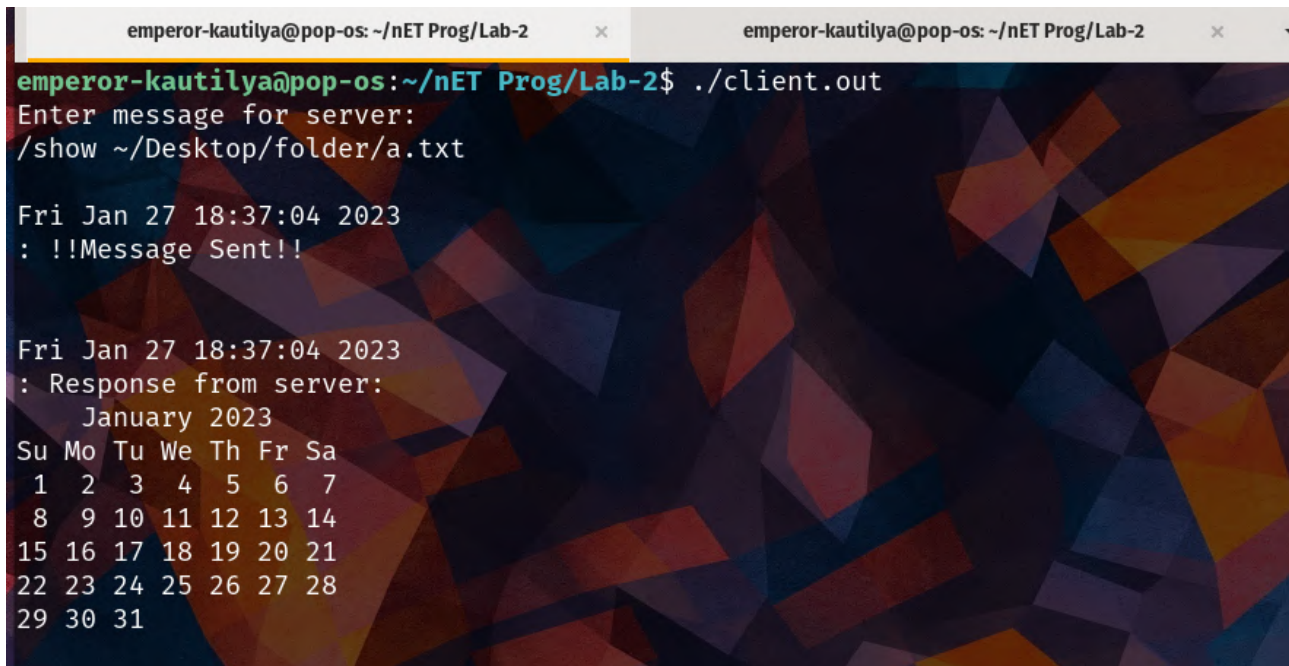
      ^  ^
      (oo)\_____
      (__) \       )\/\
           ||----w |
           ||     ||

emperor-kautilya@pop-os:~/nET Prog/Lab-2$ g++ new-server.c++ -o new-server.out
emperor-kautilya@pop-os:~/nET Prog/Lab-2$ ./new-server.out

Fri Jan 27 18:44:21 2023
: Message from client:
Hello this is client

Enter response for client:
Hello client this is server
```


Client-side terminal:



```
emperor-kautilya@pop-os: ~/nET Prog/Lab-2  x  emperor-kautilya@pop-os: ~/nET Prog/Lab-2  x
emperor-kautilya@pop-os:~/nET Prog/Lab-2$ ./client.out
Enter message for server:
/show ~/Desktop/folder/a.txt

Fri Jan 27 18:37:04 2023
: !!Message Sent!!

Fri Jan 27 18:37:04 2023
: Response from server:
  January 2023
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Language in use: Java

Server.java

```
// A Java program for a Server
import java.net.*;
import java.io.*;
import java.io.File;
import java.util.Scanner;
import java.sql.Timestamp;
import java.util.Date;

public class Server {
    // initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;
    private BufferedReader input = null;
    private DataOutputStream out = null;

    // constructor with port
    public Server(int port) throws IOException {

        // starts server and waits for a connection
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            socket = server.accept();
            System.out.println("Client accepted");

            input = new BufferedReader(new InputStreamReader(System.in));
            // takes input from the client socket
            in = new DataInputStream(
                new BufferedInputStream(socket.getInputStream()));

            out = new DataOutputStream(
                socket.getOutputStream());
            String line = "";

            // reads message from client until "Over" is sent
            while (!line.equals("Over")) {
                try {
                    Date date = new Date();
                    line = in.readUTF();
                    if (line.charAt(0) == '/') {
                        String cmd = line.substring(1, line.indexOf(' '));
                        String path = line.substring(line.indexOf(' ') + 1);
```

```

        if (cmd.equalsIgnoreCase("list")) {
            System.out.println("\n"+new
Timestamp(date.getTime())+ " Command request: \n"+line);
            File directoryPath = new File("/home/emperor-
kautilya" + path);

            String output = "";
            String contents[] = directoryPath.list();
            for (int i = 0; i < contents.length; i++) {
                output = output + contents[i] + "\n";
            }
            System.out.println(new Timestamp(date.getTime())
+" : \nResponse Sent!!");
            out.writeUTF(output);
        } else if (cmd.equalsIgnoreCase("show")) {
            System.out.println("\n"+new
Timestamp(date.getTime())+ " Command request: \n"+line);
            File filePath = new File("/home/emperor-
kautilya" + path);

            Scanner fileread = new Scanner(filePath);
            String output = "";

            while (fileread.hasNextLine()) {
                output = output + fileread.nextLine() + "\
n";
            }
            System.out.println(new Timestamp(date.getTime())
+" : \nResponse Sent!!");
            fileread.close();
            out.writeUTF(output);
        }
    } else if (!line.equalsIgnoreCase("Over"))
    {
        System.out.println("\n"+new
Timestamp(date.getTime())+" Client : ");
        System.out.println(line);
        System.out.println(new Timestamp(date.getTime())+"
Response : ");

        line = input.readLine();
        out.writeUTF(line);
    }
    else{
        out.writeUTF("Over");
    }

} catch (IOException i) {
    System.out.println(i);
}

}
System.out.println("Closing connection");

// close connection

```

```

        socket.close();
        in.close();
    } catch (

        IOException i) {
        System.out.println(i);
    }
}

public static void main(String args[]) throws IOException {
    Server server = new Server(5000);
}
}

```

Algorithm:

Step-1: Start class Server

Step-2: Define Socket object as socket, SocketServer object as server, DataInputStream as in, BufferedReader as input, DataOutputStream as out.

Step-3: Create a constructor server which has port as a parameter to start server on particular port and wait for connection.

Step-4: Initialize server with a socket on port from port variable.

Step-5: Display the current status of connection as "waiting"

Step-6: Initialize socket object with server. accept() to accept client connections and display "client accepted".

Step-7: Initialize input with buffered reader object to take input from user.

Step-8: Initialize in with data input stream object to take input from socket.

Step-9: Initialize out with data output stream object to give output to socket.

Step-10: Create a string line which will be used to store messages.

Step-11: While line is not equal to over repeat step-12 to step-18

Step-12: Create date object to time stamp your messages.

Step-13: Use readUTF function to get input from socket.

Step-14: If first character of line is '/' then recognize it as a command operation.

Step-15: Use substring function to separate command with path and store them in variables cmd and path respectively.

Step-16: Check if cmd is list, if it is the display the requested operation with time stamp, and use directoryPath function to get path and they use path.list() to get content of files stored at that path. Since it returns an array of string, convert it to a single string using a for loop separated by new line and they pass it through the socket using out.writeUTF() function.

Step-17: Check if cmd is show, if it is the display the requested operation with time stamp, and use directoryPath function to get path and they use path.list() to get content of file showcased path using Scanner. Since it returns a Scanner object. TO convert to a single string, use a for loop separated by new line and they pass it through the socket using out.writeUTF() function.

Step-18: Else check if it is not over in which case it is a simple communication message hence prompt the user with time stamp and pass the input string through the socket.

Step-19: Create a main function and create a Server class object to create a server side TCP socket on port 8080.

Step-20: End class Server

Client.java

```
// A Java program for a Client
import java.io.*;
import java.net.*;
import java.sql.Timestamp;
import java.util.Date;

public class Client {
    // initialize socket and input output streams
    private Socket socket = null;
    private BufferedReader input = null;
    private DataOutputStream out = null;
    private DataInputStream in= null;

    // constructor to put ip address and port
    public Client(String address, int port) throws IOException
    {
        // establish a connection
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // takes input from terminal
            input = new BufferedReader (new InputStreamReader(System.in));

            // sends output to the socket
            out = new DataOutputStream(
                socket.getOutputStream());

            in = new DataInputStream(
                new BufferedInputStream(socket.getInputStream()));

        }
        catch (UnknownHostException u) {
            System.out.println(u);
            return;
        }
        catch (IOException i) {
            System.out.println(i);
            return;
        }
    }

    // string to read message from input
    String line = "";

    // keep reading until "Over" is input
    while (!line.equals("Over")) {
```



```

        try {
            Date date = new Date();
            System.out.println("\n"+new Timestamp(date.getTime())+"
Message : ");
            line = input.readLine();
            out.writeUTF(line);
            line = in.readUTF();
            if(!line.equalsIgnoreCase("Over"))
            {
                System.out.println(new Timestamp(date.getTime())+"
Response : ");
                System.out.println(line);
            }
            else
                System.out.println("Connection terminated");
        }
        catch (IOException i) {
            System.out.println(i);
        }
    }

    // close the connection
    try {
        input.close();
        out.close();
        socket.close();
    }
    catch (IOException i) {
        System.out.println(i);
    }
}

public static void main(String args[]) throws IOException
{
    Client client = new Client("127.0.0.1", 5000);
}
}

```

Algorithm:

Step-1: Start class Client

Step-2: Define Socket object as socket, SocketServer object as server, DataInputStream as in, BufferedReader as input, DataOutputStream as out.

Step-3: Create a constructor client which has port and address as a parameter to connect to particular server through particular port.

Step-4: Use a try and catch block for exception handling in case the connection establishment fails.

Step-5: Initialize server with a socket on port and address from port and address variables.

Step-6: Display the current status of connection as "Connected".

Step-7: Initialize input with buffered reader object to take input from user.

Step-8: Initialize in with data input stream object to take input from socket.

Step-9: Initialize out with data output stream object to give output to socket.

Step-10: Create a string line which will be used to store messages.

Step-11: While line is not equal to over repeat step to step

Step-12: Create date object to time stamp your messages.

Step-13: Prompt the user with time stamp and pass the input string and use writeUTF function to output to socket.

Step-14: Read the text from socket using readUTF function and if it is not over display it with timestamp on the console.

Step-15: Else display "connection terminated"

Step-16: Create a main function and create a Client class object to create a server side TCP socket on port 8080 and address localhost.

Step-17: End class Client.

Input/Outputs:

Server-Side Terminal:

```
-----\
/ You should emulate your heros, but \
| don't carry it too far. Especially if |
\ they are dead.                      /
-----\
      ^ ^
      (oo)\
      (__) \
          ||---w ||
          ||    ||

emperor-kautilya@pop-os:~/nET Prog/Lab-$ javac Server.java
emperor-kautilya@pop-os:~/nET Prog/Lab-$ java Server
Server started
Waiting for a client ...
Client accepted

2023-01-28 12:49:16.069 Client :
Hi server
2023-01-28 12:49:16.069 Response :
Hi client, I am online.

2023-01-28 12:49:58.404 Command request:
/list /Desktop/folder
2023-01-28 12:49:58.404 :
Response Sent!!

2023-01-28 12:50:16.62 Command request:
/show /Desktop/folder/a.txt
2023-01-28 12:50:16.62 :
Response Sent!!

2023-01-28 12:50:29.747 Client :
Thanks for the resources
2023-01-28 12:50:29.747 Response :
You are welcome
Closing connection
emperor-kautilya@pop-os:~/nET Prog/Lab-$
```

Client-Side Terminal:

```
emperor-kautilya@pop-os:~/nET Prog/Lab-$ java Client  
Connected
```

```
2023-01-28 12:49:16.069 Message :  
Hi server
```

```
2023-01-28 12:49:16.069 Response :  
Hi client, I am online.
```

```
2023-01-28 12:49:58.411 Message :  
/list /Desktop/folder
```

```
2023-01-28 12:49:58.411 Response :  
c.txt  
a.txt  
b.txt
```

```
2023-01-28 12:50:16.62 Message :  
/show /Desktop/folder/a.txt
```

```
2023-01-28 12:50:16.62 Response :  
January 2023
```

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

```
2023-01-28 12:50:29.747 Message :  
Thanks for the resources
```

```
2023-01-28 12:50:29.747 Response :  
You are welcome
```

```
2023-01-28 12:50:52.249 Message :  
Over
```

```
Connection terminated
```

```
emperor-kautilya@pop-os:~/nET Prog/Lab-$
```