# A
# Project Report
# On

**"CLIENT SERVER BASED CHARACTER TO ASCII AND VICE-VERSA CODE CONVERSION APPLICATION USING TCP."**

**Department of Computer Science & Engineering**

**NATIONAL INSTITUE OF TECHNOLOGY PATNA**

**University Campus, Bihar - 800005**

**Submitted By:**

| Name | Roll No. |
|---|---|
| Sujeet Kumar | **2144048** |
| Hemant Mewade | **2144049** |
| Yogesh Patel | **2144050** |
| Rinki Raghuwanshi | **2144051** |

**Course Code: MC440209**

**Course Title: System and networks programming lab**

**Branch: CSE(MCA)**

# TABLE OF CONTENTS

# LIST OF FIGURES

## AIM OF THE EXPERIMENT

Implementation of a Client Server based Character to ASCII and vice-versa code conversion application using TCP.

# 1. INTRODUCTION

### 1.1 Process-to-Process delivery

The data link layer is responsible for delivery of frames between two neighbouring nodes over a link. This is called node-to-node delivery. The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery. Communication on the Internet is not defined as the exchange of data between two nodes or between two hosts. Real communication takes place between two processes (application programs). We need process-to-process delivery. However, at any moment, several processes may be running on the source host and several on the destination host. To complete the delivery.

The transport layer is responsible for process-to-process delivery-the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship.

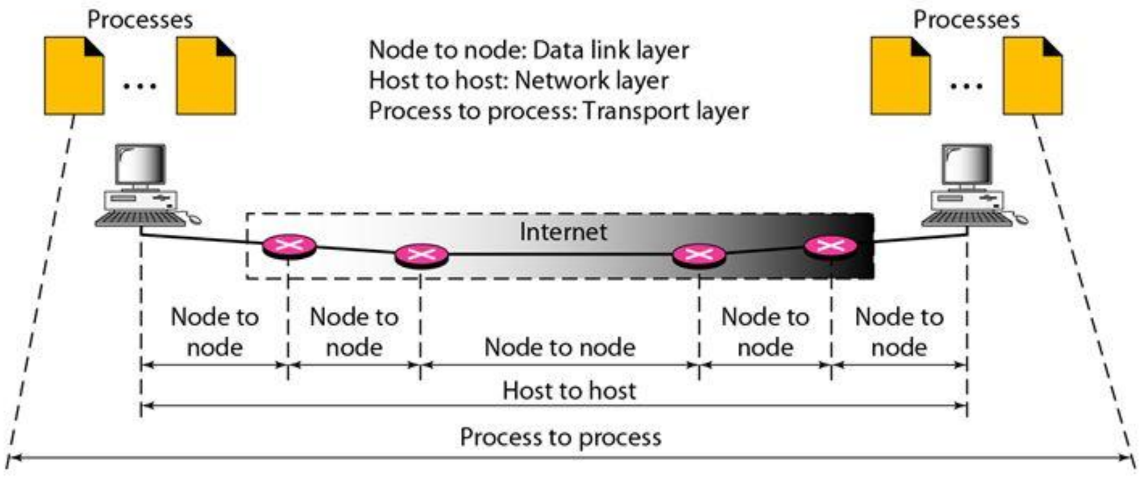


**Fig -1 - Process to Process Delivery**

### 1.2 Client/Server Paradigm

The best ways to achieve process-to-process communication is through the client/server paradigm. A process on the local host, called a client, needs services from a process usually on the remote host, called a server. Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine.

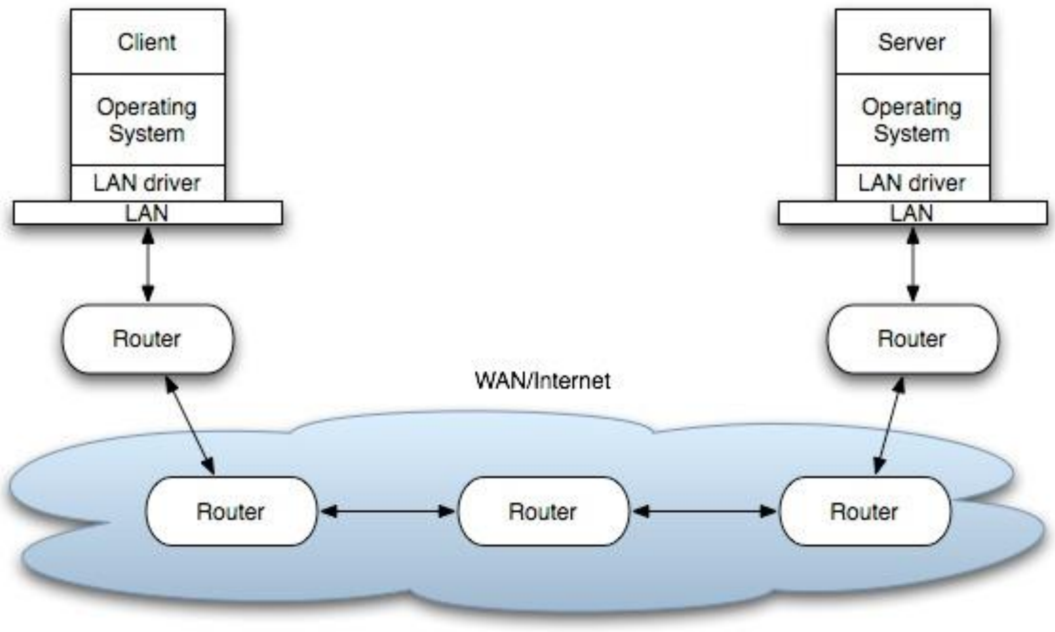


**Fig-2- Client Server Interaction**

## 1.3 Connectionless Versus Connection-Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

### Connectionless Service

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either. We will see shortly that one of the transport layer protocols in the Internet model, UDP, is connectionless.

### USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to process communication instead of host-to-host communication. Also, it performs very limited error checking. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

**Connection-Oriented Service**

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. We will see shortly that TCP and SCTP are connection-oriented protocols.

**TCP**

TCP is a process-to-process (program-to-program) protocol. TCP is a connection oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level. In brief, TCP is called a connection-oriented, reliable transport protocol. It adds connection-oriented and reliability features to the services of IP.

**A TCP Connection**

TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a message are then sent over this virtual path. Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted. Unlike TCP, IP is unaware of this retransmission. If a segment arrives out of order, TCP holds it until the missing segments arrive; IP is unaware of this reordering.

In TCP, connection-oriented transmission requires three phases: c, data transfer, and connection termination.

## 1.4 Connection Establishment Phase

Transmission of data is done in full-duplex mode. The connection establishment in TCP is mainly termed as three-way handshaking. Let us understand this with the help of an example: An application program called client wants to make a connection with another application program called server by using the TCP as the transport layer protocol.

Mainly the process starts with the server, the server program mainly tells the TCP that it is ready to accept a connection. This is mainly known as a request for a passive open. Though the server TCP is ready to accept any connection from any machine of the world and it cannot make the connection itself.

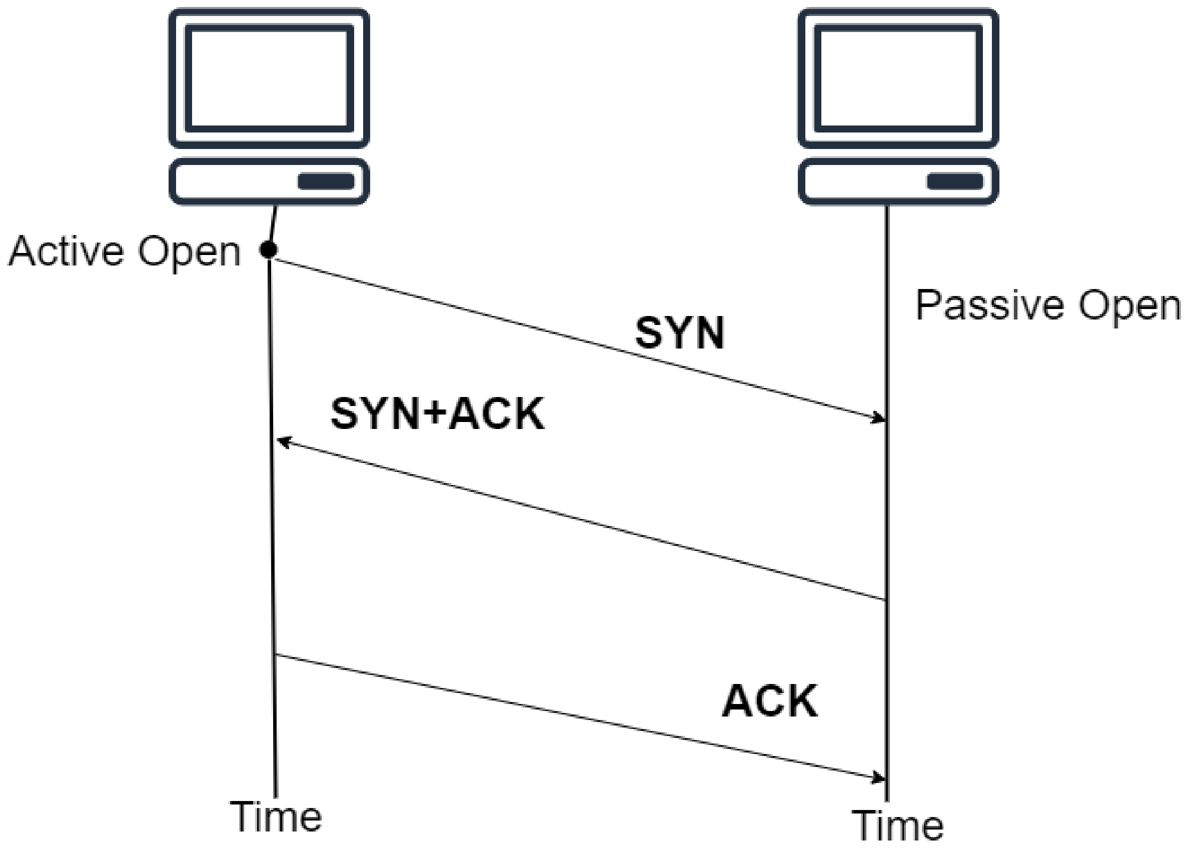


**Figure 3- Connection establishment using three way handshaking.**

**SYN**

- This flag is used for the synchronization of sequence numbers.
- It does not carry any real data.
- It mainly consumes 1 sequence number.

**SYN+ACK**

- This is mainly used for synchronization in other directions and ACK for the signal received.
- It does not carry any real data.

- It also consumes 1 sequence number

**ACK**

- It is just an ACK segment.
- It does not consume any sequence number if it does not carry any data

## 1.5 Data Transfer Phase

After the establishment of the connection, the bidirectional data transfer can take place. Both the client and server can send data and acknowledgments.

## 1.6 Connection Termination Phase

The two parties that are involved in the data exchange can close the connection, although it is initiated usually by the client. There ate two ways for the connection termination.

- Three-way handshaking
- four-way handshaking with a half close option.

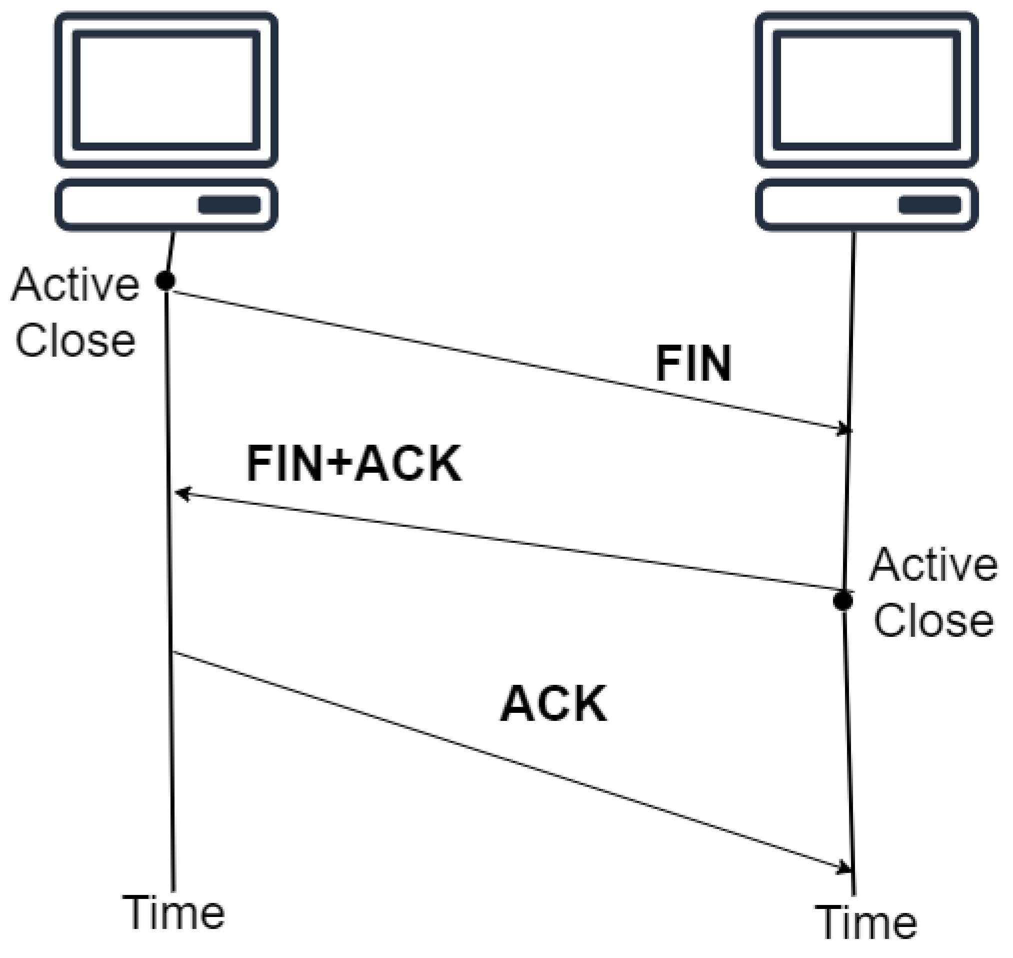


**Fig-4- Connection Termination using three-way handshaking**

**FIN**

- This segment consumes one sequence number if it does not carries any data.
- It may or may not carries any real data.

**FIN+ACK**

- This segment consumes consequence numbers if it does not carries any data.
- The FIN segment announces the closing of the connection in another direction.
- ACK is for received FIN.
- It consumes only 1 sequence number.

**ACK**

- It is just an ACK segment.
- It does not consume any sequence number.

## 1.7 What is ASCII?

ASCII (American Standard Code for Information Interchange) is the most common character encoding format for text data in computers and on the internet. In standard ASCII-encoded data, there are unique values for 128 alphabetic, numeric or special additional characters and control codes.

ASCII encoding is based on character encoding used for telegraph data. The American National Standards Institute first published it as a standard for computing in 1963.

Characters in ASCII encoding include upper- and lowercase letters A through Z, numerals 0 through 9 and basic punctuation symbols. It also uses some non-printing control characters that were originally intended for use with teletype printing terminals

ASCII characters may be represented in the following ways:

- as pairs of hexadecimal digits -- base-16 numbers, represented as 0 through 9 and A through F for the decimal values of 10-15;
- as three-digit octal (base 8) numbers;
- as decimal numbers from 0 to 127; or
- as 7-bit or 8-bit binary

## 1.8 How does ASCII work?

ASCII offers a universally accepted and understood character set for basic data communications. It enables developers to design interfaces that both humans and computers understand. ASCII codes a string of data as ASCII characters that can be interpreted and displayed as readable plain text for people and as data for computers.

Using this encoding, developers can easily convert ASCII digits to numerical values by stripping off the four most significant bits of the binary ASCII values (0011). This calculation can also be done by dropping the first hexadecimal digit or by subtracting 48 from the decimal ASCII code.

Developers can also check the most significant bit of characters in a sequence to verify that a data stream, string or file contains ASCII values. The most significant bit of basic

ASCII characters will always be 0; if that bit is 1, then the character is not an ASCII-encoded character.

## 1.9 ASCII code table

| Dec | Char | Dec | Char | Dec | Char | Dec | Char |
|---|---|---|---|---|---|---|---|
| 0 | NUL (null) | 32 | SPACE | 64 | @ | 96 | ` |
| 1 | SOH (start of heading) | 33 | ! | 65 | A | 97 | a |
| 2 | STX (start of text) | 34 | " | 66 | B | 98 | b |
| 3 | ETX (end of text) | 35 | # | 67 | C | 99 | c |
| 4 | EOT (end of transmission) | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ (enquiry) | 37 | % | 69 | E | 101 | e |
| 6 | ACK (acknowledge) | 38 | & | 70 | F | 102 | f |
| 7 | BEL (bell) | 39 | ' | 71 | G | 103 | g |
| 8 | BS (backspace) | 40 | ( | 72 | H | 104 | h |
| 9 | TAB (horizontal tab) | 41 | ) | 73 | I | 105 | i |
| 10 | LF (NL line feed, new line) | 42 | * | 74 | J | 106 | j |
| 11 | VT (vertical tab) | 43 | + | 75 | K | 107 | k |
| 12 | FF (NP form feed, new page) | 44 | , | 76 | L | 108 | l |
| 13 | CR (carriage return) | 45 | - | 77 | M | 109 | m |
| 14 | SO (shift out) | 46 | . | 78 | N | 110 | n |
| 15 | SI (shift in) | 47 | / | 79 | O | 111 | o |
| 16 | DLE (data link escape) | 48 | 0 | 80 | P | 112 | p |
| 17 | DC1 (device control 1) | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 (device control 2) | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 (device control 3) | 51 | 3 | 83 | S | 115 | s |
| 20 | DC4 (device control 4) | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK (negative acknowledge) | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN (synchronous idle) | 54 | 6 | 86 | V | 118 | v |
| 23 | ETB (end of trans. block) | 55 | 7 | 87 | W | 119 | w |
| 24 | CAN (cancel) | 56 | 8 | 88 | X | 120 | x |
| 25 | EM (end of medium) | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB (substitute) | 58 | : | 90 | Z | 122 | z |
| 27 | ESC (escape) | 59 | ; | 91 | [ | 123 | { |
| 28 | FS (file separator) | 60 | < | 92 | \ | 124 | \| |
| 29 | GS (group separator) | 61 | = | 93 | ] | 125 | } |
| 30 | RS (record separator) | 62 | > | 94 | ^ | 126 | ~ |
| 31 | US (unit separator) | 63 | ? | 95 | _ | 127 | DEL |

## 1.10 What is Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Sockets have their own vocabulary −

| Sr.No. | Term & Description |
|---|---|
| 1 | **Domain**<br>The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on. |
| 2 | **type**<br>The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols. |
| 3 | **protocol**<br>Typically zero, this may be used to identify a variant of a protocol within a domain and type. |
| 4 | **hostname**<br>The identifier of a network interface −<br>• A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation<br>• A string "<broadcast>", which specifies an INADDR_BROADCAST address.<br>• A zero-length string, which specifies INADDR_ANY, or<br>• An Integer, interpreted as a binary address in host byte order. |
| 5 | **port**<br>Each server listens for clients calling on one or more ports. A port may be a Fix num port number, a string containing a port number, or the name of a service. |

## The socket Module

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax –

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters –

- **socket_family** − This is either AF_UNIX or AF_INET, as explained earlier.
- **socket_type** − This is either SOCK_STREAM or SOCK_DGRAM.
- **protocol** − This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required −

## Server Socket Methods

| Sr.No. | Method & Description |
|---|---|
| 1 | **s.bind()** This method binds address (hostname, port number pair) to socket. |
| 2 | **s.listen()** This method sets up and start TCP listener. |
| 3 | **s.accept()** This passively accept TCP client connection, waiting until connection arrives (blocking). |

## Client Socket Methods

| Sr.No. | Method & Description |
|---|---|
| 1 | **s.connect()** |

| | |
|---|---|
| | This method actively initiates TCP server connection. |

## General Socket Methods

| Sr.No. | Method & Description |
|---|---|
| 1 | **s.recv()**<br>This method receives TCP message |
| 2 | **s.send()**<br>This method transmits TCP message |
| 3 | **s.recvfrom()**<br>This method receives UDP message |
| 4 | **s.sendto()**<br>This method transmits UDP message |
| 5 | **s.close()**<br>This method closes socket |
| 6 | **socket.gethostname()**<br>Returns the hostname. |

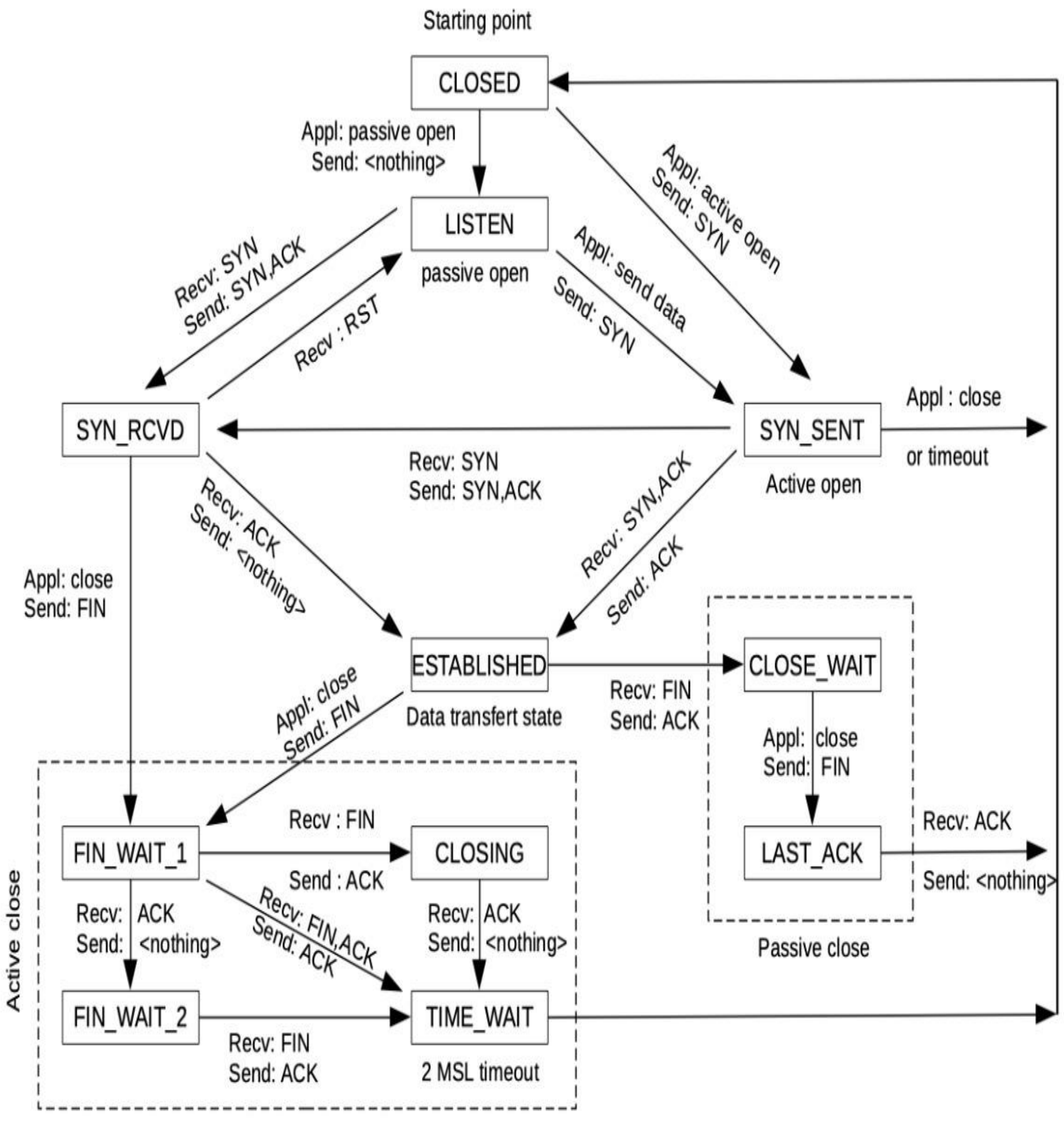


Fig-5- Different TCP connection states

# 2. SYSTEM REQUIREMENTS

## Hardware Requirements:

- 2 GHz x86 processor
- 256 MB of system memory (RAM)
- 100 MB of hard-drive space
- Monitor to display output
- Keyboard/Mouse for data input

## Software Requirements:

- VS code, Windows power Shell, Jupyter notebook for Coding
- MS Word(Documentation)

## Language used:

- Python for both client and server.

# 3. FLOWCHART / DATA FLOW DIAGRAM

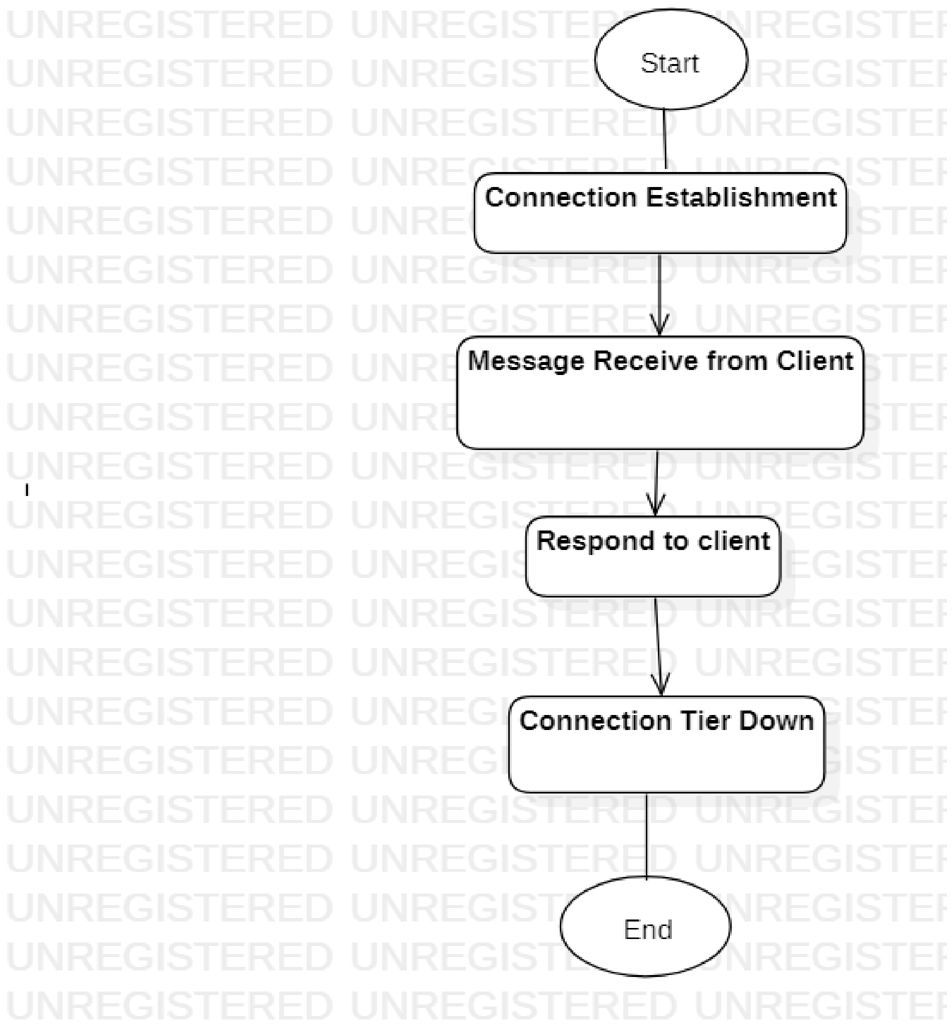


**Figure 5- Flowchart of Server Side**

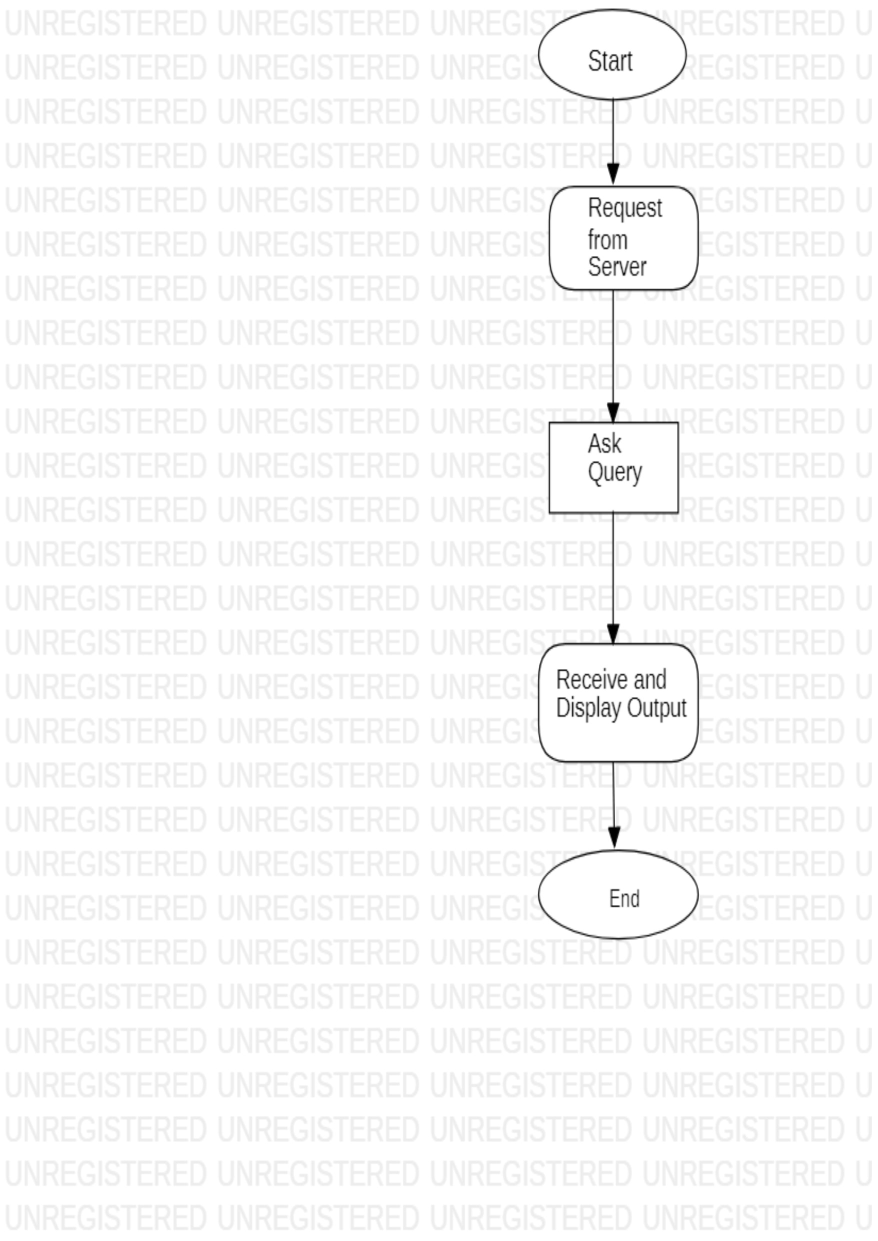


**Figure 6- Flowchart of Client side**

# 4. CODE

## 4.1 Code of  Server Side for ASCII to Character

```
# server converting ASCII code to CHARACTER

from codecs import encode
import socket
import sys

# s = socket.socket (socket_family, socket_type, protocol=0)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print(s)
print("socket created")

# s.bind method binds address (hostname, port number pair) to
socket.
s.bind(('localhost',9999))

# s.listen method sets up and start TCP listening.
s.listen(3)
print("waiting for connections")

while True:
    # s.accept() will passively accept TCP client connection,
    # waiting until connection arrives (blocking).
    c,add = s.accept()
    print('connected with ',add)

    c.send('hello client ok '.encode())   #Transmiting TCP message
    cdata1 = c.recv(1024).decode()        #To receives TCP message
    cdata1 = int(cdata1)

    # Python chr() function is used to get a string representing of
a character
    # which points to a Unicode code integer.
    numb = chr(cdata1)

    # numb = str(numb) #Converting number to character
    c.send(numb.encode())

    c.close                         #This method closes socket
```

## 4.2 Code of Client Side for ASCII to Character

```
#client reqest to convert ASCII CODE to CHARACTER


import socket


# s = socket.socket (socket_family, socket_type,
protocol=0)
c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('client socket created')

# c.connect method actively initiates TCP server
connection.
c.connect(('localhost',9999))

# s.recv() method receives TCP message
print(c.recv(1024).decode())

# Taking input from client
tm1 = input('enter number')

#Sendall() method transmits TCP message
c.sendall(tm1.encode())

# Client wait to get response from server
print('wait getting ascii char.....')

# client is now ready print character assembled from server
print(c.recv(1024).decode())

#c.close() closes the socket
c.close()
```

## 4.3 Code of server side for character to ASCII

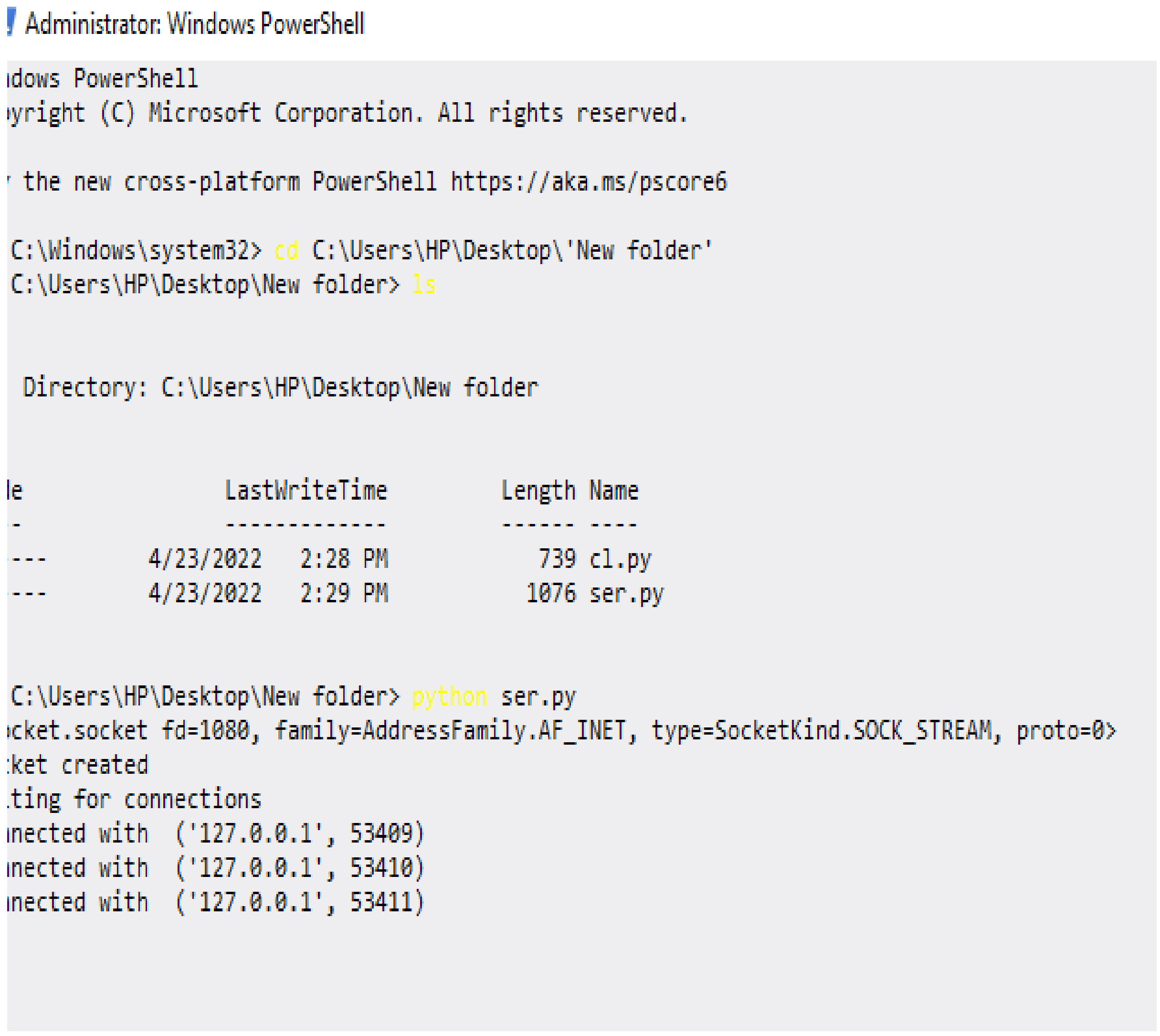
```
# server converting CHARACTER to ASCII CODE

from codecs import encode
import socket
import sys

# s = socket.socket (socket_family, socket_type,
protocol=0)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print(s)
print("socket created")

# s.bind method binds address (hostname, port number pair)
to socket.
s.bind(('localhost',9999))

# s.listen method sets up and start TCP listener.
s.listen(3)
print("waiting for connections")

while True:
    # s.accept() will passively accept TCP client
connection,
    # waiting until connection arrives (blocking).
    c,add = s.accept()
    print('connected with ',add)
    c.send('hello client ok '.encode()) #Transmiting TCP
message

    cdata1 = c.recv(1024).decode() #To receives TCP
message

    aski = ord(cdata1) #Python ord() function returns the
Unicode code from a given character.

    aski = str(aski) #Converting number to character
    c.send(aski.encode())
    c.close
```

## 4.4 Code for Client Side for character to ASCII

```
#client reqest to convert CHARACTER to ASCII CODE
import socket

# s = socket.socket (socket_family, socket_type,
protocol=0)
c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('client socket created')

# c.connect method actively initiates TCP server
connection.
c.connect(('localhost',9999))

# s.recv() method receives TCP message
print(c.recv(1024).decode())

# Taking input from client
tm1 = input('enter character')

#Sendall() method transmits TCP message
c.sendall(tm1.encode())

# Client wait to get response from server
print('wait getting ascii.....')

# client is now ready print character assembled from server
print(c.recv(1024).decode())

#c.close() closes the socket
c.close()
```

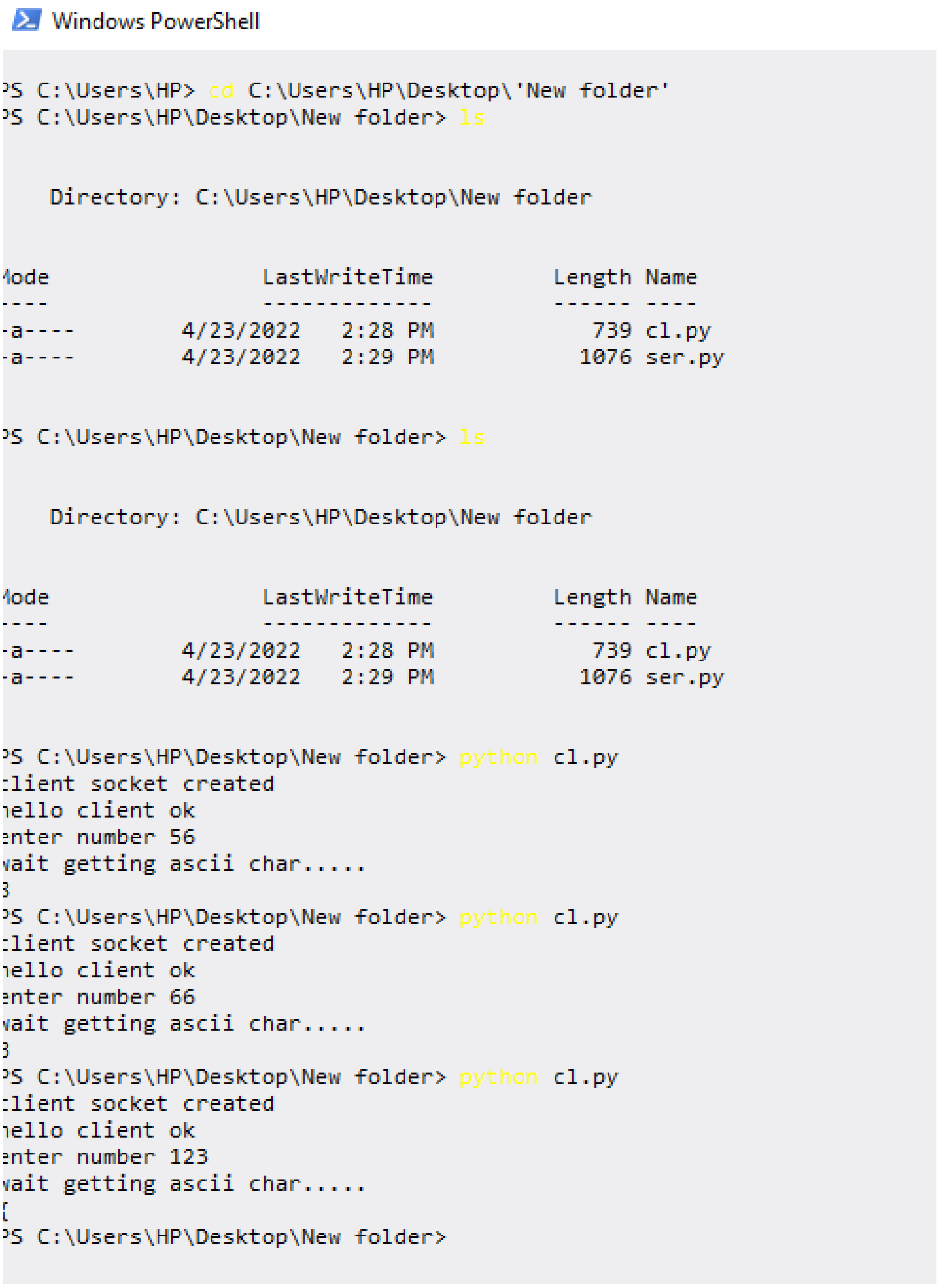
# 5.Output

## 5.1 Output  for  ASCII to Character



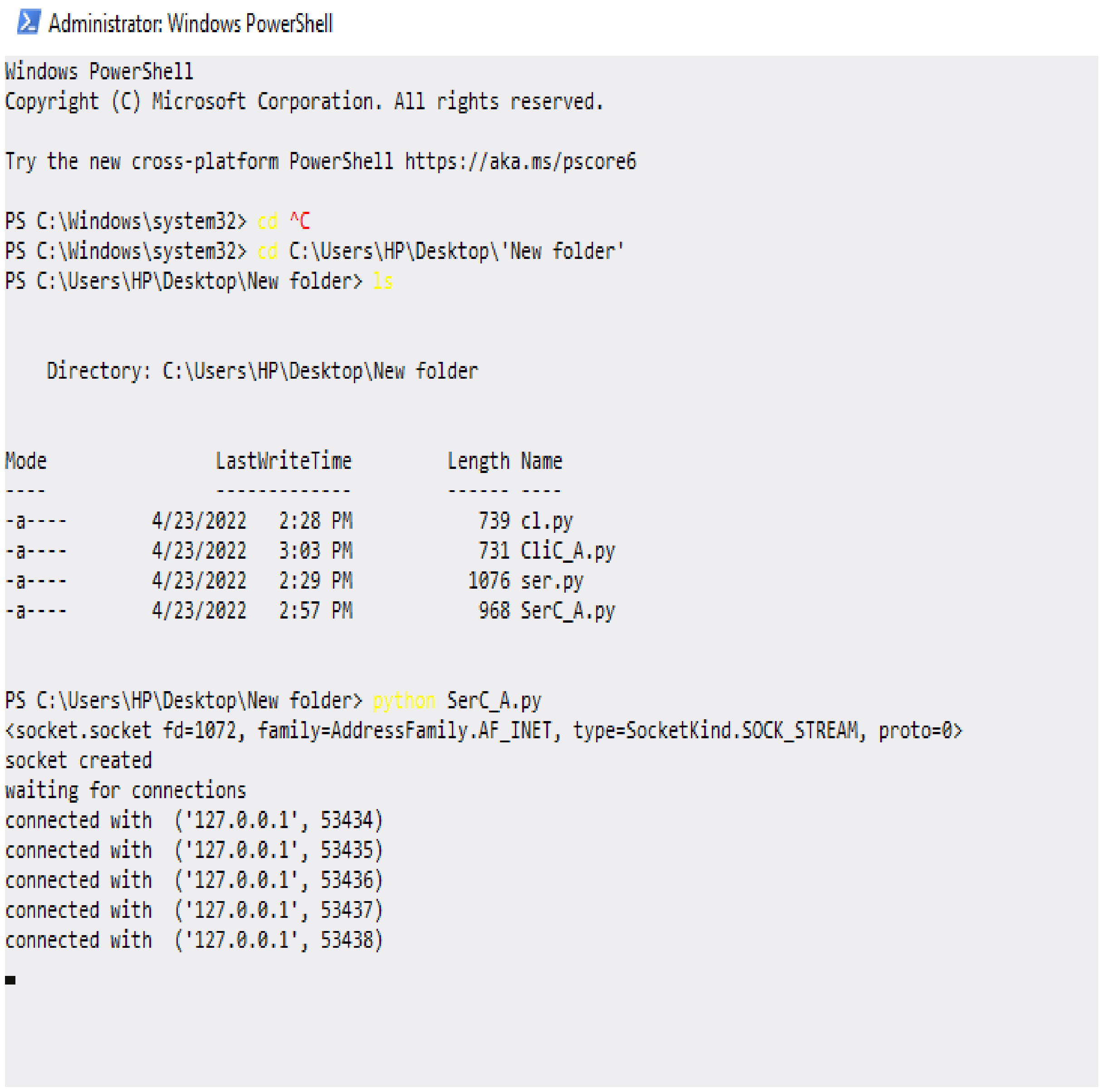
Administrator: Windows PowerShell

```
ıdows PowerShell
ıyright (C) Microsoft Corporation. All rights reserved.

' the new cross-platform PowerShell https://aka.ms/pscore6

 C:\Windows\system32> cd C:\Users\HP\Desktop\'New folder'
 C:\Users\HP\Desktop\New folder> ls


  Directory: C:\Users\HP\Desktop\New folder


le                 LastWriteTime         Length Name
-                  -------------         ------ ----
 ---        4/23/2022   2:28 PM             739 cl.py
 ---        4/23/2022   2:29 PM            1076 ser.py


 C:\Users\HP\Desktop\New folder> python ser.py
ıcket.socket fd=1080, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0>
:ket created
.ting for connections
ınected with  ('127.0.0.1', 53409)
ınected with  ('127.0.0.1', 53410)
ınected with  ('127.0.0.1', 53411)
```


Windows PowerShell

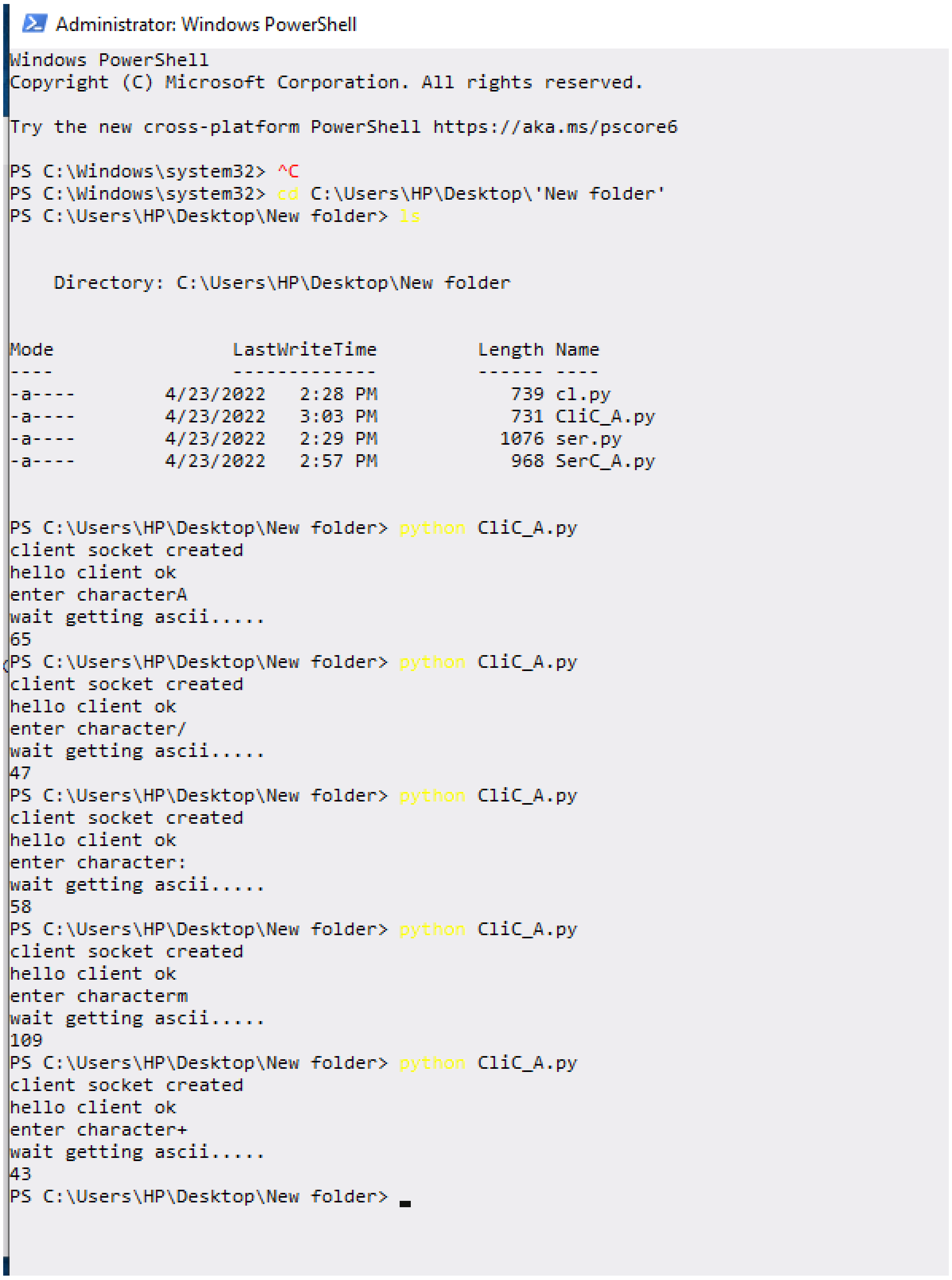
```
PS C:\Users\HP> cd C:\Users\HP\Desktop\'New folder'
PS C:\Users\HP\Desktop\New folder> ls


    Directory: C:\Users\HP\Desktop\New folder


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/23/2022   2:28 PM            739 cl.py
-a----         4/23/2022   2:29 PM           1076 ser.py


PS C:\Users\HP\Desktop\New folder> ls


    Directory: C:\Users\HP\Desktop\New folder


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/23/2022   2:28 PM            739 cl.py
-a----         4/23/2022   2:29 PM           1076 ser.py


PS C:\Users\HP\Desktop\New folder> python cl.py
:lient socket created
nello client ok
enter number 56
vait getting ascii char.....
3
PS C:\Users\HP\Desktop\New folder> python cl.py
:lient socket created
nello client ok
enter number 66
vait getting ascii char.....
3
PS C:\Users\HP\Desktop\New folder> python cl.py
:lient socket created
nello client ok
enter number 123
vait getting ascii char.....
[
PS C:\Users\HP\Desktop\New folder>
```

## 5.2 Output for Character to ASCII



Administrator: Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> cd ^C
PS C:\Windows\system32> cd C:\Users\HP\Desktop\'New folder'
PS C:\Users\HP\Desktop\New folder> ls


    Directory: C:\Users\HP\Desktop\New folder


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        4/23/2022   2:28 PM            739 cl.py
-a----        4/23/2022   3:03 PM            731 CliC_A.py
-a----        4/23/2022   2:29 PM           1076 ser.py
-a----        4/23/2022   2:57 PM            968 SerC_A.py


PS C:\Users\HP\Desktop\New folder> python SerC_A.py
<socket.socket fd=1072, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0>
socket created
waiting for connections
connected with  ('127.0.0.1', 53434)
connected with  ('127.0.0.1', 53435)
connected with  ('127.0.0.1', 53436)
connected with  ('127.0.0.1', 53437)
connected with  ('127.0.0.1', 53438)
```


Administrator: Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> ^C
PS C:\Windows\system32> cd C:\Users\HP\Desktop\'New folder'
PS C:\Users\HP\Desktop\New folder> ls


    Directory: C:\Users\HP\Desktop\New folder


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        4/23/2022   2:28 PM             739 cl.py
-a----        4/23/2022   3:03 PM             731 CliC_A.py
-a----        4/23/2022   2:29 PM            1076 ser.py
-a----        4/23/2022   2:57 PM             968 SerC_A.py


PS C:\Users\HP\Desktop\New folder> python CliC_A.py
client socket created
hello client ok
enter characterA
wait getting ascii.....
65
PS C:\Users\HP\Desktop\New folder> python CliC_A.py
client socket created
hello client ok
enter character/
wait getting ascii.....
47
PS C:\Users\HP\Desktop\New folder> python CliC_A.py
client socket created
hello client ok
enter character:
wait getting ascii.....
58
PS C:\Users\HP\Desktop\New folder> python CliC_A.py
client socket created
hello client ok
enter characterm
wait getting ascii.....
109
PS C:\Users\HP\Desktop\New folder> python CliC_A.py
client socket created
hello client ok
enter character+
wait getting ascii.....
43
PS C:\Users\HP\Desktop\New folder>
```

# 6. OBSERVATIONS

## 6.1 Socket Programming

- Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

- **TCP:** The most obvious technique for achieving broadcast is a flooding. The source node send copy of the packet to all of its neighbours. When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its neighbours (except the neighbour from which it received the packet). This scheme eventually delivers a copy of the broadcast packet to all nodes if they are connected.
- A TCP connection normally consists of three phases: connection establishment, data transfer, and connection termination.

## 6.2 TCP Vs UDP

- TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol. A key difference between TCP and UDP is speed, as TCP is comparatively slower than UDP. Overall, UDP is a much faster, simpler, and efficient protocol, however, retransmission of lost data packets is only possible with TCP.

## 6.3 Involvement of Various Layer

**From Code we observe that:**

- The physical layer, describes specifies the characteristics of the hardware to be used for the network.
- Internet layer, delivers packets from client to server and vice versa.This layer includes the powerful Internet Protocol (IP) , in Socket library we included Ip addresses.
- Transport Layer protocols ensure that packets arrive in sequence and without error, by swapping acknowledgments of data reception, and retransmitting lost packets.
- Application Layer the output is displaying in the screen in understandable form and enable to see the communication between client and server.

# 7. CONCLUSION

This mini project details the great potential that **socket programming** has. We were able to implement concept of **client and server using TCP** through programme. Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP handles reliability and congestion control. It also does error checking and error recovery. The entire process can be broken down into following steps:

**TCP Server –**

- using create(), Create TCP socket.
- using bind(), Bind the socket to server address.
- using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
- using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
- Go back to Step 3.

**TCP Client –**

- Create TCP socket.
- connect newly created client socket to server.

Through this mini project we were able to understand how to use these function for establish connection between client and user and successfully we did it and create application to convert ASCII to Binary and vice-versa. Even we are in condition to implement these concept of socket programming different application.

# 8. LEARNING OUTCOME

Following are the learning outcomes of mine by this mini project that is "a Client Server based Character to ASCII and vice-versa code conversion application using TCP":

- After this experiment, we are familiar with the working of socket programming.
- We are also able to write program using TCP and UDP protocol.
- We are able to solve numerical related to these algorithms.
- After this experiment, we are able to classify difference between TCP and UDP concept and their different programming concept.
- We are able to understand different ASCI value of Different character.
- After this experiment, we can apply my learnings when we want to develop such program related to socket programming.
- Having a solid understanding of different protocol used in Transport layer and difference between them.
- Good knowledge of ASCI value of different character especially special symbol in keyboard.

## 9. REFERENCES

[1] Tanenbaum S. Andrew and David J. Wetherall, Computer Networks(Vol. 5), Pearson Education Inc., 2011.

[2] Forouzan A. Behrouz, Data Communications And Networking(Vol.4), McGraw-HilIForouzan networking series, 2007.

[3] S.E. Deering and D.R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," ACM Transactions on Computer Systems, vol. 8, no. 2, May 1990, pp. 85-110.

[4] LimiKalita, "Socket Programming", Networks, vol. 5(3), pp. 4802-4807.

[5] http://en.wikipedia.org/wiki/Berkeley_sockets.

[6] Joseph M. Dibella , “Socket Programming with Java”.

[7] https://www.tutorialspoint.com/java/java_networking.htm

[8] Sarvesh Kumar, “overview in developing socket programming”, Sept. 1997.

[9] Rajkumar Buyya, “Socket Programming”.

[10] https://www.geeksforgeeks.org/socket-programming-python/

[11]https://www.studytonight.com/computer-networks/tcp-transmission-control-protocol

.