



AY : 2023_24
CLASS : BE E&TC
SUBJECT : Mobile Computing

Sem. : II
DATE :
EXPT. No. : 07

TITLE: FILE TRANSFER USING TCP

OBJECTIVE: Perform a file transfer operation for server-client model using TCP/IP

SOFTWARE USED: Open-source Linux operating system
Software: Python

THEORY:

Socket

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, etc. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as `read()` and `write()` work with sockets in the same way they do with files and pipes.

Types of Socket

A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data. There are four types of sockets available to the users. Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types.

Stream Sockets – Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order – "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

Datagram Sockets – Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets – you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

Raw Sockets – these provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact



characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

Sequenced Packet Sockets – they are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets, either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.

Sequenced Packet Sockets –

The client-server model is one of the most commonly used communication paradigms in networked systems. Clients normally communicate with one server at a time. From a server's perspective, at any point in time, it is not unusual for a server to be communicating with multiple clients. Client need to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. The client and the server on the same local network (usually called LAN, Local Area Network), the client and the server may be in different LANs, with both LANs connected to a Wide Area Network (WAN) by means of *routers*

Transmission Control Protocol (TCP)

TCP provides a *connection oriented service*, since it is based on connections between clients and servers.

TCP provides reliability. When a TCP client sends data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically retransmit the data and waits for a longer period of time for acknowledgement. The sequence of function calls for the client and a server participating in a TCP connection is presented in following Figure.

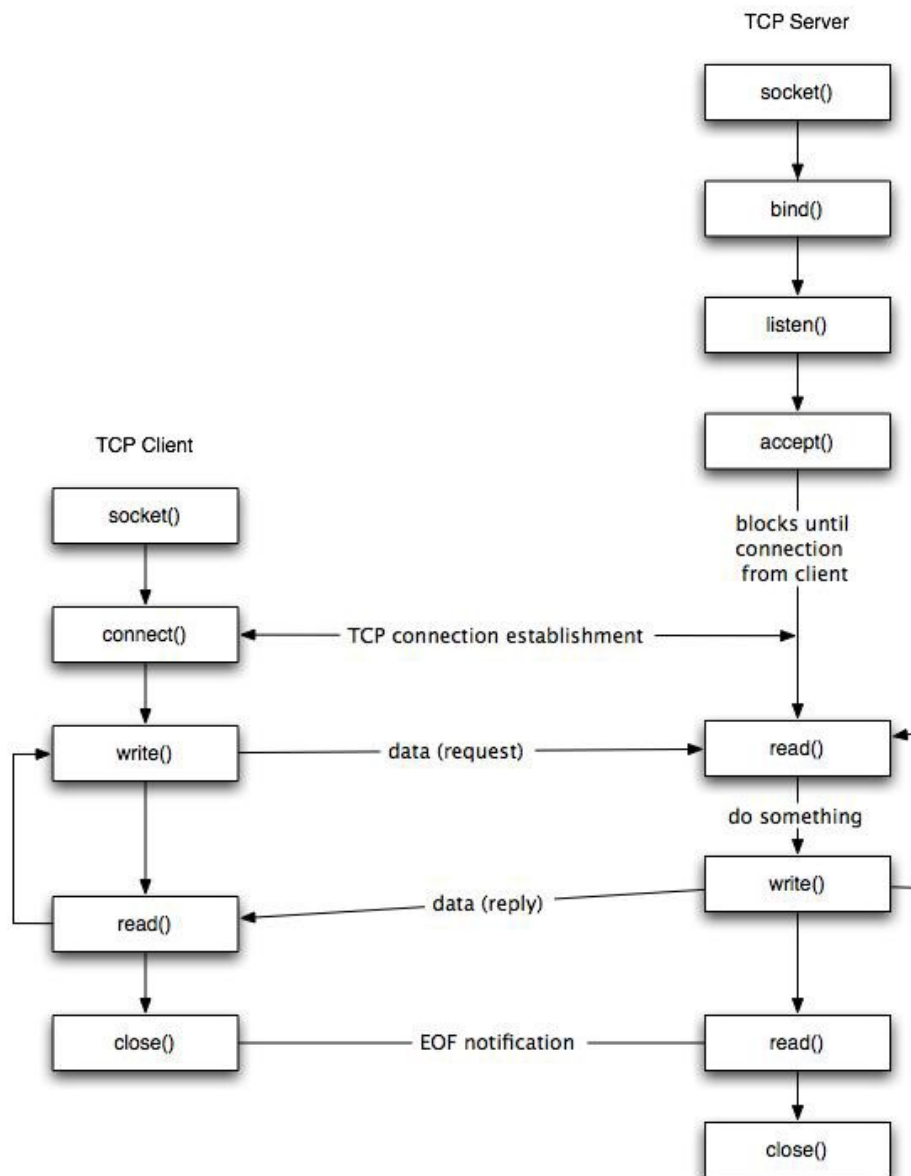
As shown in the Figure, the steps for establishing a TCP socket on the client side are as follows:

- Create a socket using the `socket()` function;
- Connect the socket to the address of the server using the `connect()` function;
- Send and receive data by means of the `read()` and `write()` functions.
- Close the connection by means of the `close()` function.

The steps involved in establishing a TCP socket on the server side are as follows:

- Create a socket with the `socket()` function;

- Bind the socket to an address using the bind() function;
- Listen for connections with the listen() function;
- Accept a connection with the accept() function system call. This call typically blocks until a client connects with the server.
- Send and receive data by means of send() and receive().
- Close the connection by means of the close() function.

**CODE:****Client code in python –**

```
import socket
Server_ip="localhost"
Server_host=8002
FORMAT="utf-8"
```



```
CS=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
CS.connect((Server_ip, Server_host))
file = open("data/abc.txt","r")
data=file.read()
CS.send("abc.txt".encode(FORMAT))
msg=CS.recv(1024)
print( msg)
CS.send(data.encode(FORMAT))
msg=CS.recv(1024)
print(msg)
```

Server code in python -

```
import socket
Server_ip="localhost"
Server_host=8002
FORMAT="utf-8"
SS=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
SS.bind((Server_ip,Server_host))
SS.listen(5)
s1, addr=SS.accept()
file_name= s1.recv(1024).decode(FORMAT)
print(file_name)
file=open(file_name,"w")
s1.send("File name received")
data=s1.recv(1024).decode(FORMAT)
print("File data received")
s1.send("File data received")
file.write(data)
file.close()
```

CONCLUSION:

SIGNATURE

REFERENCES:

1. "Mobile Communications" – Jochen Schiller