

CYCLE 3

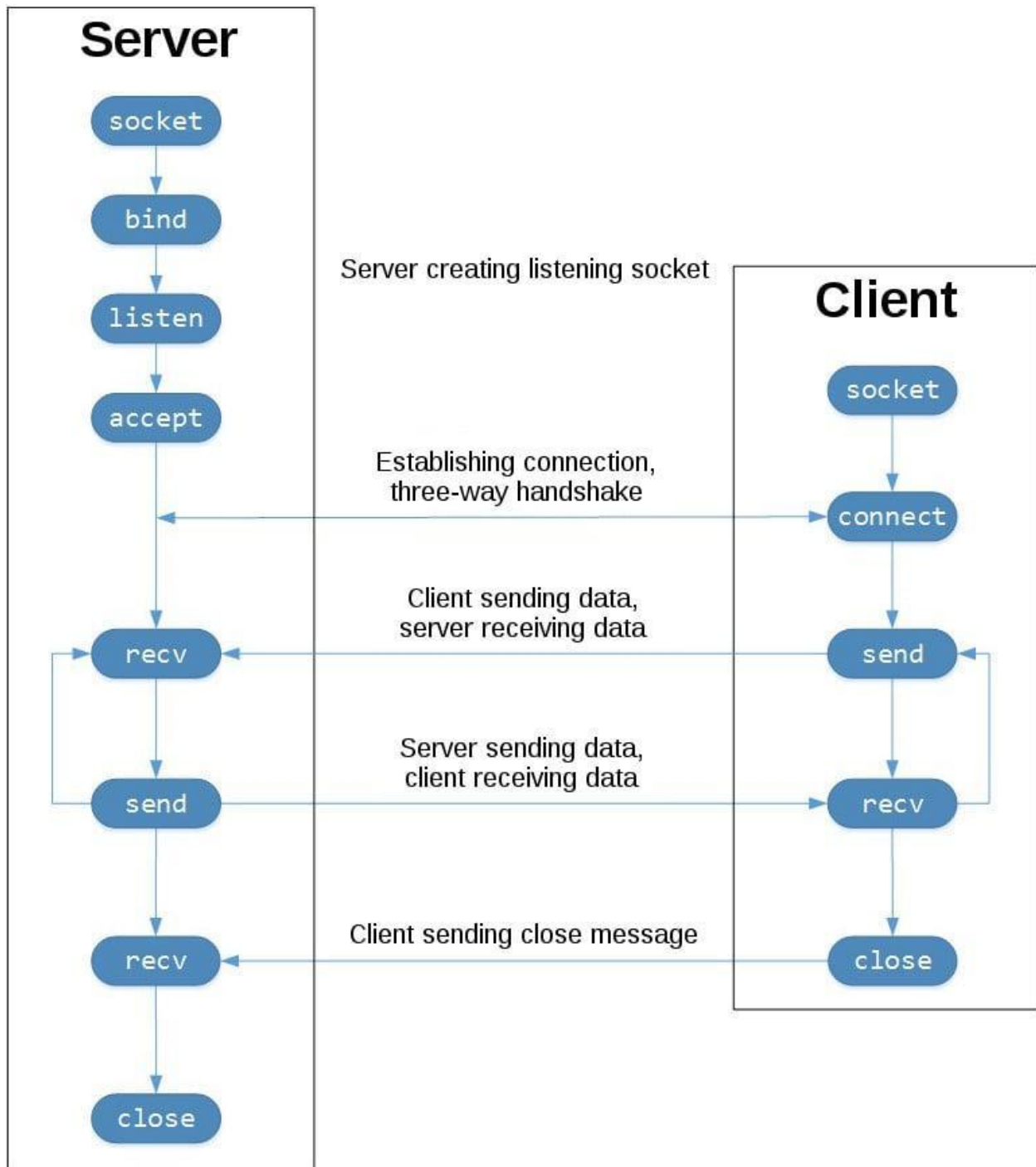
1. Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

2. Server :

3. A server has a bind() method which binds it to a specific IP and port so that it can listen to incoming requests on that IP and port. A server has a listen() method which puts the server into listening mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.
4. First of all, we import socket which is necessary.
5. Then we made a socket object and reserved a port on our pc.
6. After that, we bound our server to the specified port. Passing an empty string means that the server can listen to incoming connections from other computers as well. If we would have passed 127.0.0.1 then it would have listened to only those calls made within the local computer.
7. After that we put the server into listening mode. 5 here means that 5 connections are kept waiting if the server is busy and if a 6th socket tries to connect then the connection is refused.
8. At last, we make a while loop and start to accept all incoming connections and close those connections after a thank you message to all connected sockets.

This output shows that our server is working.
Now for the client-side:

- First of all, we make a socket object.
- Then we connect to localhost on port 12345 (the port on which our server runs) and lastly, we receive data from the server and close the connection.
- Now save this file as client.py and run it from the terminal after starting the server script.



The arguments passed to `socket()` are constants used to specify the address family and socket type. `AF_INET` is the Internet address family for IPv4. `SOCK_STREAM` is the socket type for TCP, the protocol that will be used to transport messages in the network. The IP address `127.0.0.1` is the standard IPv4 address for the loopback interface, so only processes on the host will be able to connect to the server. `port` represents the TCP port number to accept connections on from clients. It should be an integer from 1 to 65535, as 0 is reserved.

The `.bind()` method is used to associate the socket with a specific network interface and port number

The `listen()` method has a backlog parameter. It specifies the number of unaccepted connections that the system will allow before refusing new connections. If your server receives a lot of connection requests simultaneously, increasing the backlog value may help by setting the maximum length of the queue for pending connections.

The `accept()` method blocks execution and waits for an incoming connection. When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client. The tuple will contain (host, port) for IPv4 connections

SOLUTION:

ClientTCP.py

```
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = input("\nEnter file name: ")
```

```
clientSocket.send(sentence.encode())
filecontents = clientSocket.recv(1024).decode()
print ("\nFrom Server:\n")
print(filecontents)
clientSocket.close()
```

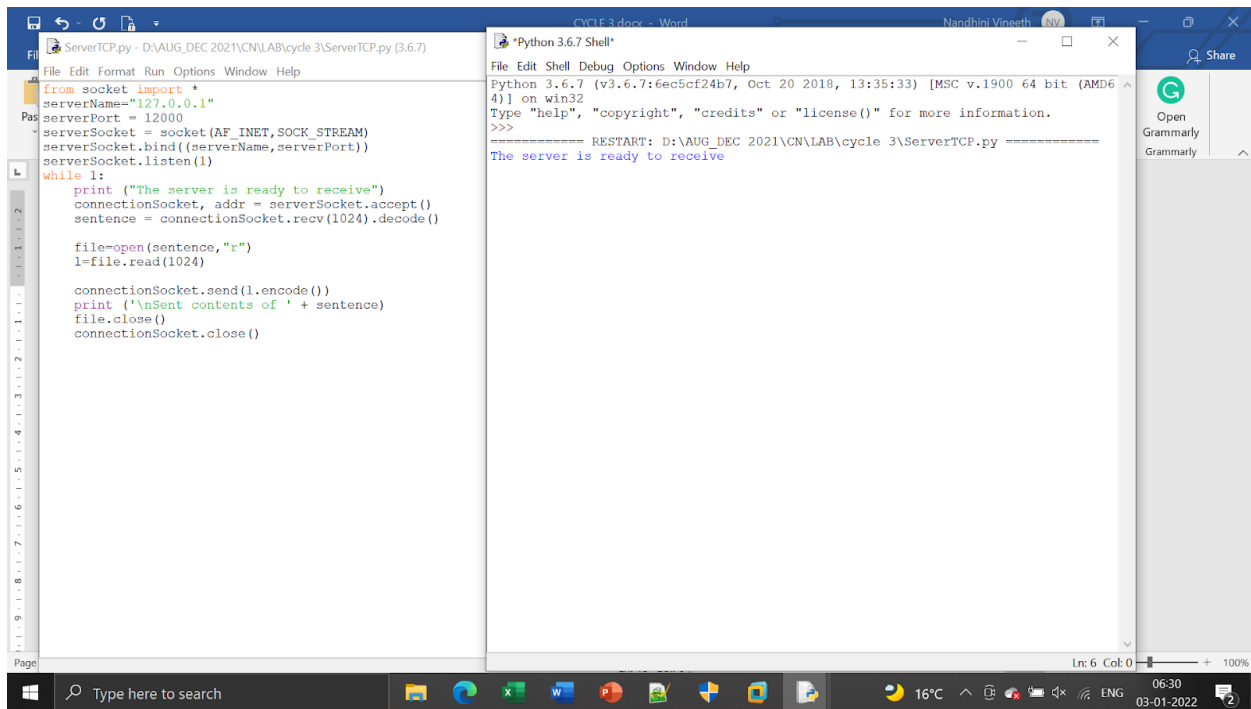
ServerTCP.py

```
from socket import *
serverName="127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((serverName, serverPort))
serverSocket.listen(1)
while 1:
    print ("The server is ready to receive")
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()

    file=open(sentence, "r")
    l=file.read(1024)

    connectionSocket.send(l.encode())
    print ("\nSent contents of " + sentence)
    file.close()
    connectionSocket.close()
```

OUTPUT:



The screenshot displays a Windows desktop environment. On the left, a text editor window titled 'ServerTCP.py - D:\AUG_DEC 2021\CN\LAB\cycle 3\ServerTCP.py (3.6.7)' contains the following Python code:

```
from socket import *
serverName="127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((serverName,serverPort))
serverSocket.listen(1)
while 1:
    print("The server is ready to receive")
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()

    file=open(sentence,"r")
    l=file.read(1024)

    connectionSocket.send(l.encode())
    print('\nSent contents of ' + sentence)
    file.close()
    connectionSocket.close()
```

On the right, a Python 3.6.7 Shell window titled '*Python 3.6.7 Shell*' shows the execution output:

```
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\AUG_DEC 2021\CN\LAB\cycle 3\ServerTCP.py =====
The server is ready to receive
```

The taskbar at the bottom shows the Windows Start button, a search bar, and several application icons. The system tray on the right indicates a temperature of 16°C, the time 06:30, and the date 03-01-2022.

OBSERVATION :

Program - 3

Using TCP/IP sockets, write a client-server program to make client sending the file name & the server to send back the contents of the requested file if present.

Solution :-

```
clientTCP.py
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = input('Enter file name: ')
clientSocket.send(sentence.encode())
fileContents = clientSocket.recv(1024).decode()
print('In From server: \n')
print(fileContents)
clientSocket.close()
```

serverTCP.py

```
from socket import *
serverName = "127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((serverName, serverPort))
```

output:-

First run serverUDP.py then run clientUDP.py

output on serverUDP.py

the server is ready to receive

Send contents of serverUDP.py

output on clientUDP.py

Enter file name: serverUDP.py

Reply from server:

contents of serverUDP.py is displayed
here same as serverUDP.py file.

28/9

1/9/2022

serverUDP.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("127.0.0.1", serverPort))
print("The server is ready to receive")
while 1:
    sentence, clientAddress = serverSocket.recvfrom(2048)
    sentence = sentence.decode("utf-8")
    file = open(sentence, "x")
    con = file.read(2048)
    serverSocket.sendto(bytes(con, "utf-8"), clientAddress)

    print("\n Send contents of: ", end = "")
    print(sentence)
    # for i in sentence:
    #     print(chr(i), end = ' ')
    file.close()
```

9) Using UDP sockets, write a client-server program to make client sending the file name to the server to send back the contents of the requested file if present.

clientUDP.py

```
from socket import *
ServerName = "127.0.0.1"
ServerPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
sentence = input("In Enter file name: ")
clientSocket.sendto(bytes(sentence, 'utf-8'), (ServerName, serverPort))
fileContents, serverAddress = clientSocket.recvfrom(2048)
print('In Reply from server: \n')
print(fileContents, decode("utf-8"))
# for i in fileContents:
# print(str(i), end=" ")
clientSocket.close()
clientSocket.close()
```



```

serverSocket.listen(1)
while 1:
    print("the server is ready to receive")
    connectionSocket, address = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    file = open(sentence, "r")
    l = file.read(1024)
    connectionSocket.send(l.encode())
    print('In Send contents of: ' + sentence)
    file.close()
    connectionSocket.close()

```

output:

First Run serverTCP.py then run clientTCP.py
 output on serverTCP

the server is ready to receive
 send contents of serverTCP.py
 the server is ready to receive
 output on clientTCP terminal.

Enter file name: serverTCP.py

from server:
 contents in serverTCP.py is displayed
 here