Name: Muhammad Rizwan Khalid **Regn. No.: 180459**

Name: Syed Abbas Hussain Naqvi **Regn. No.: 177410**

Lab Title: Network Programming

1.0 Objectives:

After this lab, the students should be able to

- Use the sockets interface of Python programming language
- Implement simple Client / Server applications using UDP and TCP
- File transfer trough udp socket
- Mp4 video transfer using TCP socket

2.0 Instructions:

- *Read carefully before starting the lab.*
- *These exercises are to be done individually.*
- *To obtain credit for this lab, you are supposed to complete the lab tasks and provide the source codes and the screen shot of your output in this document (please use red font color) and upload the completed document to your course's LMS site.*
- *Avoid plagiarism by copying from the Internet or from your peers. You may refer to source/ text but you must paraphrase the original work.*

3.0 Background:

4.0 Client-Server Socket Programming

We introduce UDP and TCP socket programming by way of a simple UDP application and a simple TCP application both implemented in Python.

We'll use the following simple 'Echo' client-server application to demonstrate socket programming for both UDP and TCP:

1. The client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts the characters to uppercase.

3. The server sends the mod+ified data to the client.

4. The client receives the modified data and displays the line on its screen.

4.1 Socket programming using UDP

.

Now let's take a look at the client-server program pair for a UDP implementation of this simple application. We'll begin with the UDP client, which will send a simple application-level message to the server. In order for the server to be able to receive and reply to the client's message, it must be ready and running— that is, it must be running as a process before the client sends its message.

The client program is called UDPClient.py, and the server program is called UDPServer.py. In order to emphasize the key issues, we intentionally provide code that is minimal. "Good code" would certainly have a few more auxiliary lines, in particular for handling error cases. For this application, we have arbitrarily chosen 12000 for the server port number.

UDPClient.py

```
=============================================================================

from socket import * # import socket module

  serverIP = 'hostname'  # replace with IP address of the server

serverPort = 25000  #port where server is listening

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = raw_input('Input lowercase sentence:')

clientSocket.sendto(message,(serverIP, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

print modifiedMessage # print the received message

clientSocket.close() # Close the socket

=============================================================================
```

UDPServer.py

```
=============================================================

from socket import *

serverPort = 25000
```

```
serverIP="10.99.26.161"
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind((serverIP, serverPort))
print ”The server is ready to receive”
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

==========

To test the pair of programs, you install and run UDPClient.py in one host and UDPServer.py in another host. Be sure to include the proper hostname or IP address of the server in UDPClient.py. Next, you execute UDPServer.py, the server program, in the server host. This creates a process in the server that idles until some client contacts it. Then you execute UDPClient.py, the client program, in the client. This creates a process in the client. Finally, to use the application at the client, you type a sentence followed by a carriage return.

4.2 Socket programming using TCP

We use the same simple client-server application to demonstrate socket programming with TCP: The client sends one line of data to the server, the server capitalizes the line and sends it back to the client. Figure 2 highlights the main socket-related activity of the client and server that communicate over the TCP transport service.

TCPClient.py

==========

```
from socket import *
serverIP = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.connect((serverIP,serverPort))
```

clientSocket.send(sentence)

modifiedSentence = clientSocket.recv(1024)

print 'From Server:', modifiedSentence

clientSocket.close()

==========

TCPServer.py

==========

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((' ',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
        connectionSocket, addr = serverSocket.accept()
        sentence = connectionSocket.recv(1024)
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence)
        connectionSocket.close()
```

5.1 File I/O

Opening and closing file:
Python provides basic functions and methods necessary to manipulate files by default. File manipulation can be done using a file object.

The Open function: Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object that is utilized to call other support methods associated with it. The syntax is as follows:

```
file object = open(file_name [, access_mode][, buffering])
```

- ***file_name:*** The file_name argument is a string value that contains the name of the file that you want to access.
- ***access_mode:*** The access_mode determines the mode in which the file has to be opened ie. read, write, append etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r)

- ***buffering:*** If the buffering value is set to 0, no buffering will take place. If the buffering value is 1, line buffering will be performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action will be performed with the indicated buffer size.

Following is the list of different modes of opening file.

Modes	**Description**
R	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
Rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file.
W	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
Wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
A	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
Ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file object attributes: Once a file is opened and you have one file object, you can get information related to that file. Here is a list of all attributes related to file object:

Attributes	**Description**
file.closed	Returns true if file is closed, false otherwise

file.mode	Returns access mode with which file was opened
file.name	Returns name of the file.

The close() Method: The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file. The syntax is as follows:

```
fileObject.close();
```

Reading and writing files

The file object provides a set of access methods for reading or writing to a file.

The write method: The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text. The write() method does not add a newline character ('\n') to the end of the string. The syntax is as follows:

```
fileObject.write(string);
```

The read method: The read() method read a string from an open file. It is important to note that Python strings can have binary data and not just text. The syntax is as follows:

```
fileObject.read([count]);
```

Here passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing then it tries to read as much as possible.

Lab Task 1

Write UDP client server code to transfer a txt file from client to server.

- Steps of sender side
- Import relevant library
- Set server ip and prot
- Set buffer length 1024
- Read file name e,g filename= c:\python27\myfile.txt
- Create udp socket
- Open txt file e.g f = open(file_name, "r")
- Read data from file e.g data = f.read(bufferlenth)
- Send dat through socket e.g sock.sendto(data, (UDP_IP, UDP_PORT))

- Do previous 2 steps while there is data remaining to send and Give receiver a bit time to save data e.g time.sleep(0.02)
- Close file and socket

Step for receiver side

- Import relevant library
- Set server ip and prot
- Create udp socket
- bind ip and port to socket
- In while(1) loop read data using data, addr = sock.recvfrom(1024)
- Open file for writing e.g f = open(file_name, 'wb')
- Write data to file e.g f.write(data)
- If no more data break and close file (use select() funtion)

Lab Task 2

Write TCP client server code to transfer MP4 file from client to server.

Task: 01:

Sender Code:

```
from socket import *
import sys
serverSocket = socket(AF_INET,SOCK_DGRAM)
file_name = raw_input("Enter Filename: ")
serverSocket.sendto(file_name,('10.3.93.28',12000))
f=open(file_name,"rb")
data = f.read(1024)
```

```
while (data):
    if(serverSocket.sendto(data,('10.3.93.28',12000))):
        print "sending ..."
        data = f.read(1024)
serverSocket.close()
f.close()
```

Receiver Code:

```
from socket import *
import sys
import select
clientSocket = socket(AF_INET,SOCK_DGRAM)
clientSocket.bind(('10.3.93.28',12000))
data,addr = clientSocket.recvfrom(1024)
print "Received File:",data.strip()
f = open(data.strip(),'wb')
try:
    while(data):
        f.write(data)
        clientSocket.settimeout(2)
        data,addr = clientSocket.recvfrom(1024)
except timeout:
    f.close()
    clientSocket.close()
    print "File Downloaded"
```

Task: 02:

Sender Code:

```
from socket import *
import sys
clientSocket = socket(AF_INET,SOCK_STREAM)
videoFile = 'small.mp4'
f=open(videoFile,'rb')
data = f.read(2048);
clientSocket.connect(('10.3.93.28',12055))
while(data):
        clientSocket.send((data))
        print "sending ..."
        data = f.read(2048)
f.close()
clientSocket.close()
```

Receiver Code:

```
from socket import *
import sys
import select
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('10.3.93.28',12055))
```

```
serverSocket.listen(1)
print 'Connection Established'
connectionSocket, addr = serverSocket.accept()
data = connectionSocket.recv(2048)
print "Received File:",data.strip()
f = open('abbas.mp4','wb')
while(data):
    f.write(data)
    data = connectionSocket.recv(2048)
f.close()
connectionSocket.close()
```