Week #5

Simple Client-Server Application using Network Socket Programming

(No Instructor-Led Training Lab)

Objective:

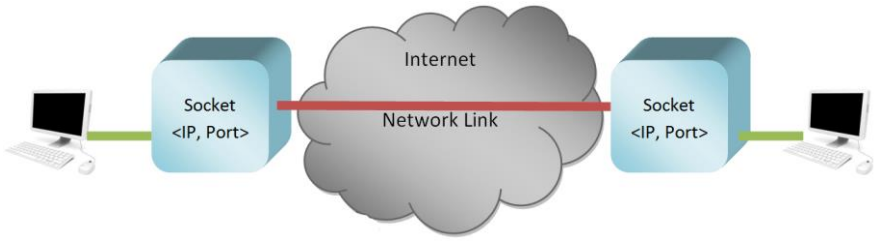
To develop a simple Client-Server application using TCP and UDP.

Pre requisites:

- Basic understanding of networking concepts and socket programming
- Knowledge of python

Sockets

Sockets are just the **endpoints of a two-way communication link** in a network. Socket helps in the communication of two processes/programs on a network (eg. Internet). The programs can communicate by reading/writing via their sockets. A socket comprises of: ***IP Address & Port number***



Task 1: (Mandatory for all students)

1. Create an application that will
    a. Convert lowercase letters to uppercase
        - e.g. [a…z] to [A…Z]
        - code will not change any special characters, e.g. &*!
    b. If the character is in uppercase, the program must not alter
2. Create Socket API both for
3. Must take the server address and port from the CLI

Socket Programming with UDP

UDPClient.py

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

UDPServer.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
message, clientAddress = serverSocket.recvfrom(2048)
modifiedMessage = message.upper()
serverSocket.sendto(modifiedMessage, clientAddress)
```

Socket Programming with TCP

TCPClient.py

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

Problems:

Install and compile the Python programs TCPClient and UDPClient on one host and TCPServer and UDPServer on another host.

1. Suppose you run TCPClient before you run TCPServer. What happens? Why?
2. Suppose you run UDPClient before you run UDPServer. What happens? Why?
3. What happens if you use different port numbers for the client and server sides?

Any one of the below tasks is mandatory for Week-5 completion and will be assigned by faculty members.

Task 2: Web Server

In this assignment, you will develop a simple Web server in Python that is capable of processing only one request. Specifically, your Web server will

a) create a connection socket when contacted by a client (browser);
b) receive the HTTP request from this connection;

c) parse the request to determine the specific file being requested;
d) get the requested file from the server's file system;
e) create an HTTP response message consisting of the requested file preceded by header lines; and
f) send the response over the TCP connection to the requesting browser.

If a browser requests a file that is not present in your server, your server should return a "404 Not Found" error message.

For this assignment, the companion Web site provides the skeleton code for your server. Your job is to complete the code, run your server, and then test your server by sending requests from browsers running on different hosts. If you run your server on a host that already has a Web server running on it, then you should use a different port than port 80 for your Web server.

Task 3: Multi-Threaded Web Proxy

In this assignment, you will develop a Web proxy. When your proxy receives an HTTP request for an object from a browser, it generates a new HTTP request for the same object and sends it to the origin server. When the proxy receives the corresponding HTTP response with the object from the origin server, it creates a new HTTP response, including the object, and sends it to the client. This proxy will be multi-threaded, so that it will be able to handle multiple requests at the same time.

For this assignment, the companion Web site provides the skeleton code for the proxy server. Your job is to complete the code, and then test it by having different browsers request Web objects via your proxy.

Task 4: Mail Client

The goal of this programming assignment is to create a simple mail client that sends email to any recipient. Your client will need to establish a TCP connection with a mail server (e.g., a Google mail server), dialogue with the mail server using the SMTP protocol, send an email message to a recipient (e.g., your friend) via the mail server, and finally close the TCP connection with the mail server.

For this assignment, the companion Web site provides the skeleton code for your client. Your job is to complete the code and test your client by sending email to different user accounts. You may also try sending through different servers (for example, through a Google mail server and through your university mail server).

Implementation Procedure:

The above-mentioned tasks can be implemented in several ways. They are,

1) Using 2 terminals
2) Using virtual machines (VirtualBox)
3) Connecting two machines using switch and cables
4) ClayNet

Any two methods should be used for deployment.

Edmodo Submissions:

- Client program for datagram service (UDP)
- Server program for datagram service (UDP)
- Client program for reliable byte-stream service (TCP)
- Server program for reliable byte-stream service (TCP)
- Screenshots of your test cases (terminal)
- Wireshark screenshots for all the communications