**Computer Networks Lab Report – Assignment 3**

**TITLE**

**Name – Aditya Ganguly**

**Roll – 002010501019**

**Class – BCSE 3rd year**

**Group – A1**

**Number – 8**

**Problem Statement: Implement any two protocols using TCP/UDP Socket as suitable.**

**1. FTP**

**2. DNS**

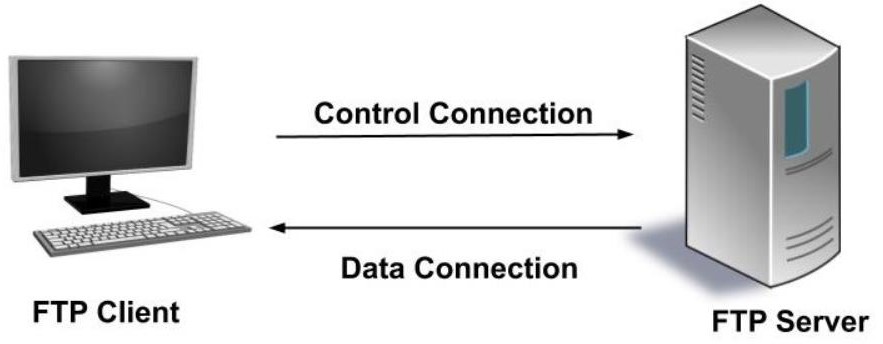
**3. Telnet**

**DESIGN**

**FTP (File Transfer Protocol)**

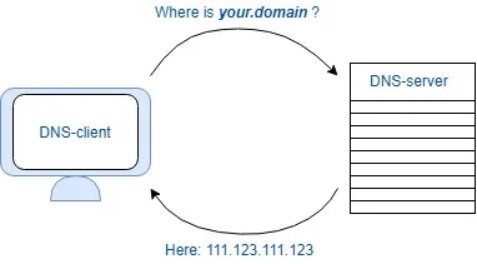
FTP is implemented using TCP sockets here and socket.SOCK\_STREAM has been used accordingly. Whenever a client connects to the FTP Server, it requests a file by sending the filename. The FTP server reads the file and transfers the contents of the file to the client.

The Client also has the options to Upload a file to the server, and to List the files with the server.



**DNS (Domain Name System)**

DNS is implemented using TCP sockets here and socket.SOCK\_STREAM has been used accordingly. DNS is a hostname for IP address translation service. The client sends a domain name to the DNS server, which then responds with the corresponding IP address of the server accordingly.



**CODE**

**FTP**

There are 3 files, FTP\_utils, FTP\_client, FTP\_server. FTP\_utils contails the functions for data and file transfer. FTP\_client and server contain the logic for control connection ans how it influences the data connection

**FTP\_utils.py**

import socket as socket

import random

import pickle

import socket as socket

import random

import pickle

import time

from multiprocessing import Process, Manager

from threading import Thread

import os

def dataSendOp(fil,clientAddr,serverAddr):

time.sleep(0.5)

TCPsock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

TCPsock.connect(serverAddr)

with open(fil, "r") as f:

lines = f.readlines()

TCPsock.send(str(len(lines)).encode())

time.sleep(2)

for line in lines:

TCPsock.send(line.encode())

time.sleep(2)

TCPsock.close()

def dataRecvOp(fil,clientAddr,serverAddr):

TCPsock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

TCPsock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

TCPsock.bind(serverAddr)

TCPsock.listen(1)

conn, addr = TCPsock.accept()

num = int(conn.recv(1024).decode())

time.sleep(2)

op = open(fil, "w")

for i in range(num):

msg = conn.recv(1024).decode()

op.write(msg)

time.sleep(1)

op.flush()

time.sleep(1)

conn.close()

TCPsock.close()

op.close()

def listSendOp(clientAddr,serverAddr,lis):

time.sleep(0.5)

TCPsock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

TCPsock.connect(serverAddr)

lis = pickle.dumps(lis)

TCPsock.send(lis)

TCPsock.close()

def listRecvOp(clientAddr,serverAddr):

time.sleep(0.5)

TCPsock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

TCPsock.bind(serverAddr)

TCPsock.listen(1)

conn,addr = TCPsock.accept()

lis = conn.recv(1024)

lis = pickle.loads(lis)

print(f"Received list {lis}")

conn.close()

TCPsock.close()

**FTP\_client.py**

import socket as socket

import random

import pickle

import time

from multiprocessing import Process, Manager

from threading import Thread

from FTPUtils import dataRecvOp,dataSendOp,listRecvOp,listSendOp

import tempfile

import os

controlClientAddr = ('127.0.0.1', 8023)

controlServerAddr = ('127.0.0.2', 8023)

dataClientAddr = ('127.0.0.1', 8022)

dataServerAddr = ('127.0.0.2', 8022)

def controlClientRoutine(clientDict):

controlClientAddr = ('127.0.0.1', 8023)

controlServerAddr = ('127.0.0.2', 8023)

print("Verifying authenticity")

username = input("Enter username :")

password = input("Enter password :")

data = ["VERIFY", username, password]

TCPsock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

TCPsock.connect(controlServerAddr)

TCPsock.setblocking(True)

TCPsock.send(pickle.dumps(data))

time.sleep(0.5)

reply = TCPsock.recv(1024)

reply = pickle.loads(reply)

if reply[0] == "ACCEPT":

print("User Verified")

while True:

code = input()

if code == "EXIT":

data = [code]

data = pickle.dumps(data)

TCPsock.send(data)

time.sleep(0.1)

clientDict["EXIT"] = True

TCPsock.close()

break

elif code == "SEND":

sendName = input("Enter the client filename ")

recvName = input("Enter the server filename ")

sendName = "client/"+sendName

recvName = "server/"+recvName

clientDict["SENDER"] = sendName

data = ["SEND", recvName]

data = pickle.dumps(data)

TCPsock.send(data)

elif code == "RECV":

recvName = input("Enter the client filename ")

sendName = input("Enter the server filename ")

recvName = "client/"+recvName

sendName = "server/"+sendName

clientDict["RECEIVER"] = recvName

data = ["RECV", sendName]

data = pickle.dumps(data)

TCPsock.send(data)

elif code=="LIST":

clientDict["LIST"]=True

data = ["LIST"]

data = pickle.dumps(data)

TCPsock.send(data)

elif code =="DEL":

name = input("Enter the name ")

name = "server/"+name

data = ["DEL",name]

data = pickle.dumps(data)

TCPsock.send(data)

time.sleep(0.5)

ack = TCPsock.recv(1024)

ack = pickle.loads(ack)

if ack[0]=="NOT FOUND":

print("File not found at server")

else:

clientDict[code]=True

else:

data = ["EXIT"]

data = pickle.dumps(data)

clientDict["Exit"] = True

TCPsock.send(data)

time.sleep(0.1)

TCPsock.close()

return

return

if \_\_name\_\_=='\_\_main\_\_':

dataClientAddr = ('127.0.0.4', 8022)

dataServerAddr = ('127.0.0.5', 8022)

clientDict = dict()

t1 = Thread(target=controlClientRoutine, args=[clientDict])

keys = ["EXIT", "SEND", "RECV", "SENDER", "RECEIVER","LIST"]

for k in keys:

clientDict[k] = False

t1.start()

while True:

if clientDict["EXIT"]:

break

if clientDict["SEND"]:

fil = clientDict["SENDER"]

t2 = Thread(target=dataSendOp, args=[

fil, dataClientAddr, dataServerAddr])

t2.start()

t2.join()

clientDict["SEND"]=False

if clientDict["RECV"]:

fil = clientDict["RECEIVER"]

t2 = Thread(target=dataRecvOp, args=[

fil, dataServerAddr, dataClientAddr])

t2.start()

t2.join()

clientDict["RECV"]=False

if clientDict["LIST"]:

t2 = Thread(target=listRecvOp, args=[

dataServerAddr, dataClientAddr])

t2.start()

t2.join()

clientDict["LIST"]=False

time.sleep(1)

t1.join()

**FTP\_server.py**

import socket as socket

import random

import pickle

import time

from multiprocessing import Process, Manager

from threading import Thread

from FTPUtils import dataRecvOp, dataSendOp, listRecvOp, listSendOp

import tempfile

import os

controlClientAddr = ('127.0.0.1', 8023)

controlServerAddr = ('127.0.0.2', 8023)

dataClientAddr = ('127.0.0.1', 8022)

dataServerAddr = ('127.0.0.2', 8022)

def controlServerRoutine(serverDict):

controlClientAddr = ('127.0.0.1', 8023)

controlServerAddr = ('127.0.0.2',8023)

fileList = ["server/"+i for i in os.listdir("server")]

TCPsock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

TCPsock.bind(controlServerAddr)

TCPsock.listen(1)

with open("server/hosts.txt","r") as f:

lines = f.readlines()

users = dict()

for line in lines:

username = line.split()[0]

password = line.split()[1]

users[username]=password

conn,addr = TCPsock.accept()

conn.setblocking(True)

data = conn.recv(1024)

data = pickle.loads(data)

if data[0]=="VERIFY":

username = data[1]

password = data[2]

if username not in users.keys() or users[username]!=password:

reply=["REJECT"]

else:

reply=["ACCEPT"]

time.sleep(0.2)

conn.send(pickle.dumps(reply))

while True:

conn.setblocking(True)

data = conn.recv(1024)

data = pickle.loads(data)

flag = 0

if data[0]=="EXIT":

serverDict["EXIT"]=True

conn.close()

TCPsock.close()

break

if data[0]=="SEND":

if data[1] in fileList:

serverDict["SEND"] = True

serverDict["RECEIVER"] = data[1]

flag=1

if data[0]=="RECV":

if data[1] in fileList:

serverDict["RECV"] = True

serverDict["SENDER"] = data[1]

flag = 1

if data[0]=="DEL":

if data[1] in fileList:

serverDict["DEL"]=True

serverDict["SENDER"]=data[1]

flag=1

if data[0]=="LIST":

flag = 1

serverDict["LIST"]=True

if flag==1:

ack="FOUND"

else:

ack = "NOT FOUND"

ack = pickle.dumps([ack])

conn.send(ack)

time.sleep(0.5)

if \_\_name\_\_=='\_\_main\_\_':

dataClientAddr = ('127.0.0.4', 8022)

dataServerAddr = ('127.0.0.5', 8022)

serverDict = dict()

t1 = Thread(target=controlServerRoutine, args=[serverDict])

keys = ["EXIT", "SEND", "RECV", "SENDER", "RECEIVER","LIST","DEL"]

for k in keys:

serverDict[k] = False

t1.start()

while True:

if serverDict["EXIT"]:

break

if serverDict["SEND"]:

fil = serverDict["RECEIVER"]

t2 = Thread(target=dataRecvOp, args=[

fil, dataClientAddr, dataServerAddr])

t2.start()

t2.join()

serverDict["SEND"]=False

if serverDict["RECV"]:

fil = serverDict["SENDER"]

t2 = Thread(target=dataSendOp, args=[

fil, dataServerAddr, dataClientAddr])

t2.start()

t2.join()

serverDict["RECV"]=False

if serverDict["LIST"]:

lis = os.listdir("server")

t2 = Thread(target=listSendOp, args=[

dataServerAddr, dataClientAddr,lis])

t2.start()

t2.join()

serverDict["LIST"]=False

if serverDict["DEL"]:

os.remove(serverDict["SENDER"])

serverDict["DEL"]=False

time.sleep(1)

t1.join()

**DNS**

There is a single file from which the server and client processes are forked. For every node of the namespace, a separate process is generated, message passing is done using UDP. Recursive DNS resolution is implemented where the initial request is made to the root and the final message is reached from a subtree and forwarded to the querying node.

import socket as socket

import random

import pickle

import time

from multiprocessing import Process, Manager

hostCount = 3

def insert(adj,wordList,currWord):

if len(wordList)==0:

return

if wordList[0] not in adj[currWord].keys():

adj[currWord][wordList[0]] = dict()

temp = wordList[0]

wordList.pop(0)

insert(adj[currWord],wordList,temp)

return

def assign(adj,label2ip,name2ip,currLis):

global hostCount

lis = reversed(currLis)

name = ".".join(lis)

name = name[0:-1]

label2ip[name] = (f"127.0.0.{hostCount}", 8080)

hostCount+=1

for word in adj.keys():

currLis.append(word)

assign(adj[word],label2ip,name2ip,currLis)

currLis.pop(-1)

return

def dnsServerRoutine(label2ip,name2ip,hostAddr):

# hostAddr = UDPsock.getsockname()

# UDPsock.close()

# time.sleep(1)

UDPsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPsock.settimeout(10)

UDPsock.bind(hostAddr)

start = time.time()

cnt=0

tempaddr = 0

while time.time()-start<=10:

try:

msg,addr = UDPsock.recvfrom(1024)

msg = pickle.loads(msg)

cnt+=1

if cnt==1:

tempaddr = addr

except Exception:

break

time.sleep(1)

if msg[1]=="YES":

UDPsock.sendto(pickle.dumps(msg),tempaddr)

break

currLis,eraseLis,adj,string = msg

name = ".".join(reversed(eraseLis))

print(f"At server {name}\n")

if len(currLis)==0 and name in name2ip.keys():

ip = name2ip[name]

data = [ip,"YES"]

UDPsock.sendto(pickle.dumps(data),tempaddr)

elif len(currLis)>0 and currLis[0] in adj.keys():

label = ".".join(reversed(eraseLis))

addr = label2ip[label]

temp = currLis[0]

eraseLis.append(currLis[0])

currLis.pop(0)

data = (currLis,eraseLis,adj[temp],"NO")

UDPsock.sendto(pickle.dumps(data),addr)

UDPsock.close()

return

def queryRoutine(string,label2ip,adj,hostAddr):

UDPsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPsock.settimeout(10)

UDPsock.bind(hostAddr)

currLis = list(reversed(string.split(".")))

eraseLis = []

addr = label2ip[""]

data = (currLis,eraseLis,adj[""],"NO")

UDPsock.sendto(pickle.dumps(data),addr)

time.sleep(1)

try:

msg,addr = UDPsock.recvfrom(1024)

msg = pickle.loads(msg)

print(f"Found ip address {msg[0]}\n")

except Exception:

print("Ip not found\n")

finally:

UDPsock.close()

return

if \_\_name\_\_ == '\_\_main\_\_':

manager = Manager()

name2ip = manager.dict()

label2ip = manager.dict()

label2ip[""] = ("127.0.0.1",8080)

adj = dict()

adj[""] = dict()

port = 8080

print("Enter the names and ip, -1 to stop")

inp = input()

while(inp!="-1"):

name = inp.split()[0]

ip = inp.split()[1]

wordList = name.split(".")

wordList.reverse()

insert(adj,wordList,"")

name2ip[name]=ip

inp = input()

assign(adj,label2ip,name2ip,[])

x = int(input("Continue ? "))

print(adj)

while x!=0:

ent = input("Enter the name to be searched ")

queryProcess = Process(target=queryRoutine,args=(ent,label2ip,adj,("127.0.0.1",8080)))

processDict = list()

for key in label2ip:

processDict.append(Process(target=dnsServerRoutine,args=(label2ip,name2ip,label2ip[key])))

for pro in processDict:

pro.start()

queryProcess.start()

queryProcess.join()

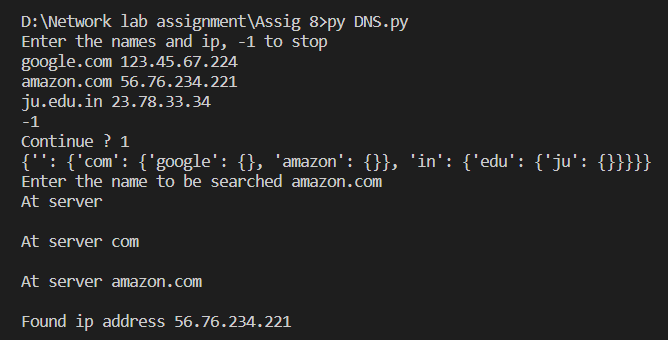
for pro in processDict:

pro.join()

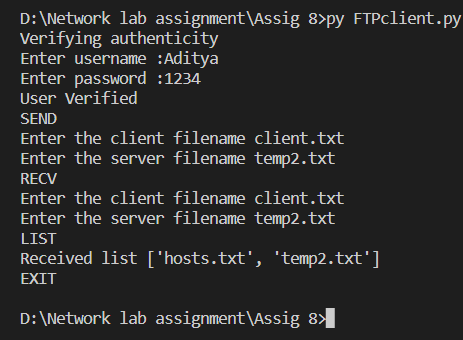
x = int(input("Continue ? "))

**OUTPUT**

**DNS**

****

**FTP**

****