**icmp.py file:**

from socket import \*

import os

import sys

import struct

import time

import select

import binascii

ICMP\_ECHO\_REQUEST = 8

timeRTT = []

packageSent =0;

packageRev = 0;

def checksum(string):

csum = 0

countTo = (len(string) // 2) \* 2

count = 0

while count < countTo:

thisVal = ord(string[count+1]) \* 256 + ord(string[count])

csum = csum + thisVal

csum = csum & 0xffffffff

count = count + 2

if countTo < len(string):

csum = csum + ord(string[len(string) - 1])

csum = csum & 0xffffffff

csum = (csum >> 16) + (csum & 0xffff)

csum = csum + (csum >> 16)

answer = ~csum

answer = answer & 0xffff

answer = answer >> 8 | (answer << 8 & 0xff00)

return answer

def receiveOnePing(mySocket, ID, timeout, destAddr):

global packageRev

global timeRTT

timeLeft = timeout

while 1:

startedSelect = time.time()

whatReady = select.select([mySocket], [], [], timeLeft)

howLongInSelect = (time.time() - startedSelect)

if whatReady[0] == []: # Timeout

return "0: Destination Network Unreachable,"

timeReceived = time.time()

recPacket, addr = mySocket.recvfrom(1024)

#Fill in start

#Fetch the ICMP header from the IP packet

icmpHeader = recPacket[20:28]

requestType, code, revChecksum, revId, revSequence = struct.unpack('bbHHh',icmpHeader)

if ID == revId:

bytesInDouble = struct.calcsize('d')

timeData = struct.unpack('d',recPacket[28:28 + bytesInDouble])[0]

timeRTT.append(timeReceived - timeData)

packageRev += 1

return timeReceived - timeData

else:

return "ID is not the same!"

#Fill in end

timeLeft = timeLeft - howLongInSelect

if timeLeft <= 0:

return "1: Destination Host Unreachable."

def sendOnePing(mySocket, destAddr, ID):

# Header is type (8), code (8), checksum (16), id (16), sequence (16)

myChecksum = 0

# Make a dummy header with a 0 checksum

# struct -- Interpret strings as packed binary data

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, ID, 1)

data = struct.pack("d", time.time())

# Calculate the checksum on the data and the dummy header.

myChecksum = checksum(str(header + data))

# Get the right checksum, and put in the header

if sys.platform == 'darwin':

# Convert 16-bit integers from host to network byte order

myChecksum = htons(myChecksum) & 0xffff

else:

myChecksum = htons(myChecksum)

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, ID, 1)

packet = header + data

mySocket.sendto(packet, (destAddr, 1)) # AF\_INET address must be tuple, not str

# Both LISTS and TUPLES consist of a number of objects

# which can be referenced by their position number within the object.

def doOnePing(destAddr, timeout):

icmp = getprotobyname("icmp")

# SOCK\_RAW is a powerful socket type. For more details: http://sockraw.org/papers/sock\_raw

mySocket = socket(AF\_INET, SOCK\_RAW, icmp)

myID = os.getpid() & 0xFFFF # Return the current process i

sendOnePing(mySocket, destAddr, myID)

delay = receiveOnePing(mySocket, myID, timeout, destAddr)

mySocket.close()

return delay

def ping(host, timeout=1):

# timeout=1 means: If one second goes by without a reply from the server,

# the client assumes that either the client's ping or the server's pong is lost

dest = gethostbyname(host)

print("Pinging " + dest + " using Python:")

print("")

# Send ping requests to a server separated by approximately one second

while 1 :

delay = doOnePing(dest, timeout)

print(delay)

time.sleep(1)# one second

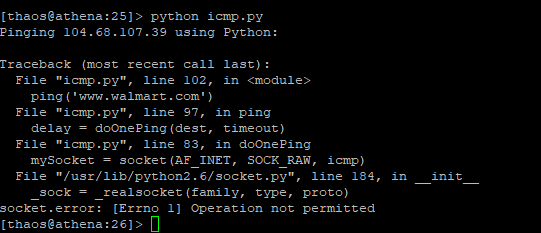
return delay

ping('www.walmart.com')

#ping('www.gmail.com')

#ping('www.yahoo.com')

#ping('www.google.com')



**stmp.py file:**

from socket import \*

msg = "\r\n I love computer networks!"

endmsg = "\r\n.\r\n"

# Choose a mail server (e.g. Google mail server) and call it mailserver

mailserver = "localhost"

port = 25

sender = "<test@ecs.csus.edu>"

recipient = "<samlee@csus.edu>"

# Create socket called clientSocket and establish a TCP connection with mailserver

clientSocket = socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((mailserver, port))

recv = clientSocket.recv(1024).decode()

print(recv)

if recv[:3] != '220':

print('220 reply not received from server.')

# Send HELO command and print server response.

print "Sending HELO command"

heloCommand = 'HELO Alice\r\n'

clientSocket.send(heloCommand.encode())

recv1 = clientSocket.recv(1024).decode()

print(recv1)

if recv1[:3] != '250':

print('250 reply not received from server.')

# Send MAIL FROM command and print server response.

print "Send MAIL FROM command"

mailFromCommand = 'MAIL FROM: ' + sender + '\r\n'

clientSocket.send(mailFromCommand.encode())

recv1 = clientSocket.recv(1024).decode()

print recv1

if recv1[:3] != '250':

print '250 reply not received from server'

# Send RCPT TO command and print server response.

print "Sending RCPT TO command"

rcptToCommand = 'RCPT TO: ' + recipient + '\r\n'

clientSocket.send(rcptToCommand.encode())

recv1 = clientSocket.recv(1024).decode()

print recv1

if recv1[:3] != '250':

print '250 reply not received from server'

# Send DATA command and print server response.

print "Send DATA command"

dataCommand = 'DATA\r\n'

clientSocket.send(dataCommand.encode())

recv1 = clientSocket.recv(1024).decode()

print recv1

if recv1[:3] != '354':

print ('354 reply not received from server')

# Send message data.

print "Sending message data"

clientSocket.send(msg + endmsg.encode())

recv1 = clientSocket.recv(1024).decode()

print recv1

if recv1[:3] != '250':

print ('250 reply not received from server\n')

# Message ends with a single period.

print "."

# Send QUIT command and get server response.

print "Sending QUIT command"

quitCommand = 'QUIT\r\n'

clientSocket.send("QUIT\r\n".encode())

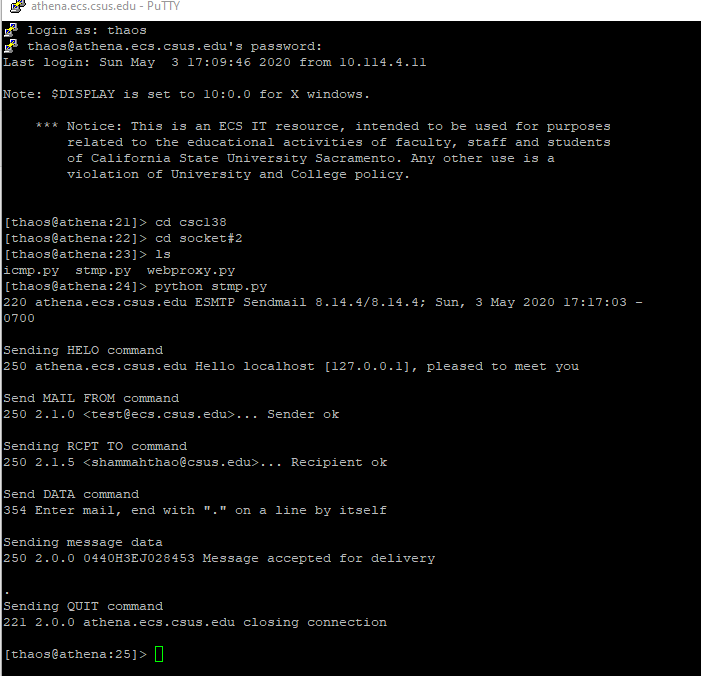
recv1 = clientSocket.recv(1024).decode()

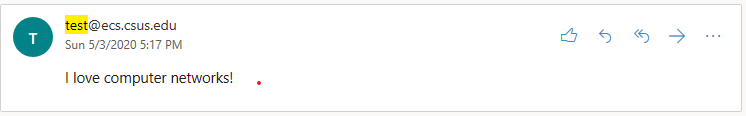
print recv1

if recv1[:3] != '221':

print ('221 reply not received from server')

clientSocket.close()





**webproxy.py file:**

from socket import \*

import sys

if len(sys.argv) <= 1:

print('Usage : "python ProxyServer.py server\_ip"\n[server\_ip : It is the IP Address Of Proxy Server')

sys.exit(2)

# Create a server socket, bind it to a port and start listening

tcpSerSock = socket(AF\_INET, SOCK\_STREAM)

tcpSerPort = 2345

tcpSerSock.bind(('', tcpSerPort))

tcpSerSock.listen(5)

while 1:

# Strat receiving data from the client

print('Ready to serve...')

tcpCliSock, addr = tcpSerSock.accept()

print('Received a connection from:', addr)

message = tcpCliSock.recv(1024)

print(message)

# Extract the filename from the given message

print(message.split()[1])

filename = message.split()[1].partition("/")[2]

print(filename)

fileExist = "false"

filetouse = "/" + filename

print(filetouse)

try:

# Check wether the file exist in the cache

f = open(filetouse[1:], "r")

outputdata = f.readlines()

fileExist = "true"

# ProxyServer finds a cache hit and generates a response message

tcpCliSock.send("HTTP/1.0 200 OK\r\n")

tcpCliSock.send("Content-Type:text/html\r\n")

for i in range(0, len(outputdata)) :

tcpCliSock.send(outputdata[i])

print('Read from cache')

# Error handling for file not found in cache

except IOError:

if fileExist == "false":

# Create a socket on the proxyserver

c = socket(AF\_INET, SOCK\_STREAM) # Fill in start. # Fill in end.

hostn = filename.replace("www.","",1)

print(hostn)

try:

# Connect to the socket to port 80

c.connect(hostn, 80)

print 'Socket connected to port 80 of the host'

# Create a temporary file on this socket and ask port 80 for the

# file requested by the client

fileobj = c.makefile('r', 0)

fileobj.write("GET "+"http://" + filename + " HTTP/1.0\n\n")

# Read the response into buffer

buff = fileobj.readlines()

# Create a new file in the cache for the requested file.

# Also send the response in the buffer to client socket and the

# corresponding file in the cache

tmpFile = open("./" + filename,"wb")

for i in range(0, len(buff)) :

tmpFile.write(buff[i])

tcpCliSock.send(buff[i])

except:

print("Illegal request")

else:

print '404 Error file not found.'

#Close the client and server sockets

tcpCliSock.close()

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

main()

