

Code for SMTP Client

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" #Fill in start #Fill in end

serverPort = 25;

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

#Fill in start

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

clientSocket.connect((mailserver,serverPort));

#Fill in end

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.

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.

# Fill in start

print ("Sending MAIL FROM Command")

mailfromCommand = 'MAIL FROM:talaljawaid916@gmail.com\r\n'

clientSocket.send(mailfromCommand.encode())

recv2 = clientSocket.recv(1024).decode()

if recv2[:3] != '250':

print("250 Reply not Received")

# Fill in end

# Send RCPT TO command and print server response.

# Fill in start

print("Sending RCPT TO Command");

rcpttoCommand = 'RCPT TO: talaljawaid916@gmail.com\r\n'

clientSocket.send((rcpttoCommand).encode());

recv3 = clientSocket.recv(1024).decode();

# Fill in end

# Send DATA command and print server response.

# Fill in start

print("Sending DATA Command");

dataCommand = 'DATA\r\n'

clientSocket.send((dataCommand).encode());

recv4 = clientSocket.recv(1024).decode();

print(recv4)

if recv4[:3] != '354':

print("354 Reply Not Received")

# Fill in end

# Send message data.

# Fill in start

print("Sending message data");

clientSocket.send(msg);

if recv4[:3] != '250':

print("250 Reply Not Received")

# Fill in end

# Message ends with a single period.

# Fill in start

print("Sending endmsg");

clientSocket.send((endmsg).encode());

# Fill in end

# Send QUIT command and get server response.

# Fill in start

print("Sending QUIT Command");

quitCommand = 'QUIT\r\n'

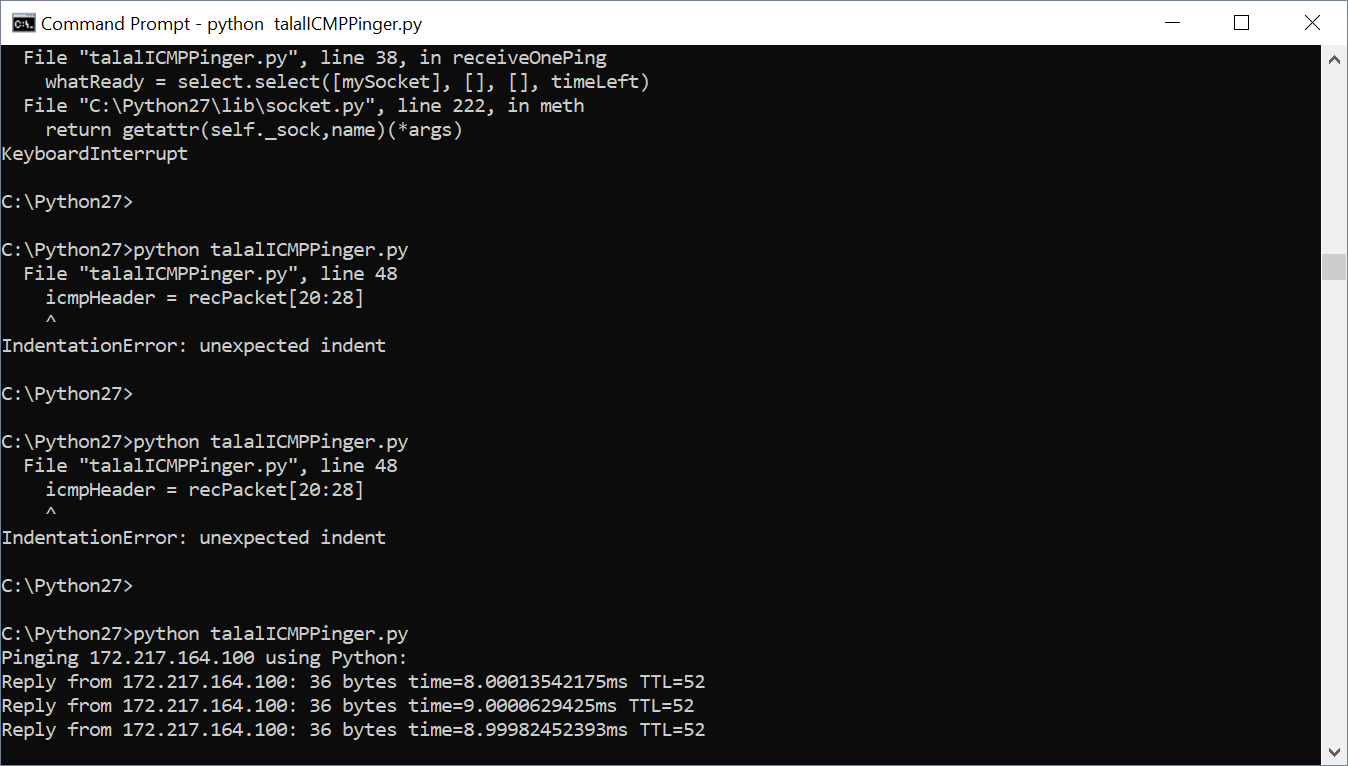
clientSocket.send(quitCommand);

if recv4[:3] != '250':

print("250 Reply Not Received");

# Fill in end

Plab Part 2 ICMP



from socket import \*

import os

import sys

import struct

import time

import select

import binascii

ICMP\_ECHO\_REQUEST = 8

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):

timeLeft = timeout

while 1:

startedSelect = time.time()

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

howLongInSelect = (time.time() - startedSelect)

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

return "Request timed out."

timeReceived = time.time()

recPacket, addr = mySocket.recvfrom(1024)

#Fill in start

#Fetch the ICMP header from the IP packet

send\_time, = struct.unpack('d', recPacket[28:])

rtt = (timeReceived - send\_time) \* 1000

ip\_header = struct.unpack('!BBHHHBBH4s4s' , recPacket[:20])

timeToLive = ip\_header[5]

saddr = socket.inet\_ntoa(ip\_header[8])

length = len(recPacket)

return 'Reply from {}: {} bytes time={}ms TTL={}'.format(saddr, length, rtt, timeToLive)

#Fill in end

timeLeft = timeLeft - howLongInSelect

if timeLeft <= 0:

return "Request timed out."

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':

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");

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("127.0.0.1")

ping("www.google.com")

ping("google.com.pk")

ping("google.co.uk")

ping("google.co.ke")