CSE476 – Mobile Communication Networks Course

Term Project

Progress Report

OĞUZHAN AGKUŞ
161044003

Assignment 1 – Web Server

In this assignment, I have implemented a simple web server. It does not use any Python libaries for communication, only uses TCP sockets. It is capable of handling HTTP GET requests. The requested page is send back and the socket will be closed. I needed to change the given skeloton a little bit. Additionally, I write a signal handler for CTRL+C because I want to shutdown my server succesfully. First part of my code is like below:

```python
import socket, signal, datetime

# Set global variables
server_address = ("", 80)
server_socket = None

# If ctrl+c occurs
def signal_handler(signal, frame):
  server_socket.close()
  exit(0)

# Prepare socket
try:
  server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  server_socket.bind(server_address)
  server_socket.listen(5)
except Exception as error:
  print("Error occured!", error)
  exit(1)

# Set signal handler
signal.signal(signal.SIGINT, signal_handler)
```

It listens incoming requests in a infinte loop. When request method is not GET it raises an exception. Otherwise, it tries to open and read requested page. If an error occurs here, it raises some expetions which will be handled following “except” part.

```python
while True:
  # Accept a connection
  print("Server is ready!")
  client_socket, address = server_socket.accept()
  client_socket.settimeout(5)

  try:
    request = client_socket.recv(1024).decode()
    items = request.split()

    request_type = items[0]
    if request_type != "GET":
      raise Exception("Method not allowed!")

    filename = items[1]
    file = open(filename[1:])
    data = file.read()
    file.close()
```

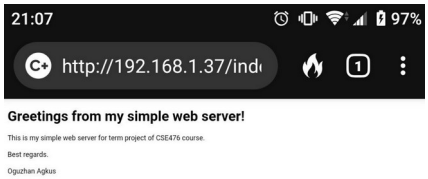
If there is no error until here, it creates and send HTTP headers. Then sends the data of the page.

```python
    # Prepare headers
    status_line = "HTTP/1.1 200 OK\r\n"
    header_lines = "Connection: close\r\n"
    header_lines += "Date: {}\r\n".format(datetime.datetime.now())
    header_lines += "Server: MyServer\r\n"
    header_lines += "Content-Length: {}\r\n".format(len(data))
    header_lines += "Content-Type: text/html\r\n"
    blank_line = "\r\n"
    response_message = status_line + header_lines + blank_line

    # Send headers
    client_socket.send(response_message.encode())

    # Send content of the file
    for i in range(len(data)):
      client_socket.send(data[i].encode())

    print(filename + " sent to " + address[0])

    # Close client socket
    client_socket.shutdown(socket.SHUT_RDWR)
    client_socket.close()
```

Following part is exception handling part. Returns apporpriate response message and closes the socket of the client socket. The last line is out of the try-except block. It closes the server socket.

```python
    except Exception as error:
      # Print and response error message
      print("Error occured!", error)
      if (error == "Method not allowed!"):
        client_socket.send("HTTP/1.1 405 METHOD NOT ALLOWED\r\n".encode())
      else:
        client_socket.send("HTTP/1.1 404 NOT FOUND\r\n".encode())

      # Close client socket
      client_socket.shutdown(socket.SHUT_RDWR)
      client_socket.close()

server_socket.close()
```

I create a index.html page for testing my server. It is in the same directory with the code. I tried to connect from different devices (which are in my local network) to my server. The browser screenshots are:

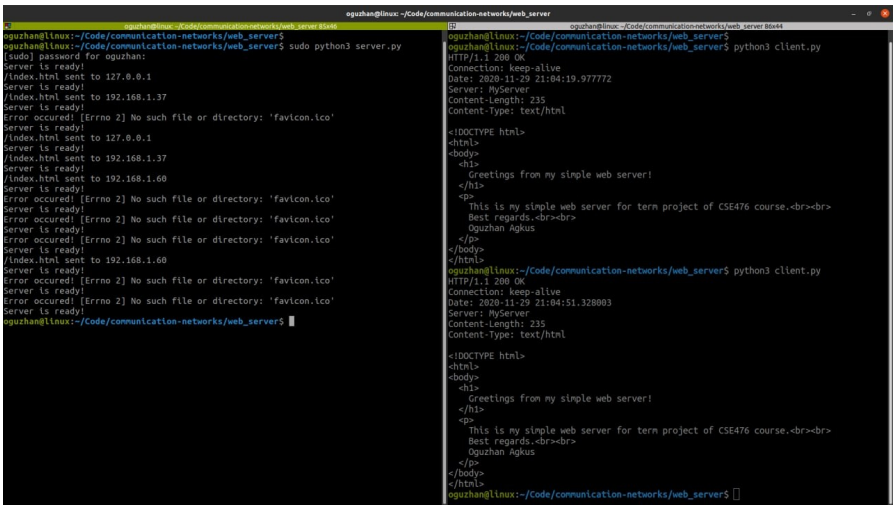


I implemented optional exercise client code which simulates the browser. It sends request and prints the server's response. It is possible to pass arguments to code but it has defaults. I will make better my server with threading.

```python
import socket, sys

# Default arguments
server_host = "localhost"
server_port = 80
filename = "index.html"

# If any argument passed
if len(sys.argv) == 4:
  server_host = sys.argv[1]
  server_port = int(sys.argv[2])
  filename = sys.argv[3]

# Global variables
server_address = (server_host, server_port)
connection = None

# Prepare socket
try:
  connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  connection.connect(server_address)
except Exception as error:
  print("Error occured!", error)
  sys.exit(1)
```

```python
# Prepare headers
request_line = "GET /" + filename + " HTTP/1.1\r\n"
header_lines = "Host: " + server_host + "\r\n"
header_lines += "Connection: close\r\n"
blank_line = "\r\n"
request_message = request_line + header_lines + blank_line

# Send headers
connection.send(request_message.encode())

# Recieve data
response = ""
while True:
    data = connection.recv(1024).decode()
    if not data:
        break
    response += data

# Print data
print(response)

# Close socket
connection.close()
```

The terminal screenshot of both server and client is below:



Assignment 2 – UDP Pinger

In this assignment, I have implemented a ping server and client which uses UDP instead of ICMP. Actually I modified the server code. I think it has some extra and missing parts. But I did not change the main purpose of the server. It accetps connections, reads data, simulates data loss and sends back the incoming data. The final form of the server:

```python
import socket, signal, random

# Set global variables
server_address = ("", 12000)
server_socket = None

# If CTRL+C occurs
def signal_handler(signal, frame):
  server_socket.close()
  exit(0)

# Prepare UDP socket
try:
  server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  server_socket.bind(server_address)
except Exception as error:
  print("Error occured!", error)
  exit(1)

# Set signal handler
signal.signal(signal.SIGINT, signal_handler)

# Main loop
while True:
  message, address = server_socket.recvfrom(1024)
  print(message.decode())

  loss = random.randint(0, 10)
  if loss < 4:
    continue

  server_socket.sendto(message, address)
```

The client send 10 packets. It has 1 second timeout. The sending sequence cannot be interrupted.

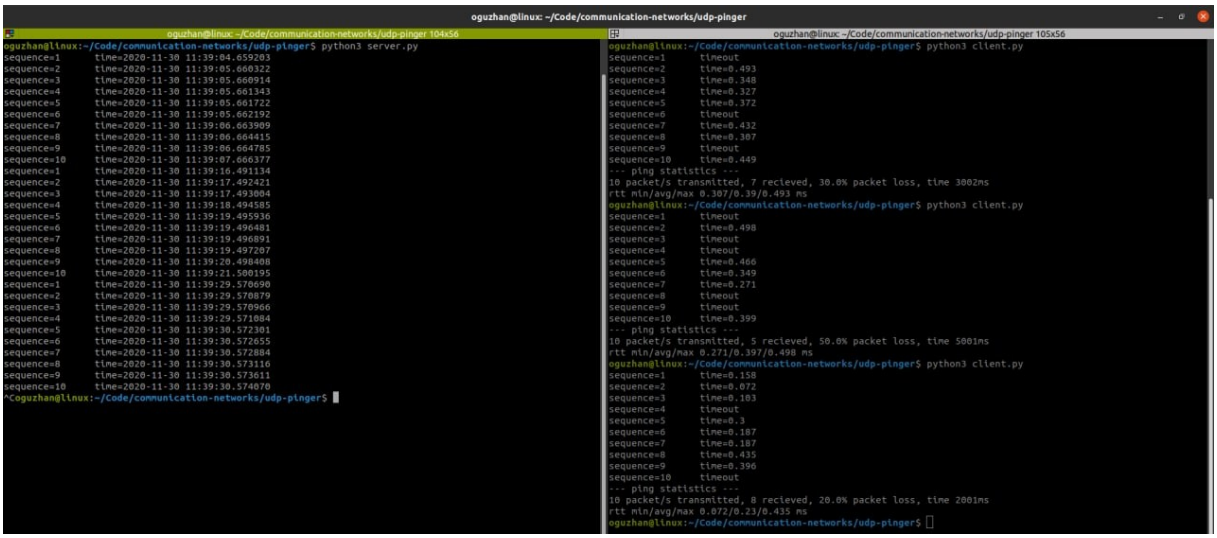
```python
import socket, signal, datetime

# Set global variables
server_address = ("", 12000)
client_socket = None
packet_count = 10
rtt_table = []
lost = 0

# If CTRL+C occurs
def signal_handler(signal, frame):
  pass

# Prepare UDP socket
try:
  client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  client_socket.settimeout(1)
except Exception as error:
  print("Error occured!", error)
  exit(1)

# Set signal handler
signal.signal(signal.SIGINT, signal_handler)
```

```python
# Sequential package sending
for i in range(1, packet_count + 1):
  temp = datetime.datetime.now()
  message = "sequence={}\ttime={}".format(i, temp)
  client_socket.sendto(message.encode(), server_address)

  try:
    reply_message, temp_address = client_socket.recvfrom(1024)
    duration = datetime.datetime.now() - temp
    rtt_table.append(duration)
    print("sequence={}\ttime={}".format(i, duration.microseconds / 1000))
  except socket.timeout:
    print("sequence={}\ttimeout".format(i))
    lost += 1

# Close socket
client_socket.close()
```

Calculating ping statistics.

```python
# Calculate statistics
total = datetime.timedelta()
minimum = datetime.timedelta()
maximum = datetime.timedelta()
average = datetime.timedelta()

if len(rtt_table):
  total = sum(rtt_table, datetime.timedelta())
  minimum = min(rtt_table)
  maximum = max(rtt_table)
  average = total / len(rtt_table)

total = sum(rtt_table, datetime.timedelta())
minimum = min(rtt_table)
maximum = max(rtt_table)
average = total / len(rtt_table)

time = lost * 1000 + total.microseconds / 1000

# Print statistics
print("--- ping statistics ---")
print("{} packet/s transmitted, {} recieved, {}% packet loss, time {}ms".format(packet_count, packet_count - lost,
lost / packet_count * 100, time))
print("rtt min/avg/max {}/{}/{} ms".format(minimum.microseconds / 1000, average.microseconds / 1000,
maximum.microseconds / 1000))
```

The sample output is at below. The left side is server and the right side is client. Also I show RTT values in ms. Because second is too big unit to measure delays, especially in local network.



I will implement Heartbeat exercise.

Assignment 3 – Mail Client

In this assignment I implemented a mail client which uses only sockets. It sends STMP commands over a TCP socket. To send a mail, you need to login on your mail server. I defined two mail servers: Gmail and Office 365. You can choose one of the mor add yours. Also you should define your username and password for the server. The reciever and mail informations are globally defined variables. There is no user input part. I have some handler functions to close socket and STMP session securely.

```
import socket, ssl, base64
import time, signal

# Set your authentication information
username = "*****"
password = "*****"

# Choose your mail server
gmail = ("smtp.gmail.com", 587)
office = ("smtp.office365.com", 587)
mail_server = gmail

# Set email details
reciever = "agkusoguzhan@gmail.com"
subject = "Test Mail From My Client"
message = "This is my simple SMTP client for term project of CSE476 course.\n\nBest regards.\n\nOguzhan Agkus\n\n"

# Global declaration for using in the functions below
client_socket = None

# After connection established, if server returns unexpected reply then call this function
def exit_handler(message = None, exit_code = 0):
  if message:
    print(message)

  client_socket.send("QUIT\r\n".encode())
  client_socket.close()
  exit(exit_code)

# If CTRL+C occurs
def signal_handler(signal, frame):
  exit_handler()
```

If the server sends a unexpected reply, the program exits.

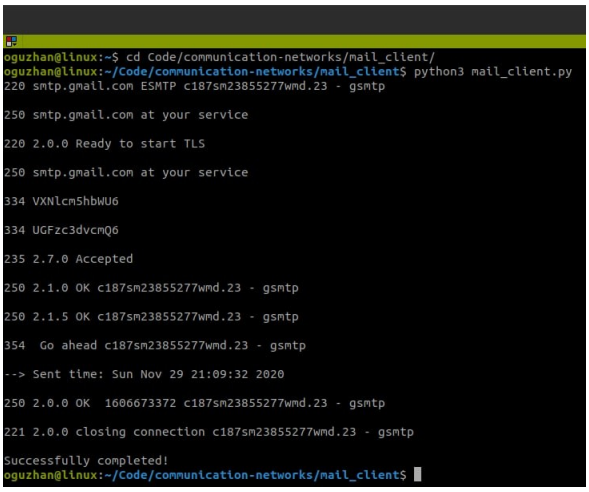
```python
# Establish TCP connection
try:
  client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  client_socket.connect(mail_server)
except Exception as error:
  print("Error occured!", error)
  exit(1)

# Set signal handler
signal.signal(signal.SIGINT, signal_handler)

# Get first data
data = client_socket.recv(1024).decode()
print(data)
if data[:3] != "220":
  exit_handler("Reply is not 220 after connection established!", 1)

# Send HELO command
client_socket.send("HELO MYSERVER\r\n".encode())
data = client_socket.recv(1024).decode()
print(data)
if data[:3] != "250":
  exit_handler("Reply is not 250 after hello!", 1)

# Send STARTTLS command, if encryption is possible, start a new session
client_socket.send("STARTTLS\r\n".encode())
data = client_socket.recv(1024).decode()
print(data)
if data[:3] == "220":
  context = ssl._create_stdlib_context(certfile=None, keyfile=None)
  client_socket = context.wrap_socket(client_socket)

  client_socket.send("HELO MYSERVER\r\n".encode())
  data = client_socket.recv(1024).decode()
  print(data)
  if data[:3] != "250":
    exit_handler("Reply is not 250 after TLS connection!", 1)
```

If the server support TLS, the client starts a TLS session. Most of modern servers, Gmail or Office 365, required encrypted connections. It was optional part but it is probably a requirement. After starting a TLS session, the server asks username and password. Also I need the encode both username and password with base64. Finally send mail details and quit from the server.

```python
# Send AUTH command
client_socket.send("AUTH LOGIN\r\n".encode())
data = client_socket.recv(1024).decode()
print(data) # It says send username

# Send username first, encode it using base64
client_socket.send(base64.b64encode(username.encode()))
client_socket.send("\r\n".encode())
data = client_socket.recv(1024).decode()
print(data) # It says send password

# Send password then, encode it using base64
client_socket.send(base64.b64encode(password.encode()))
client_socket.send("\r\n".encode())
data = client_socket.recv(1024).decode()
print(data) # Accepted or failed

if data[:3] != "235":
  exit_handler("Authentication failed!", 1)
```

```python
# Send MAIL FROM command
client_socket.send("MAIL FROM:<{}>\r\n".format(username).encode())
data = client_socket.recv(1024).decode()
print(data)

# Send RCPT TO command
client_socket.send("RCPT TO:<{}>\r\n".format(reciever).encode())
data = client_socket.recv(1024).decode()
print(data)

# Send DATA command
client_socket.send("DATA\r\n".encode())
data = client_socket.recv(1024).decode()
print(data)

# A timestamp to compare outgoing with incoming mail
timestamp = time.asctime()
print("--> Sent time:", timestamp, end="\n\n")

# Send message data
client_socket.send("Subject: {}\r\n\r\n".format(subject).encode())
client_socket.send(message.encode())
client_socket.send("Sent time: {} \r\n".format(timestamp).encode())
client_socket.send("\r\n.\r\n".encode())
data = client_socket.recv(1024).decode()
print(data)

# Send QUIT command
client_socket.send("QUIT\r\n".encode())
data = client_socket.recv(1024).decode()
print(data)

client_socket.close()

print("Successfully completed!")
```

The terminal output of the client and recieved mail are:



```
oguzhan@linux:~$ cd Code/communication-networks/mail_client/
oguzhan@linux:~/Code/communication-networks/mail_client$ python3 mail_client.py
220 smtp.gmail.com ESMTP c187sm23855277wmd.23 - gsmtp

250 smtp.gmail.com at your service

220 2.0.0 Ready to start TLS

250 smtp.gmail.com at your service

334 VXNlcm5hbWU6

334 UGFzc3dvcmQ6

235 2.7.0 Accepted

250 2.1.0 OK c187sm23855277wmd.23 - gsmtp

250 2.1.5 OK c187sm23855277wmd.23 - gsmtp

354  Go ahead c187sm23855277wmd.23 - gsmtp

--> Sent time: Sun Nov 29 21:09:32 2020

250 2.0.0 OK  1606673372 c187sm23855277wmd.23 - gsmtp

221 2.0.0 closing connection c187sm23855277wmd.23 - gsmtp

Successfully completed!
oguzhan@linux:~/Code/communication-networks/mail_client$
```

Test Mail From My Client Gelen Kutusu ×

[redacted]@gmail.com

Alıcı:

This is my simple SMTP client for term project of CSE476 course.

Best regards.

Oguzhan Agkus

Sent time: Sun Nov 29 21:09:32 2020

I will add send image functionality.