# Second Network Programming Homework

Shafik Rami Ismail
2294

# Bugs faced:

**Bug 1:**
socket.bind() takes one argument as a two element list, and it was cause by me deleting the extra () because I thought it was added by mistake.
So re-added the extra ().

```
a2@a2-Aspire-A315-53G: /media/home2/home/a2/Py...
a2@a2-Aspire-A315-53G:/media/home2/home/a2/PycharmProjects/pythonProject1$
 python setup_echo.py
Traceback (most recent call last):
  File "/media/home2/home/a2/PycharmProjects/pythonProject1/setup_echo.py"
, line 22, in <module>
    listen_socket = tincanchat.create_listen_socket(host, port)
  File "/media/home2/home/a2/PycharmProjects/pythonProject1/tincanchat.py"
, line 10, in create_listen_socket
    sock.bind(host, port)
TypeError: socket.bind() takes exactly one argument (2 given)
a2@a2-Aspire-A315-53G:/media/home2/home/a2/PycharmProjects/pythonProject1$
```

**Bug 2:**
Cause by me again selecting a port number of 1000, because I lie this number, so this error occurs when selecting a port number equals or lower than 1024 because it requires root privileges.

```
a2@a2-Aspire-A315-53G: /media/home2/home/a2/PycharmP...
a2@a2-Aspire-A315-53G:/media/home2/home/a2/PycharmProjects/pythonProject1$ pytho
n setup_echo.py
Traceback (most recent call last):
  File "/media/home2/home/a2/PycharmProjects/pythonProject1/setup_echo.py", line
 22, in <module>
    listen_socket = tincanchat.create_listen_socket(host, port)
  File "/media/home2/home/a2/PycharmProjects/pythonProject1/tincanchat.py", line
 10, in create_listen_socket
    sock.bind((host, port))
PermissionError: [Errno 13] Permission denied
a2@a2-Aspire-A315-53G:/media/home2/home/a2/PycharmProjects/pythonProject1$ ^C
```

**Bug 3:**
Cause by me again deleting extra () in socket.connect(), because it’s same as socket.bind()

**Bug 4:**
Now I'm stuck between server and client only have one send and one receive (one waiting for the other), and the thread is locked on that, so only one send or one receive at a time.
Possible solutions: either a dedicated thread for send and on for receive, or careful collecting of server messages, and then send them as one message.
Fixed it by using flags("!!!END!!!","!!!NO_UNPUT!!!") so the client shouldn't have an input for these king of messages, but faced bug 5 now.

**Bug 5:**
Sending two messages from server, and client receiving them as a one message, so made an improved version of "recv_msg" method which is "recv_multi_msg" and it splits one message into multiple ones using encoding "!!!111!!!" that I added right before sending each message, and deleted it after receiving messages, and also used it to split messages.

# Question 1: TCP Server/Client App with Multi-Threading
# Answer :

There are two folders, one for client stuff, and the other for server stuff.
The **server** folder contains 3 python files(tincanchat.py, quiz.py, setup_multi_echo.py) and one json files that contains the quiz(reused it from the first homework).

## Code:
**File tincanchat.py:**

```
import socket

host = '127.0.0.1'
port = 4040
encoding = 'utf-8'
#added an end to each message before sending to avoid mixing messages
message_end = "!!!111!!!"

def create_listen_socket(host, port):

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((host, port))
    sock.listen(100)

    return sock

def recv_msg(sock):
    #print("in receive waiting")

    data = bytearray()
    message = "
```

```
    while not message:
        #Waiting to receive data
        received_data = sock.recv(4096)


        if not received_data:
            raise ConnectionError()

        data = data + received_data

        if b'\0' in received_data:
            message = data.rstrip(b'\0')

    message = message.decode(encoding)
    message = message.strip(message_end)

    return message

def recv_multi_msg(sock):
        #if multiple messages expected use this message for receive(receiving
        # from a server

    data = bytearray()
    message = ''

    while not message:
        received_data = sock.recv(4096)


        if not received_data:
            raise ConnectionError()

        data = data + received_data

        if b'\0' in received_data:
```

```
        message = data.rstrip(b'\0')

    # for splitting a message containing multiple messages
    message = message.decode(encoding)
    messages = message.split(message_end)

    #here the filter is used to delete empty strings “” from messages list
    return list(filter(None, messages))

def prep_msg(message):

    message += '\0'
    return message.encode(encoding)

def send_msg(sock, message):
    #add message end before sending a message
    sock.sendall(prep_msg(message+message_end))
```

#######################End of File tincanchat.py:

## Explanation:

This file is used by both server and client for sending and receiving messages.
Added an end to each message before sending and spitted each message with the message end to multiple messages, because TCP can mix messages on sockets before sending them.

**File quiz.py:**

```
import json
import tincanchat

#used for statement messages
NO_INPUT = "!!!NO_INPUT!!!"
END_OF_QUIZ = "!!!END!!!"
def start(client_socket):
```

```
try:
    # Result variable to store user's name, answers and final mark
    results = {}
    final_mark = 0

    # Opening the json file
    file = open("quiz.json")
    quiz = json.load(file)

    message = "Enter your name: "
      #send enter name message
    tincanchat.send_msg(client_socket, message)
      #wait for name reply
    print("sent a message from server")
    user_name = tincanchat.recv_msg(client_socket)
    print("User name: "+user_name)

    # First value is user's name
    results["Name"] = user_name

    # There's only two values a user can enter: t for true, f for false
    instructions = "Answer \"t\" for True and \"f\" for False"
      #instuctions are statement message so add NO_INPUT
    tincanchat.send_msg(client_socket, "!!!NO_INPUT!!!"+instructions)

    j = 1
    # For loop for items in json file(quiz)
    for item in quiz:

        # Answer enter by user
        tincanchat.send_msg(client_socket, item)
        answer = tincanchat.recv_msg(client_socket)
        # Correct answer from json file, which is first element of list
        correct_answer = quiz[item][0]

        # If user entered a value that's not t or f, show a warning
        if not answer.__eq__("t") and not answer.__eq__("f"):

            instructions = "please answer with \"t\" for True, and \"f\" for False"
            tincanchat.send_msg(client_socket, NO_INPUT+instructions)
            tincanchat.send_msg(client_socket, item)
```

```
            answer = tincanchat.recv_msg(client_socket)

        # If user answered correctly, increase the mark
        if answer.__eq__(correct_answer):
            tincanchat.send_msg(client_socket, NO_INPUT+"Correct")
            final_mark += 5
            answer = "correct"

        #If user answered incorrectly, don't change the mark, and output the right answer
        else:
            reply = ""

            if correct_answer.__eq__("f"):
                reply = "Wrong, correct answer is: " + quiz[item][1]
            elif correct_answer.__eq__("t"):
                reply = "Wrong, correct answer is true"

            tincanchat.send_msg(client_socket, NO_INPUT+reply)

            answer = "Wrong"

        # Add that user answer correctly or incorrectly to question X in results
        results["Question" + str(j)] = answer
        j += 1

    results["Result"] = final_mark

    reply = "Thanks for taking the test, your result is: " + str(final_mark) + "/100"
    tincanchat.send_msg(client_socket, END_OF_QUIZ + reply)

    # write results variable to a json file
    with open(user_name+"_results.json", "w") as write_file:
        json.dump(results, write_file)

    json.dumps(results)

except (ConnectionError, BrokenPipeError):
    print("socket error from quiz.py")

finally:
    print('closed connection')
```

#######################End of File quiz.py:

## Explanation:

This file is used in the first homework, so I've converted print statements into sending the answer to client, and input statements into receiving answer from client.
Fetched questions from a json file on the server side.
Added tags before messages that requires no input from user that's to avoid waiting for an answer when there's a statement from server.
Stored user marks and answer on the server side after finishing the exam as a json file.

### File setup_multi_echo.py:

```
import tincanchat
import threading
from quiz import start

host = tincanchat.host
port = tincanchat.port


if __name__ == '__main__':
    listen_socket = tincanchat.create_listen_socket(host, port)
    address = listen_socket.getsockname()
    print('listening on {}'.format(address))

    while True:
        #Waiting for a connecting socket from client
        client_socket, address = listen_socket.accept()

        #Thread for each client
        thread = threading.Thread(target=start
                                  , args=[client_socket]
                                  , daemon=True)

        thread.start()
        print('connection from {}'.format(address))
```

#######################End of File setup_multi_echo.py:

## Explanation:

This file is used to enable each client to connect to the server synchronously. It just waits for a connecting socket, then create a thread for that connection.

####################### End of server files
####################### Client files :

The **client** folder contains 2 python files(tincanchat.py, connect_to_echo.py).

## Code:

### File tincanchat.py:

same as the one used by server.

### File connect_to_echo.py:

```
import socket, sys
import traceback

import tincanchat

host = sys.argv[-1] if len(sys.argv) > 1 else '127.0.0.1'
port = tincanchat.port

if __name__ == '__main__':

    try:
        #connect to server
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((host, port))

        print('connected to {}:{}'.format(host, port), end="\n")

        break_first_loop = False
```

```
    # a loop for receiving and sending messages
  while True:
    try:
        #multiple messages are expected from server,
        # so use receive multiple messages function
      messages = tincanchat.recv_multi_msg(sock)
       #print("messages received", messages)

        # a loop to handle all received messages
      for message in messages:
        #print("current message is ", message)
        # messages contains !!!END!!! means the the server is ending the
        #connection
        if message.__contains__("!!!END!!!"):
          message = message.split("!!!END!!!")[1]
          print(message, end="\n")
          break_first_loop = True
          break
        # messages contains !!!NO_INPUT!!! means the the server is making
        #statements
        elif message.__contains__("!!!NO_INPUT!!!"):
          print(message.split("!!!NO_INPUT!!!")[1], end="\n")
        #other messages require client input
        else:
          question = input(message.strip("\x00"))

          if message == 'q':
            break_first_loop = True
            break

          tincanchat.send_msg(sock, question)
          #print("sent message: {}".format(question), end="\n")

      if break_first_loop:
        break
    except Exception:
      print("error", traceback.format_exc())
      break

except ConnectionError:
  print("socket error")
```

```
    finally:
        print("closed connection")
```

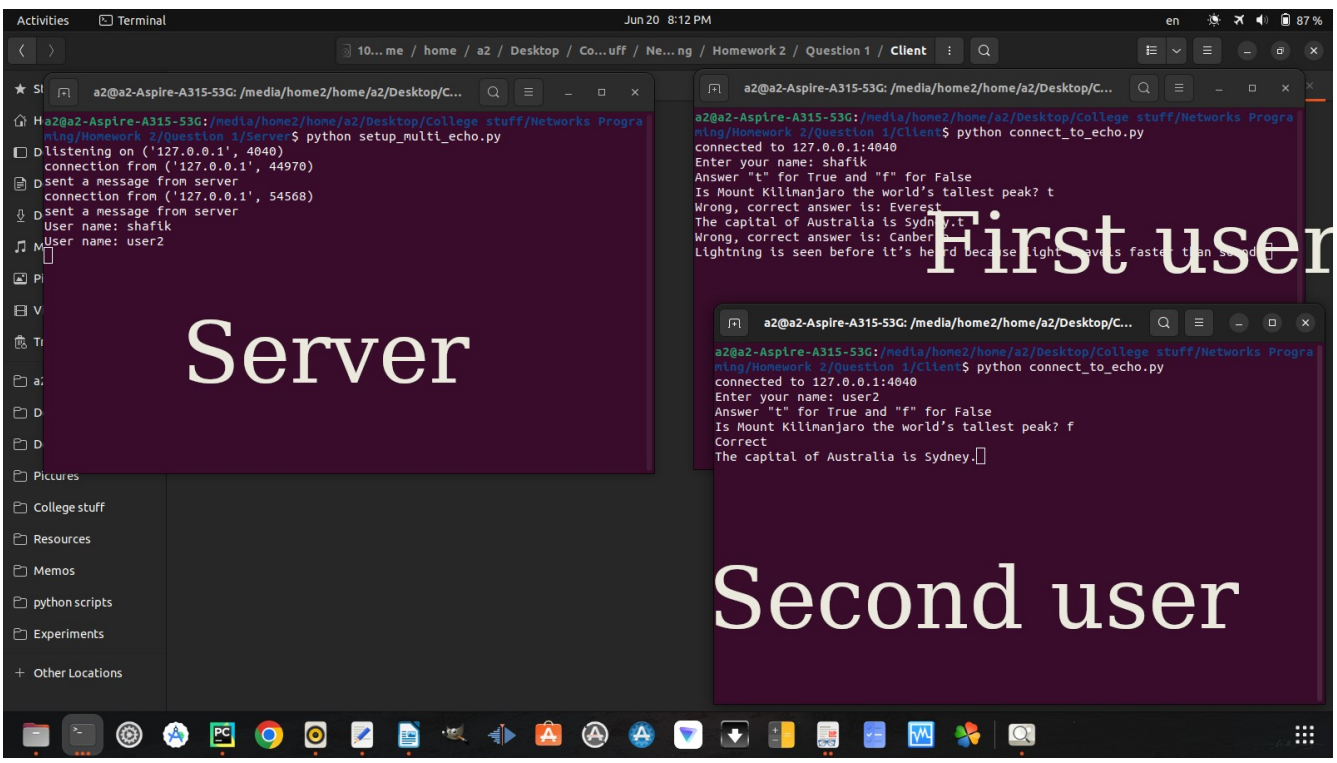
#######################End of File connect_to_echo.py:

## Explanation:

This file is used by client to connect to server and send/receive messages.
Messages containing !!!END!!! means the server is ending the connection.
Messages containing !!!NO_INPUT!!! means the server is making statements.
Messages with any tag requires client input.
The while loop goes on until the server ends the connection, or client ends it by entering ‘q’, or an error occurs.

####################### End of client files

## Result:



# Question 2: Simple website with Python Flask Framework
# Answer :

There are a python file for setting up Flask server and for routing(main,py).
There are two HTML files for two pages(index.html, more.html) inside templates folder.
There are multiple css and javascript files inside static folder, most of them are locally hosted Bootstrap files.

## Code:
### File main.py:

```
from flask import Flask, render_template, request
app = Flask(__name__)

#loading main html page
@app.route('/')
def index():
    return render_template('index.html')

#handling form data
@app.route('/students', methods =["POST"])
def handle_input():
        # getting input with name = student_name in HTML form
        student_name = request.form.get("studentname")
        # getting input with name = student_number in HTML form
        student_number = request.form.get("studentnumber")
        return render_template("more.html", name=student_name)

if __name__ == '__main__' :
    #running flask app
    app.run(port=9999)
```

#######################End of File main.py:

## Explanation:

This file is to setup Flask server at (host=127.0.0.1, port=9999), and to load index.html as main page, and also used for routing.
In this file there’s a value passed from the form in index.html to p tag in more.html that’s used to greet student by name.

## File index.html:

```
<!DOCTYPE html>
<html lang="en">

        <head>

            <meta name="viewport" content="width=device-width initial-scale=1">

            <link rel="stylesheet" type="text/css" href="{{url_for('.static',
filename='css/styles.css')}}">

            <title>Networks Programming Second Homework</title>

        </head>

        <body>

            <script type="text/javascript" src="{{url_for('.static',
filename='js/jquery.min.js')}}"></script>
            <script type="text/javascript" src="{{url_for('.static',
filename='js/bootstrap.bundle.min.js')}}"></script>


                <div class="col-6 col-md-4">


                        <h2>Electronics and Communication Engineering</h2>
                        <div class="container" id="container">
                                <div class="form-container sign-in-container">
                                        <form action="{{ url_for("handle_input")}}"
method="post">
                                                <h1>Sign in</h1>
```

```
                                        <input name="studentname"
placeholder="Name" />
                                        <input name="studentnumber"
placeholder="Number" />
                                        <button>Apply</button>
                                    </form>
                                </div>
                                <div class="overlay-container">
                                    <div class="overlay">
                                        <div class="overlay-panel overlay-right">
                                            <h1>Hello, Student</h1>
                                            <p>Enter your Name and Number.<br>
                                            And proceed at your own risk
                                            </p>

                                        </div>
                                    </div>
                                </div>
                            </div>

                            <footer>
                                <p>Networks Programming Second Homework</p>
                            </footer>

                        </div>

                    </body>
</html>
```

##########################End of File index.html:

## File more.html:

```
<!DOCTYPE html>
<html lang="en">

    <head>

        <meta name="viewport" content="width=device-width initial-scale=1">
```

```
        <link rel="stylesheet" type="text/css" href="{{url_for('.static',
filename='css/styles.css')}}">

        <title>Networks Programming Second Homework</title>

    </head>

    <body>

            <script type="text/javascript" src="{{url_for('.static',
filename='js/jquery.min.js')}}"></script>
            <script type="text/javascript" src="{{url_for('.static',
filename='js/bootstrap.bundle.min.js')}}"></script>


            <div class="col-6 col-md-4">

                    <h2>Hello {{name}}</h2>
                    <h2>Thanks for applying</h2>
                    <h3>Courses will be added soon.</h3>

                <footer>
                    <p>Networks Programming Second Homework</p>
                </footer>

            </div>

    </body>
</html>
```

##########################End of File more.html:

## File styles.css:

```
body {
    background: #f6f5f7;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    font-family: 'Montserrat', sans-serif;
    height: 100vh;
```

```
	margin: -20px 0 50px;
}
h1 {
	font-weight: bold;
	margin: 0;
}

h2 {
	text-align: center;
}
p {
	font-size: 14px;
	font-weight: 100;
	line-height: 20px;
	letter-spacing: 0.5px;
	margin: 20px 0 30px;
}
button {
	border-radius: 20px;
	border: 1px solid #FF4B2B;
	background-color: #FF4B2B;
	color: #FFFFFF;
	font-size: 12px;
	font-weight: bold;
	padding: 12px 45px;
	letter-spacing: 1px;
	text-transform: uppercase;
	transition: transform 80ms ease-in;
}
button:active {
	transform: scale(0.95);
}
button:focus {
	outline: none;
}
button.ghost {
	background-color: transparent;
	border-color: #FFFFFF;
}
form {
	background-color: #FFFFFF;
	display: flex;
	align-items: center;
	justify-content: center;
	flex-direction: column;
	padding: 0 50px;
	height: 100%;
	text-align: center;
}
```

```
input {
	background-color: #eee;
	border: none;
	padding: 12px 15px;
	margin: 8px 0;
	width: 100%;
}
.container {
	background-color: #fff;
	border-radius: 10px;
	box-shadow: 0 14px 28px rgba(0,0,0,0.25),
			0 10px 10px rgba(0,0,0,0.22);
	position: relative;
	overflow: hidden;
	width: 768px;
	max-width: 100%;
	min-height: 480px;
}
.form-container {
	position: absolute;
	top: 0;
	height: 100%;
	transition: all 0.6s ease-in-out;
}
.sign-in-container {
	left: 0;
	width: 50%;
	z-index: 2;
}
.container.right-panel-active .sign-in-container {
	transform: translateX(100%);
}
.overlay-container {
	position: absolute;
	top: 0;
	left: 50%;
	width: 50%;
	height: 100%;
	overflow: hidden;
	transition: transform 0.6s ease-in-out;
	z-index: 100;
}
.container.right-panel-active .overlay-container{
	transform: translateX(-100%);
}
.overlay {
	background: #FF416C;
	background: -webkit-linear-gradient(to right, #FF4B2B, #FF416C);
	background: linear-gradient(to right, #FF4B2B, #FF416C);
```

```
        background-repeat: no-repeat;
        background-size: cover;
        background-position: 0 0;
        color: #FFFFFF;
        position: relative;
        left: -100%;
        height: 100%;
        width: 200%;
        transform: translateX(0);
        transition: transform 0.6s ease-in-out;
}
.container.right-panel-active .overlay {
        transform: translateX(50%);
}
.overlay-panel {
        position: absolute;
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction: column;
        padding: 0 40px;
        text-align: center;
        top: 0;
        height: 100%;
        width: 50%;
        transform: translateX(0);
        transition: transform 0.6s ease-in-out;
}
.container.right-panel-active .overlay-left {
        transform: translateX(0);
}
.overlay-right {
        right: 0;
        transform: translateX(0);
}
.container.right-panel-active .overlay-right {
        transform: translateX(20%);
}
footer {
   background-color: #222;
   color: #fff;
   font-size: 14px;
   bottom: 0;
   position: fixed;
   left: 0;
   right: 0;
   text-align: center;
   z-index: 999;
}
```

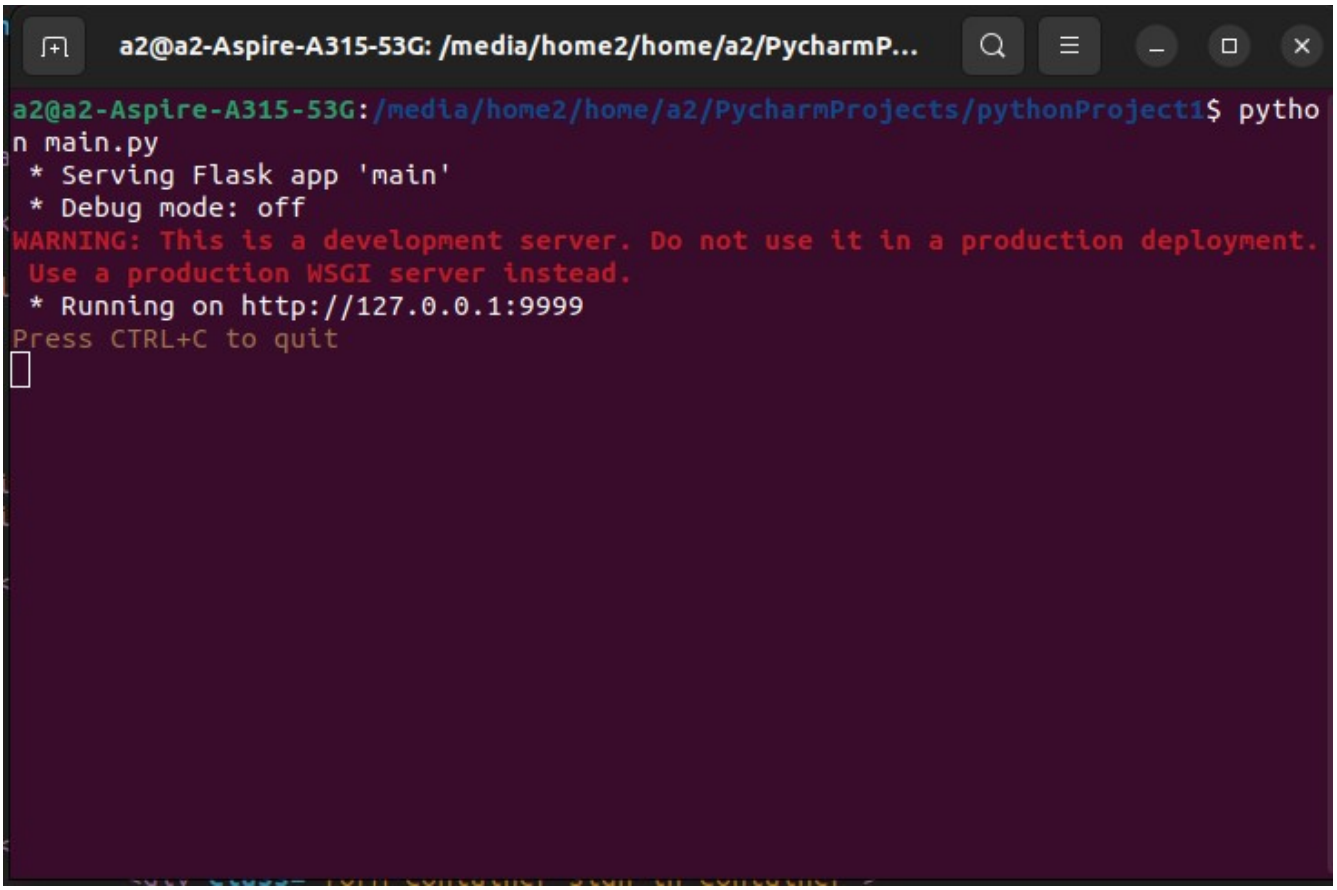
```
footer p {
   margin: 10px 0;
}
```

#######################End of File styles.css:

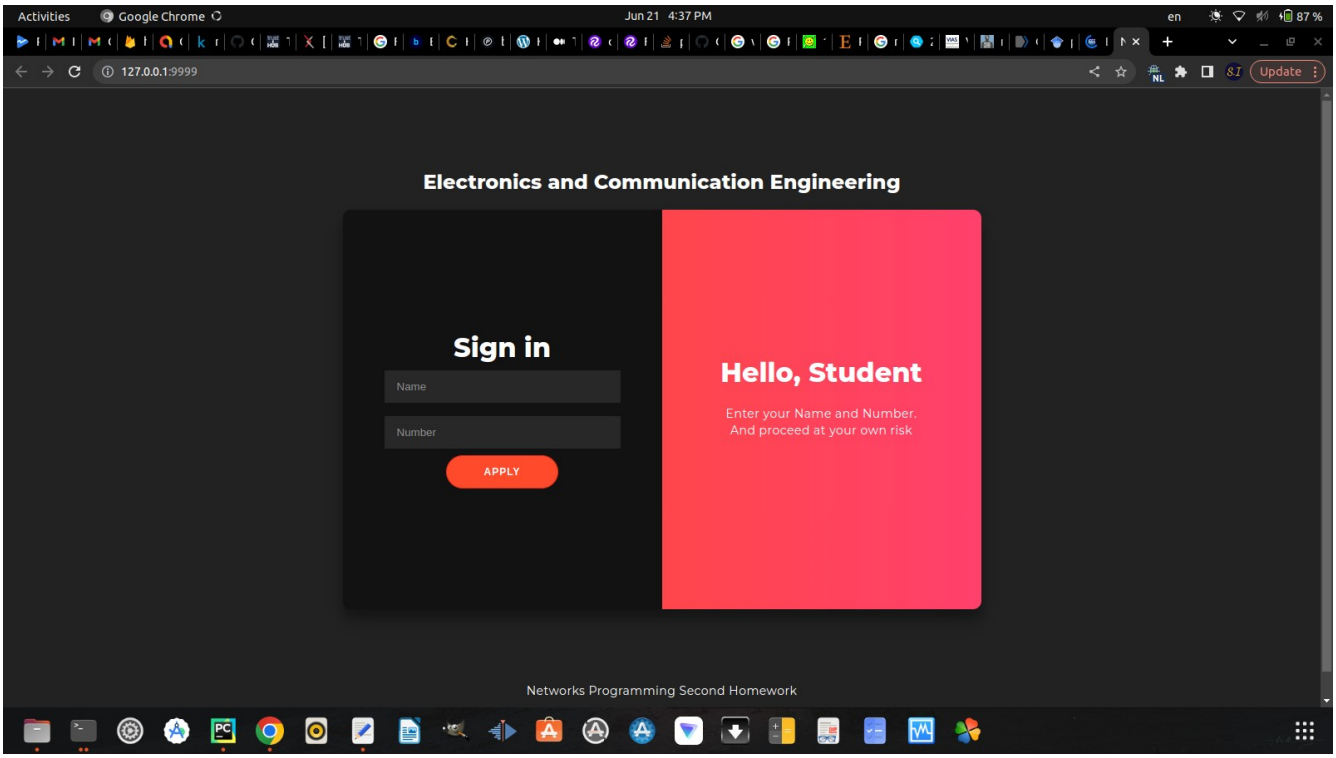
The rest of files in folders (css, js) are locally hosted Bootstrap files.

## Results:

proof of Flask server deployed:



index.html:




more.html: