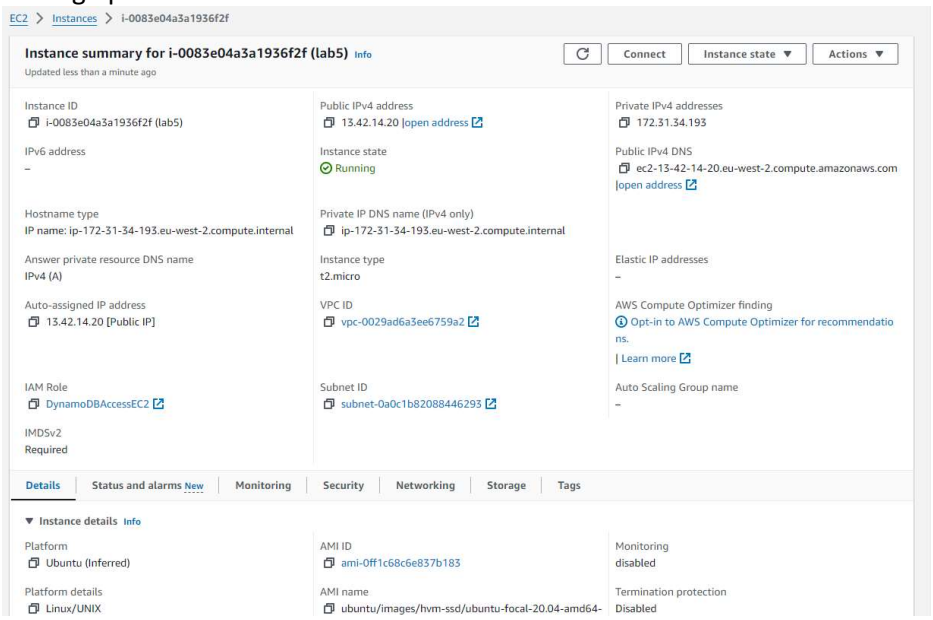
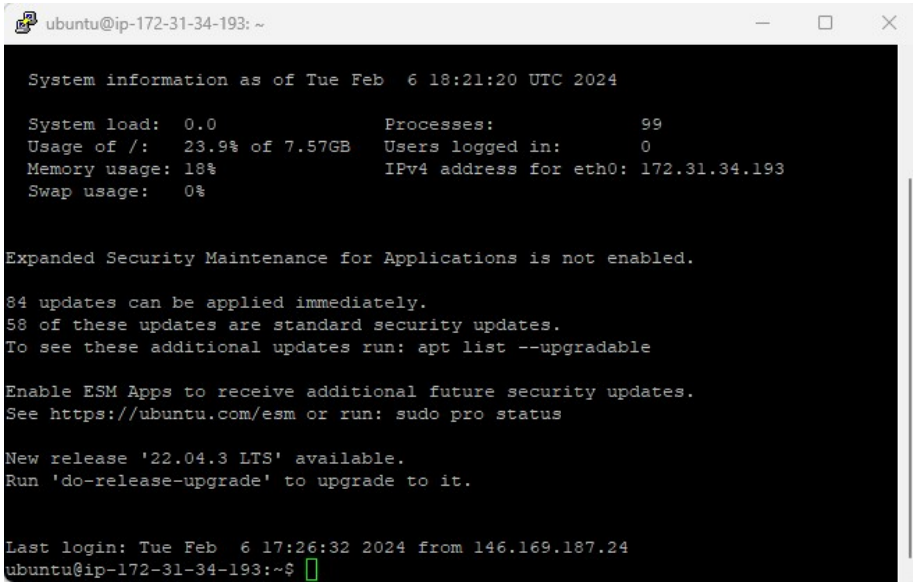
Lab5

06 February 2024 18:18

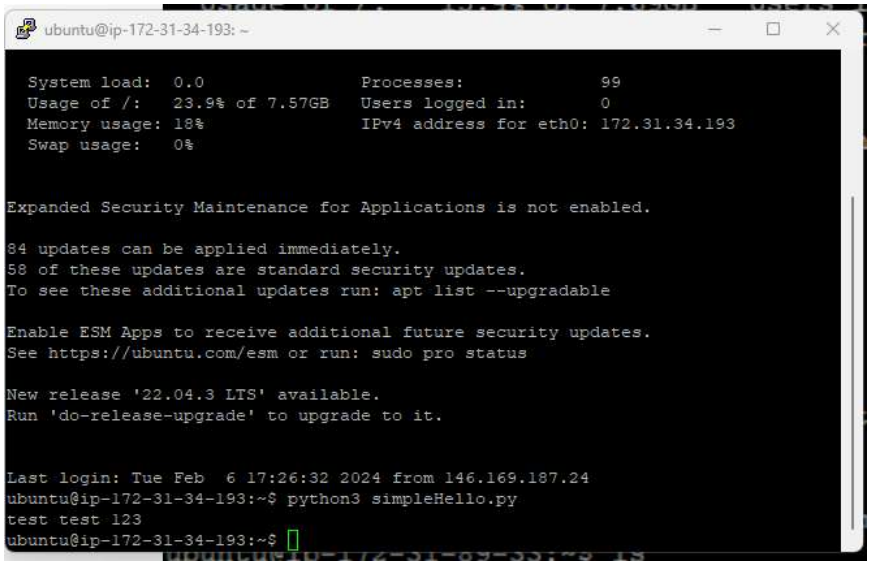
Setting up AWS server



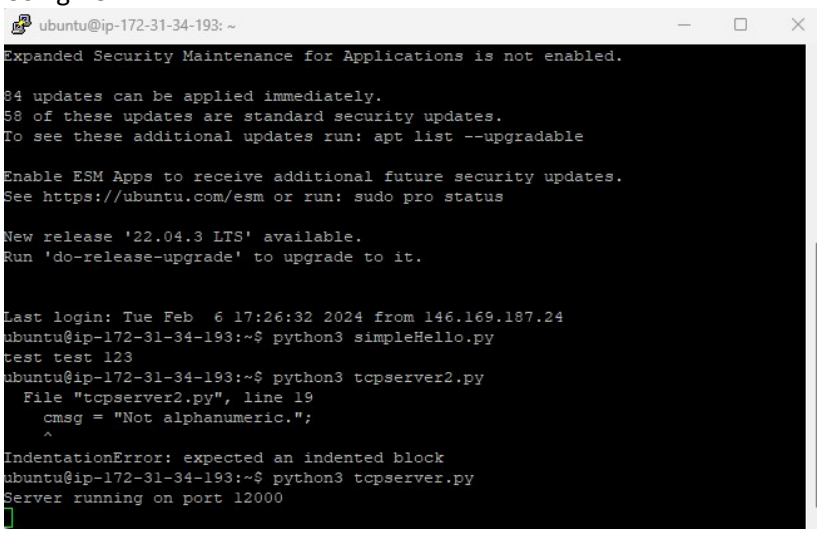
Connecting via putty and installing python



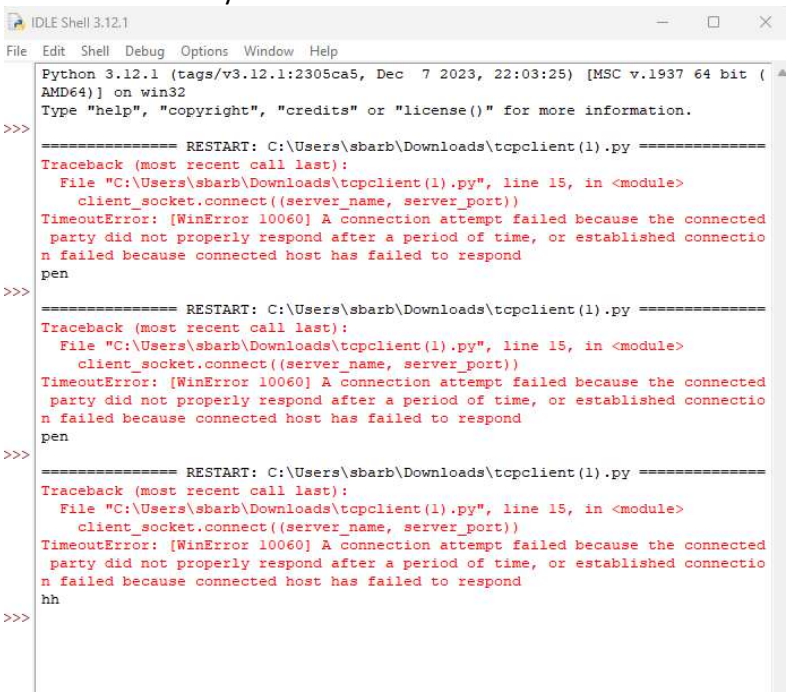
Testing python program



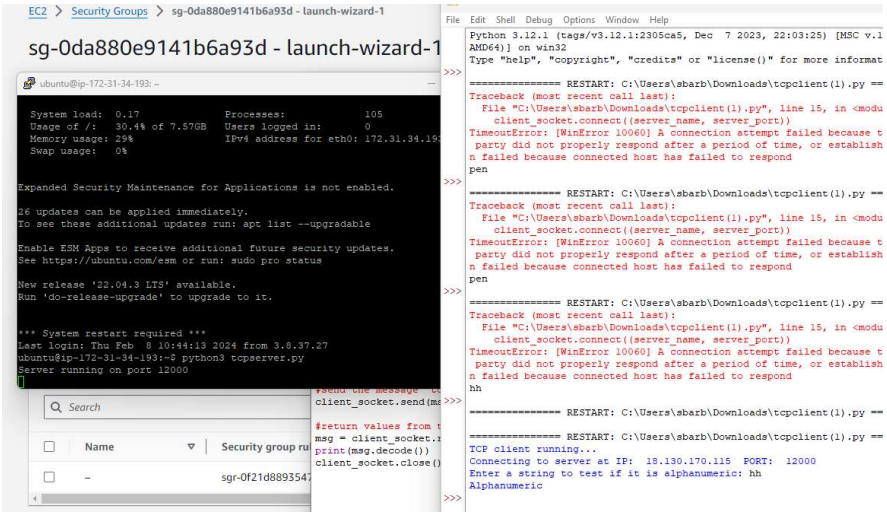
Using TCP:



Didn't work initially



Had to reconfigure outbound TCP security settings in AWS. Somehow reverted to old settings?



Now it works

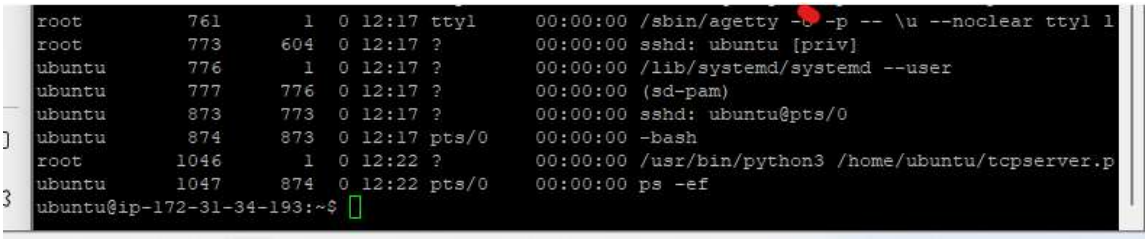
Now we want to make the service run in the background so we don't have to keep the ssh connection live for the program to stay active

[Unit]
Description=TCP server service
After=multi-user.target

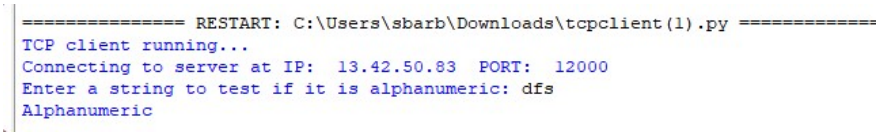
[Service]
Type=simple ExecStart=/usr/bin/python3 /home/ubuntu/tcpserver.py

[Install]
WantedBy=multi-user.target

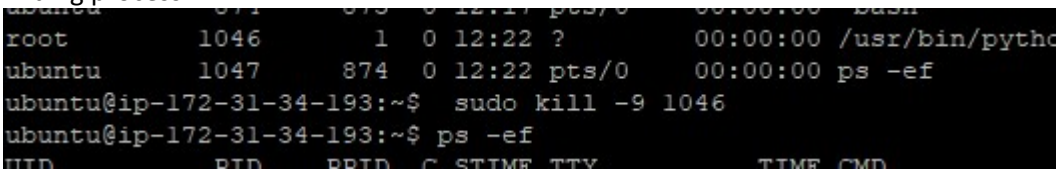
This code sets the program to start on boot



Still works



Ending process



Challenge: Computing RTT (Round Trip time) for 500 communications

RTTClient.py - C:/Users/sbarb/Downloads/RTTClient.py (3.12.1)

File Edit Format Run Options Window Help

```
import socket
import time

SERVER_HOST = '13.42.50.83'
SERVER_PORT = 12000
NUM_COMMUNICATIONS = 500

def tcp_client(data):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.connect((SERVER_HOST, SERVER_PORT))
        start_time = time.time()
        sock.sendall(data.to_bytes(4, byteorder='big'))
        response = sock.recv(1024)
        end_time = time.time()
        rtt = end_time - start_time
        return rtt

if __name__ == "__main__":
    running_average = 0
    for i in range(NUM_COMMUNICATIONS):
        rtt = tcp_client(69)
        running_average = (running_average * i + rtt) / (i + 1)
        print(f"Communication {i+1}/{NUM_COMMUNICATIONS}: RTT = {rtt:.6f}s, Runn
```

RTTServer.py - C:/Users/sbarb/Downloads/RTTServer.py (3.12.1)

File Edit Format Run Options Window Help

```
import socket

SERVER_HOST = '0.0.0.0'
SERVER_PORT = 12000

def tcp_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.bind((SERVER_HOST, SERVER_PORT))
        sock.listen(1)
        while True:
            conn, addr = sock.accept()
            with conn:
                data = conn.recv(1024)
                if not data:
                    break
                conn.sendall(data)

if __name__ == "__main__":
    tcp_server()
```

Client sends an integer and the server sends back whatever integer it received. Time between it being sent and received again is measured and then average RTT is computed by the client.

Basic python code to send communications and compute RTT

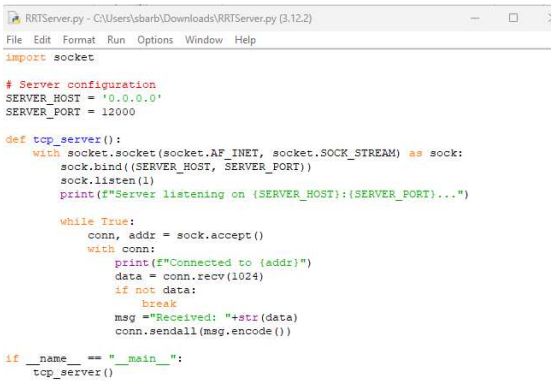
```
Communication 496/500: RTT = 0.010723s, Running Average = 0.011416s
Communication 497/500: RTT = 0.008909s, Running Average = 0.011411s
Communication 498/500: RTT = 0.007024s, Running Average = 0.011402s
Communication 499/500: RTT = 0.013177s, Running Average = 0.011405s
Communication 500/500: RTT = 0.009109s, Running Average = 0.011401s
```

Average RTT was 0.011401 over 500 communications

Sending accelerometer data to server via TCP

File Edit Format Run Options Window Help

```
import subprocess
import socket

SERVER_HOST = '13.42.50.83'
SERVER_PORT = 12000
NUM_COMMUNICATIONS = 500
NIOS_CMD_SHELL_BAT = "C:/intelFPGA_lite/18.1/nios2eds/Nios II Command Shell.bat"
#create a TCP client socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
def send_on_jtag(cmd):
    # check if atleast one character is being sent down
    assert (len(cmd) >= 1), "Please make the cmd a single character"

    # create a subprocess which will run the nios2-terminal
    process = subprocess.Popen(
        NIOS_CMD_SHELL_BAT,
        bufsize=0,
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
    )
    # send the cmd string to the nios2-terminal, read the output and terminate the process
    try:
        vals, err = process.communicate(
            bytes("nios2-terminal <<< {}".format(cmd), "utf-8")
        )
        process.terminate()
    except subprocess.TimeoutExpired:
        vals = "Failed"
        process.terminate()
    return vals
def perform_computation():
    res = send_on_jtag("1") #1 get accelometer value for after filter 1
    # you can process the output here
    print(res)
    return res
if __name__ == "__main__":
    value = perform_computation()
    #create a TCP client socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    #Set up a TCP connection with the server

    #a connection_socket will be assigned to this client on the server side

    client_socket.connect((server_name, server_port))

    print("TCP client running...")
    print("Connecting to server at IP: ", server_name, " PORT: ", server_port)

    #send the message  to the udp server
    client_socket.send(value.encode())

    #return values from the server
    msg = client_socket.recv(1024)
    print(msg.decode())
    client_socket.close()
```



Sends value 500 times to the server. Server responds with 'Received:(Value received)'