Technical Task "Client-Server"

GiTHUB Repo link:

https://github.com/Ahmed17Said06/5G_PCSE_Technical_Task.git

Design:

Server:

1- Simulation function generate bounded random number to simulate temperature degree readings.
2- Multi-threaded server uses the simulation function each one second to get new reading, and has one non-blocking socket always exposed to listen to any incoming join requests from clients.
3- Once server detects a request on the listening port, it spin a new socket to handle the communication with the concerned client. "system allows multi-clients simultaneously".
4- Server running in while infinite loop, keeps listening and generating the degrees.
5- Once it has a connection it spin new socket and save it in a vector each second it spin on the vector sockets to send them the readings.
6- If server dies the connection to all clients stops, and each client print that the server is lost.
7- If client went down, server keeps running, no matter there is clients or not.
8- Environment Ubuntu 22.04 LTS VM on VMware Player, and VS code.

Client:

1- Multi-threaded client, client listens on port "5400" on the local host.
2- Client saves the readings for the last 5 seconds into buffer, then do its calculations on it and print it.
3- If no server found it prints no there is no server and terminate as there is no point to keep looping without having a server to listen for.
4- If client is terminated, other clients keep working normally without interruption.
5- Client does the 2 average and accumulation calculations in two different threads.
6- Once it detects that the server dies, it closes the socket immediately, print that server lost and halt its operations.
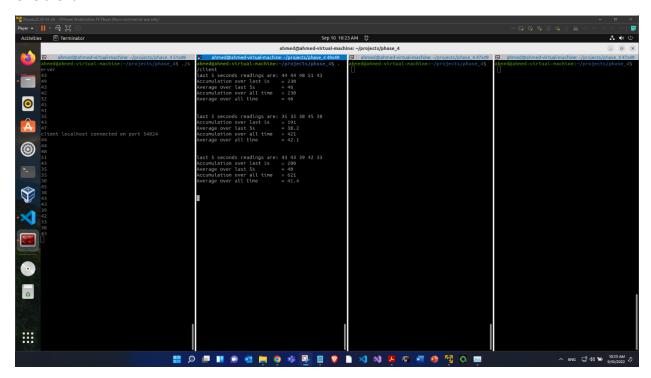
Network:

1- Connection is TCP/IP socket "BSD sockets" on localhost on port "5400".
2- Can be changed easily to UDP by changing the related protocol on the TCP of BSD socket functions.
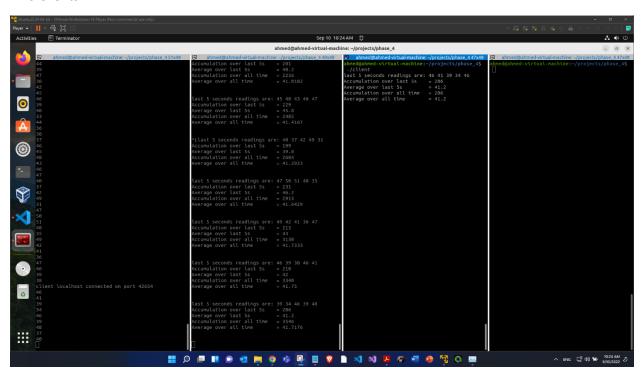3- Network handle multiple clients.

Containerized the solution:

1- Under source files there is Docker directory, that has client directory, server directory and docker-compose yaml file.
2- Wrapped the server and the clients each on separate docker file.
3- Then used docker compose to spin the 2 containers and to handle that the both are connected on localhost ports 54000.

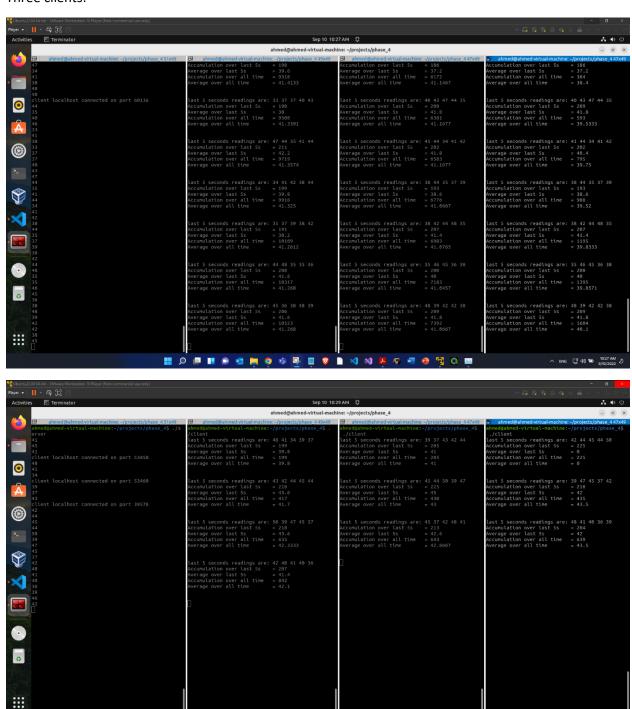## Snippets from the output conveying the requirements:
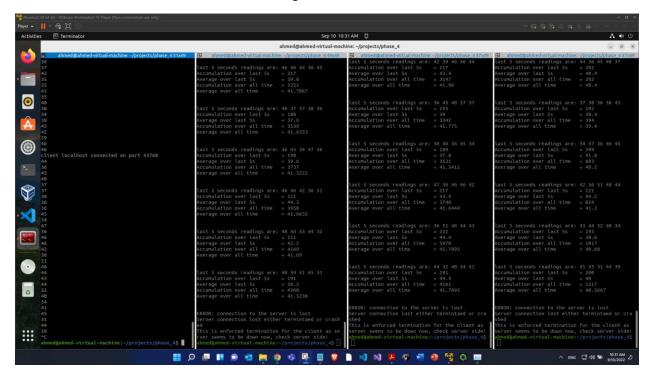
One client:

Two Clients:

Three clients:
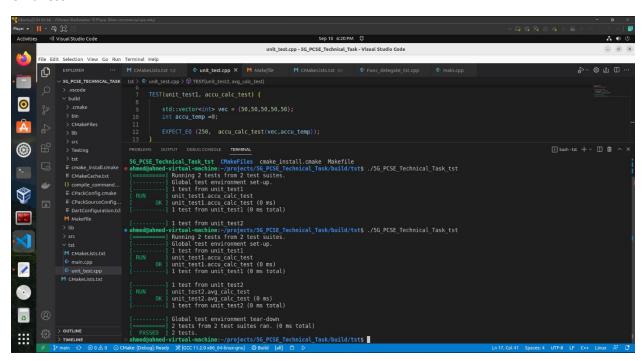
Server Shutdown terminates the client listening.

Unit Test:

Containerization (Docker and Docker Compose):