5G Platform Software Engineer Task

Forest Fire Forecasting

Habiba Mounir El-Bakry
DELL TECHNOLOGIES

1 INTRODUCTION

It is required to implement a forest fire forecasting system. The system shall gather data about the temperature from the sensory node located in the forest. This data can then be analyzed to prevent possible disasters.

2 REQUIREMENTS SPECIFICATION

The system must follow certain functional and non-functional requirements.

2.1 Functional Requirements

- The sensory node will be acting as server sending temperature grades (Celsius), each one second to a client.
- The client shall calculate the following, as it gets the data:
  - Average over time.
  - Accumulation over time.
- The client shall print the calculations each 5 seconds.

2.2 Non-Functional Requirements

- The implementation utilizes an automated build system.
- The code shall use VCS (version control system).
- The code shall include tests.

2.3 Assumptions

Accumulation over time is calculated by summing the received temperature grades. Average over time is calculated by dividing this accumulation by the number of readings. As no period is specified for the accumulation, the variable will eventually overflow. This situation isn't handled in the implementation.

3 DESIGN OVERVIEW

The system is composed of two processes. One process is for the server-side (sensory nodes) while the other is for client-side. Unix Sockets are mainly used to handle communication between these processes. The design supports the following aspects:

1. The system can allow multiple clients.
2. The server-side and the client-side processes use multithreading.
3. An abstraction layer is added to support multiple communication methods in the future with minimum effort.

3.1 Design Paradigm

Object oriented design is used which can be represented by the following class diagram.

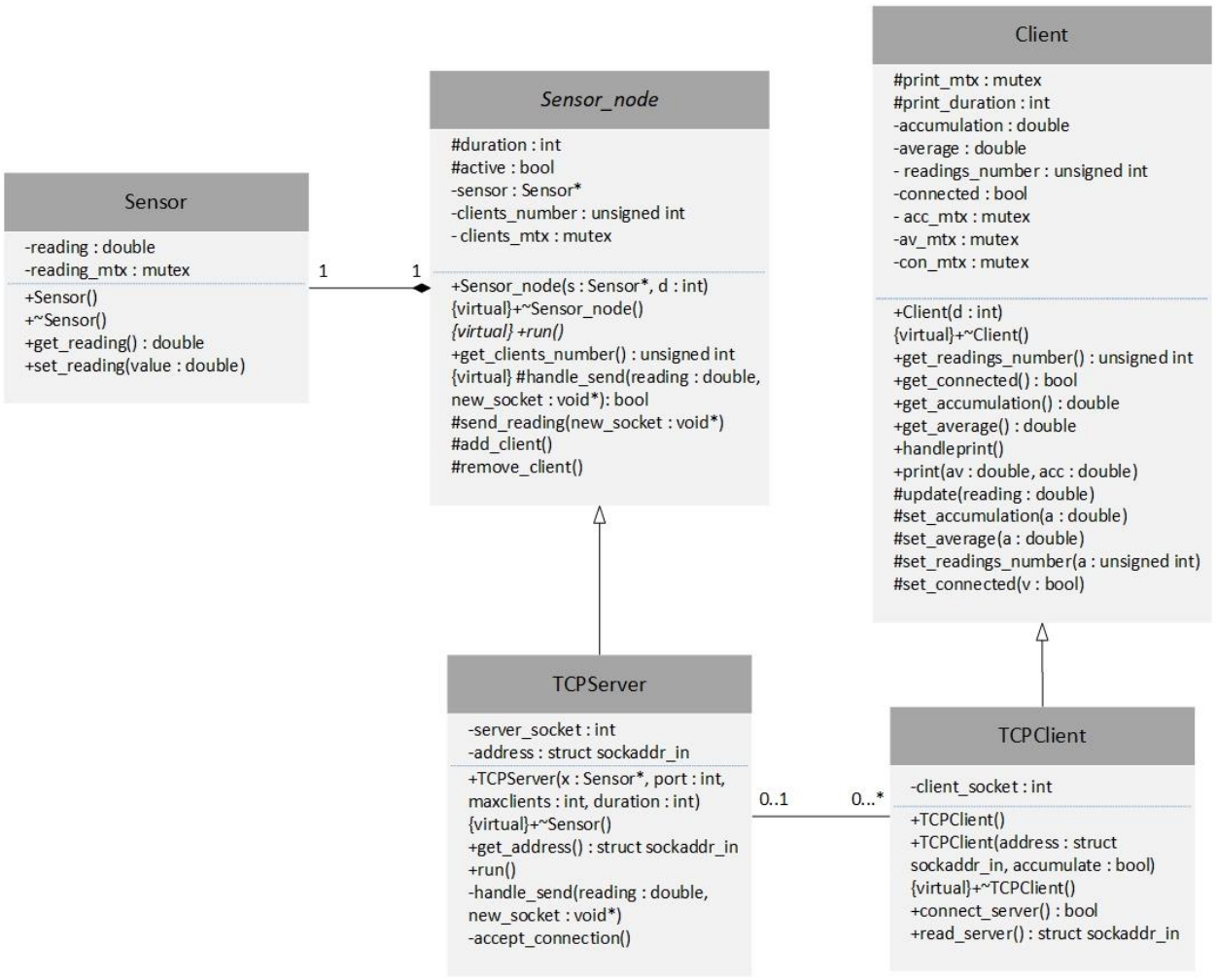

Figure 1: Class Diagram

In the following section we shall explain each of these classes.

3.1.1 Sensor class

It is used to mimic a real sensor by holding the sensor reading. As the project uses multithreading, mutex locks are used to ensure the atomic read/write operation on the shared variables between threads. Therefore, the sensor reading has an associated reading mutex. The class also includes operations to set and get the reading with the help of that mutex lock.

3.1.2 Sensor_node class

The class is composed of a Sensor instance that the Sensor_node is attached to. The Sensor_node will sample the Sensor reading to send it to the clients periodically according to the duration member value (this value is set to 1 in the problem statement). As sending the reading to the clients depends on the communication method utilized, this class will define pure virtual functions that are responsible for handling the communication (connect, send, receive, and close). Thus, the Sensor_node class helps support multiple communication methods later by implementing the common functionalities between these methods and allowing them to extend it.

3.1.3 TCPServer class

Multiple communication methods exist. In this project, blocking TCP/IP LINUX/UNIX communication sockets are used. As they are blocking, multithreading is needed. When the server starts its service using the run function, it creates a thread. This thread executes accept_connection member function. The function is responsible for accepting new clients' connections. For each accepted connection, a thread is created to execute the handle_send function defined in the Sensor_node class to send the reading to the connected client. The handle_send function has a pointer to void parameter that can carry extra data necessary for sending (may differ in different protocols). In TCP/IP, this parameter is the file descriptor of the socket.

3.1.4 Client class

Similar to the Sensor_node class in the server-side, this class implements the common functionalities between the different clients communication methods. It saves the calculated accumulation and average values, provides functions for setting and getting them using mutex locks, and handles printing them each period defined by print_duration (this value is set to 5 in the problem statement).

3.1.5 TCPClient class

As TCPServer class, this class uses the blocking TCP/IP LINUX/UNIX communication sockets. It handles the connection to a server and reading data from a server. As mentioned in TCPServer, blocking communication needs multithreading. On connection, two threads are created. The first thread executes read_server function which is responsible for reading the server data. The second thread executes handleprint function that is defined in the Client base class.

4 IMPLEMENTATION CHOICES

This section introduces various corner cases and how they are handled

4.1 Server-side Process

- On failure to create the socket or bind it to an address, the process exits.
- On failure to send data to the client, the server closes that client connection.
- A maximum of 5 clients can be queued waiting for their connection to be accepted.

4.2 Client-side Process

- The Client can't connect to more than one server at a time.
- The Client can connect to other server if its current connection is closed.
- A timeout period is specified for receiving data. If timeout occurs, the connection is closed. The timeout period is assumed to be 5 seconds.
- The print operation terminates if timeout occurs.

5 TESTING

Twelve test cases are implemented with the help of google test framework. Their results are shown in the following figure.

```
Test project /mnt/d/Fourth year/Second term/Dell/Fire_forecast/build
      Start  1: OneClient1.ConnectionDoneSuccessfully
 1/12 Test  #1: OneClient1.ConnectionDoneSuccessfully .....................   Passed    7.05 sec
      Start  2: OneClient2.DataReceivedSuccessfully
 2/12 Test  #2: OneClient2.DataReceivedSuccessfully .......................   Passed    3.54 sec
      Start  3: OneClient3.AccumulationAndAverageUpdatedSuccessfully
 3/12 Test  #3: OneClient3.AccumulationAndAverageUpdatedSuccessfully ......   Passed    7.03 sec
      Start  4: OneClient4.ServerShutDownAndClientDisconnectAfterTimeOut
 4/12 Test  #4: OneClient4.ServerShutDownAndClientDisconnectAfterTimeOut ...   Passed   10.06 sec
      Start  5: OneClient5.ClientConnectToOtherServer
 5/12 Test  #5: OneClient5.ClientConnectToOtherServer .....................   Passed   28.06 sec
      Start  6: OneClient6.ClientCannotConnectToOtherServerIfItIsConnected
 6/12 Test  #6: OneClient6.ClientCannotConnectToOtherServerIfItIsConnected ...   Passed   11.05 sec
      Start  7: OneClient7.ServerRemoveTheClientFromTheListWhenItClosesTheSocket
 7/12 Test  #7: OneClient7.ServerRemoveTheClientFromTheListWhenItClosesTheSocket ....   Passed    8.05 sec
      Start  8: OneClient8.ClientConnectionFailOnConnectingToServerUsingWrongAddress
 8/12 Test  #8: OneClient8.ClientConnectionFailOnConnectingToServerUsingWrongAddress ....   Passed    2.05 sec
      Start  9: MoreThanOneClient1.TwoClientsConnectedSuccessfullyToServer
 9/12 Test  #9: MoreThanOneClient1.TwoClientsConnectedSuccessfullyToServer ...   Passed    5.04 sec
      Start 10: MoreThanOneClient2.OneClientDisconnectButServerAndOtherClientsNotAffected
10/12 Test #10: MoreThanOneClient2.OneClientDisconnectButServerAndOtherClientsNotAffected ...   Passed   24.04 sec
      Start 11: MoreThanOneClient3.AccumulationAndAverageUpdatedSuccessfully
11/12 Test #11: MoreThanOneClient3.AccumulationAndAverageUpdatedSuccessfully .....   Passed    6.06 sec
      Start 12: MoreThanOneClient4.ServerShutDownAndClientsDisconnectAfterTimeOut
12/12 Test #12: MoreThanOneClient4.ServerShutDownAndClientsDisconnectAfterTimeOut ...   Passed   23.05 sec

100% tests passed, 0 tests failed out of 12

Total Test time (real) = 135.12 sec
```

Figure 2: Tests Result

6 APPENDIX

- The implementation can be found in the following github link:

  https://github.com/Habiba1998/Fire_forecast.git

- The server process and client process are containerized and are pushed to docker hub.
  Server process image: https://hub.docker.com/repository/docker/2724180806/server
  Client process image: https://hub.docker.com/repository/docker/2724180806/client