Machine Problem 2

Sensor Network Data Logging

Created by: Wilbert Jethro R. Limjoco and Dale Joshua R. Del Carmen

General Information

- This Machine Problem is to be done individually.
- Deadline is on the week of December 7-10 (Finals Week).
- Checking will be on the day of the deadline. You will be given a maximum of 10 minutes to present your Machine Problem. Comment out code that will break your MP and show it to your instructor during the demo.

Motivation

Sensors are devices that convert physical stimuli such as temperature, pressure, alcohol concentration, wind speed, light, and heart rate to name a few, into electrical signals. These electrical signals are then converted into digital values for monitoring, detection, storage, computing, and automation purposes. Connecting these sensors in a computer network allows us to have technologies such as Smart Grids, Real-time Health Monitoring, Driverless Cars, Automated Homes, and other “Smart Things”.

In this this machine problem, you will design a simple *sensor network*. You will simulate collecting data from multiple sensors and send them to a server (also called a “sink”), and respond accordingly to abnormal data.

Problem

You are to implement a sensor network with one server and multiple clients. Connections are established using standard TCP. Everything aside from the bonus will be done purely in Linux/OS X terminals.

The clients will read a randomly generated file (to be provided) at a rate of 1 data line per second. This file contains randomly generated numbers that will serve as a fake or sample data set read by a sensor. It will then send this data to the server. If a client detects that the data is past a certain threshold, it will experience a failure (think of it as a sensor breaking due to extreme conditions); that is it will immediately terminate without informing the server.

The server will simultaneously receive the data sent by all the clients and log them into temporary files or memory. The server should be able to identify clients (sensor identifier and IP address & port number) to determine which data is coming from which client. If the data is within a certain threshold, the server will terminate the client which sent the data and save the log of the terminated client into a text file with the client identifier.

Terminating the server (through CTRL + C), should close all connections and terminate all clients before terminating the server process itself.

The client program will be tested on at least two different computers on the same network.

Server settings such as port number and termination threshold should be specifiable via the program parameter. Client settings such as server address (IP address & port), failure threshold, and sensor number (-s option) should also be specifiable via the program parameter. Thresholds should be modifiable with -l option for less than or -g option for greater than.

Bonus

Graph all data that the server receives from each client individually using any programming language, API, and GUI.

Makefile

You should use a makefile that automatically compiles all your source codes.

Sample Execution

Sensor data:

Temperature in Celsius from 10 different sensor nodes.

See sensor-data.zip in UVLe.

Server sample run:

```
$linux@PC1:~$ ./MP2_Server 1234 -l 26.0
PID 6234
Waiting for connections...
Sensor 1 (192.168.1.13:14123) connected
Data Received: (1) 31.63
Sensor 3 (192.168.1.13:14125)  connected
Data Received: (1) 32.38
Data Received: (3) 32.21
Sensor 2 (192.168.1.8:5633) connected
Data Received: (1) 32.52
Data Received: (3) 31.30
Data Received: (2) 26.19
Sensor 6 (192.168.1.8:5634) connected
Data Received: (1) 32.04
Data Received: (3) 30.71
Data Received: (2) 26.65
Data Received: (6) 34.85
Data Received: (1) 32.56
Data Received: (3) 31.20
Data Received: (2) 27.74
Data Received: (6) 34.82
```

```
Data Received: (1) 31.10
Data Received: (3) 31.18
Data Received: (2) 27.64
Data Received: (6) 34.89
...
```

In this example, the port number is specified immediately after the program name; meanwhile a threshold of less than 26 is set. This means that if a client sends a data less than 26, the server will terminate the corresponding client. Two clients (using sensor data 1 and 3) connects from PC1 (same PC with server; local IP: 192.168.1.13) while another two clients (using sensor 2 and 6) connects from PC2 (local IP: 192.168.1.8). The server starts receiving data from the sensors as soon as one client connects to the server.

Sensor 6 fails

```
...
Data Received: (1) 31.39
Data Received: (3) 29.16
Data Received: (2) 28.24
Data Received: (6) 39.62
Data Received: (1) 31.76
Data Received: (3) 30.06
Data Received: (2) 28.02
Data Received: (6) 39.91
Sensor 6 (192.168.1.8:5698) disconnected
Sensor 6 data written to sensor-06.log
Data Received: (1) 29.99
Data Received: (3) 30.05
Data Received: (2) 27.39
...
```

Sensor 6 receives a data greater than 40 (set using the client program), so it disconnects before being able to send the data. This simulates a sensor failure.

Sensor 2 is terminated by server

```
...
Data Received: (1) 31.26
Data Received: (3) 29.93
Data Received: (2) 26.96
Data Received: (1) 32.43
Data Received: (3) 29.91
Data Received: (2) 25.35
Sensor 2 (192.168.1.8:5633) to be terminated
Sensor 2 data written to sensor-02.log
Data Received: (1) 33.32
Data Received: (3) 29.94
Data Received: (1) 33.32
```

```
Data Received: (3) 30.13
...
```

Sensor 2 is terminated by the server. This simulates a response to abnormal sensor data.

Client sample run:

on PC1 (local IP is 192.168.1.13)

```
$linux@PC1:~$ ./MP2_Client 192.168.1.13:1234 -s 1 -g 40.0 &
PID 6240
Sampling data from sensor-01.txt
$linux@PC1:~$ ./MP2_Client 192.168.1.13:1234 -s 3 -g 40.0 &
PID 6241
Sampling data from sensor-03.txt
$linux@PC1:~$
```

on PC2 (local IP is 192.168.1.8)

```
$linux@PC2:~$ ./MP2_Client 192.168.1.13:1234 -s 2 -g 40.0 &
PID 4634
Sampling data from sensor-02.txt
$linux@PC2:~$ ./MP2_Client 192.168.1.13:1234 -s 6 -g 40.0 &
PID 4635
Sampling data from sensor-06.txt
$linux@PC2:~$
```

Documentation

- Introduction
  - Introduce and explain the basic concepts involved in the Machine Problem.
- Implementation
  - Discuss how you implemented the system.
  - You may create flowcharts or block diagrams to discuss your implementation.
- Problems Encountered
  - List down all the problems that you have encountered, then explain how you solved those problems (if you were not able to solve the problem, give an idea on how to solve it)
- Create a soft copy only. Submit this in UVLe before the checking day ends.

Grading System

Specification	Points (%)
Makefile	1 (5%)
Error Handling for Program Arguments	2 (10%)
Establish TCP connection across multiple machines	2 (10%)
Clients sending data every second	1 (5%)
Server generating and printing data logs	4 (20%)

Client terminates upon breaking	2 (10%)
Server terminates client after an event	2 (10%)
SIGINT handling for the server	3 (15%)
SIGINT handling for the client	1 (5%)
Bonus: Data graph	4 (20%)
Documentation	2 (10%)
TOTAL	**24 points (120%)**