KU LEUVEN

GROUP T LEUVEN CAMPUS

Operating Systems

Project labs: step 1

Project overview and File I/O

Bachelor Electronics/ICT

Course coordinator: Bert Lagaisse

Lab coaches:

- Ludo Bruynseels
- Jonas Verbruggen
- Toon Dehaene

Last update: november 4, 2023

Project labs: step 1 – Project overview and File I/O

Lab target 1: Initiation to the project overview of the final assignment.

Lab target 2: Learn how to read and write binary and text files.

1. Project overview

The picture below visually sketches the final assignment of this course. It depicts a server process that accepts incoming network connections from multiple client devices that upload sensor data with room temperatures. The connection manager puts the incoming data into a shared data structure. The data manager processes the incoming data and compares the data with a running average of each room. The storage manager is responsible for persisting the data into a comma-separated file (csv file). Important events and especially errors are tracked by a logging process into a log file.

The relationship of this lab to the final assignment is indicated by the dashed blue line.

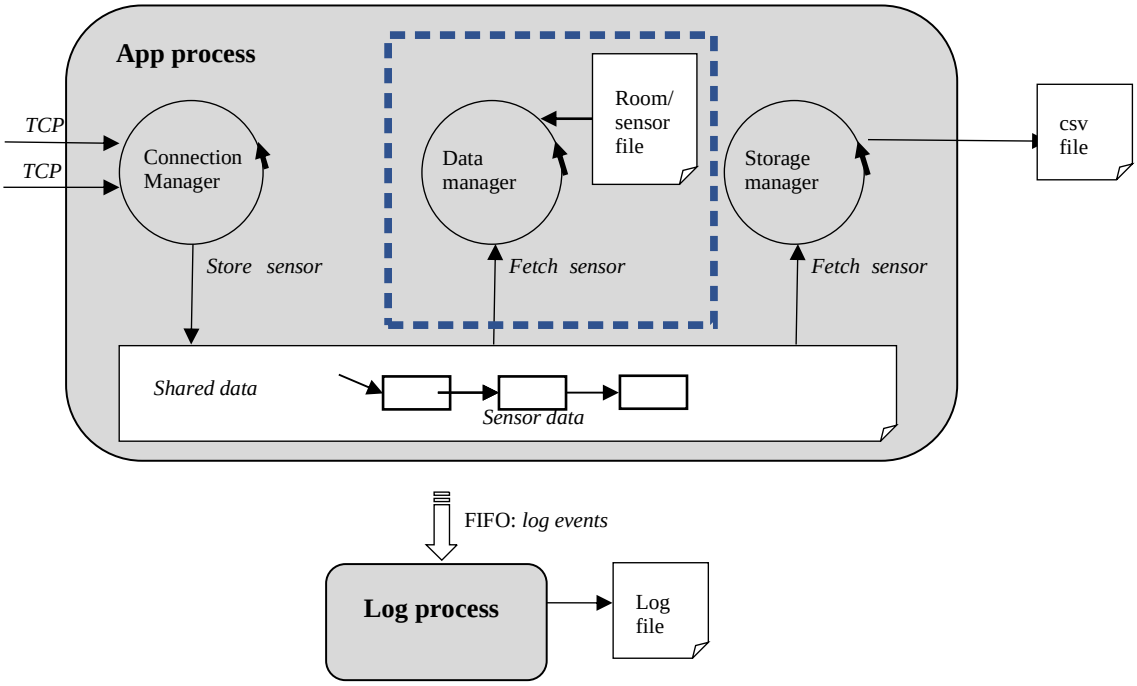

Figure 1: Overview of the full project.

2. The data manager

Assume that a sensor network is used to monitor the temperature of all rooms in an office building. Write a program, called the 'data manager', that collects sensor data and implements the sensor system intelligence. For example, the data manager could apply decision logic to control a heating installation.

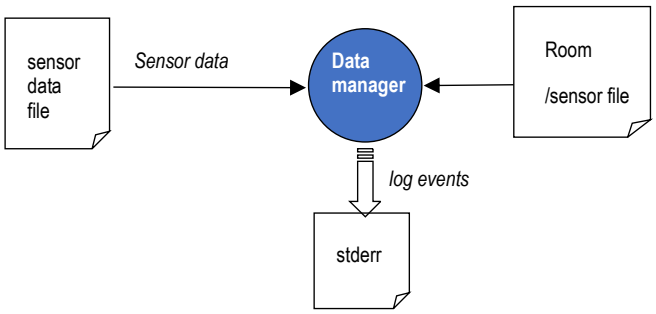

Figure 2: the datamanager.

Before handling sensor data, the data manager should first read a file called 'room_sensor.map' containing all room - sensor node mappings. The file is a text file (i.e. you can open and modify the file in a standard editor) with every line having the format:

```
<room ID><space><sensor ID><\n>
```

A room ID and sensor ID are both positive 16-bit integers (uint16_t).

The data manager organizes all sensor nodes in a **pointer list** data structure. Use the implementation of a pointer list implemented in the previous exercises to do this. An element in this list maintains **at least** information on (i) sensor node ID, (ii) room ID, (iii) data to compute a running average, and (iv) a last-modified timestamp that contains the timestamp of the last received sensor data used to update the running average of this sensor. The picture below visualizes this data structure.

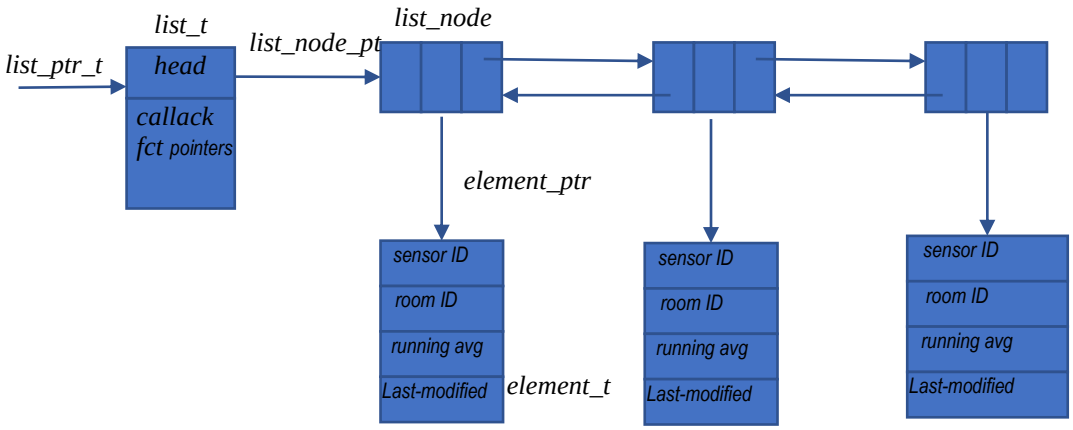

Figure 3: dp list.

The data manager starts collecting sensor data and computes for every sensor node a running average. We define a running average in this context as the average of the last RUN_AVG_LENGTH sensor values. If this running average exceeds a minimum or maximum temperature value, a log-event (message sent to stderr) should be generated

indicating in which room it's too cold or too hot.

RUN_AVG_LENGTH should be set at compile-time with the preprocessor directives RUN_AVG_LENGTH=<some_value>. If this isn't done, the default value of 5 should be used.

It should also be possible to define the minimum and maximum temperature values at compile-time with the preprocessor directives SET_MIN_TEMP=<some_value> and SET_MAX_TEMP=<some_value> where <some_value> is expressed in degrees Celsius. Compilation should fail and an appropriate error message should be printed when these preprocessor directives are not set at compile-time. Sensor data is defined as a struct with the following fields:

- sensor_id: a positive 16-bit integer;
- temperature: a double;
- timestamp: a time_t value;

The data manager reads sensor data from a binary file called 'sensor_data' with the format:

```
<sensor ID><temperature><timestamp><sensor ID><temperature><timestamp> ...
```

Notice this is a binary file, not readable in a text editor, and spaces and newlines (\n) have no meaning. *Read this file single value by single value, not struct by struct*.

Hint 1: A separate program 'file_creator' is given to generate the 'room_sensor.map' and 'sensor_data' files to test your program. If you compile this program with the '-DDEBUG' flag on, the sensor data is also printed to a text file. You may assume that the sensor data on file is sorted on the timestamps (earliest sensor data first). Sensor data of a sensor ID that did not occur in the room_sensor.map file is ignored, but an appropriate message is logged.

Hint 2: Finally, organize your code wisely in a main.c, a datamgr.c and a datamgr.h file. This will help you to easily re-use the data manager code for the final assignment!