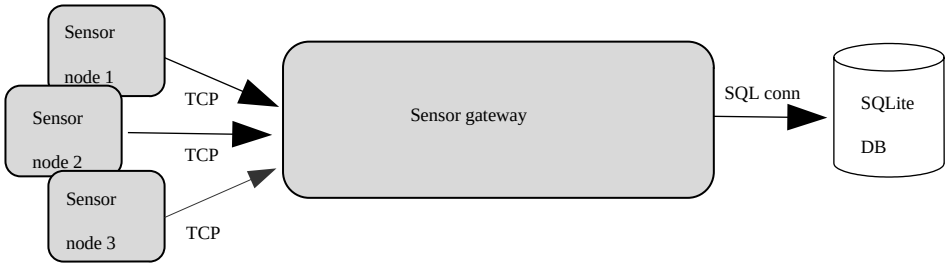
Academic Integrity

*This is an **individual** assignment! It is only allowed to submit **your own work**. You may discuss the assignment with others but you are not allowed to share work or use (part of) another's solution. If you include work (e.g. code, technical solutions,...) from external sources (internet, books …) into your solution, you must clearly indicated this in your solution and cite the original source. If two students present very similar solutions, no distinction will be made between the 'maker' and the 'copier'.*

Sensor Monitoring System

The sensor monitoring system consists of sensor nodes measuring the room temperature, a sensor gateway that acquires all sensor data from the sensor nodes, and an SQL database to store all sensor data processed by the sensor gateway. A sensor node uses a private TCP connection to transfer the sensor data to the sensor gateway. The SQL database is an SQLite system (see lab 7). The full system is depicted below.

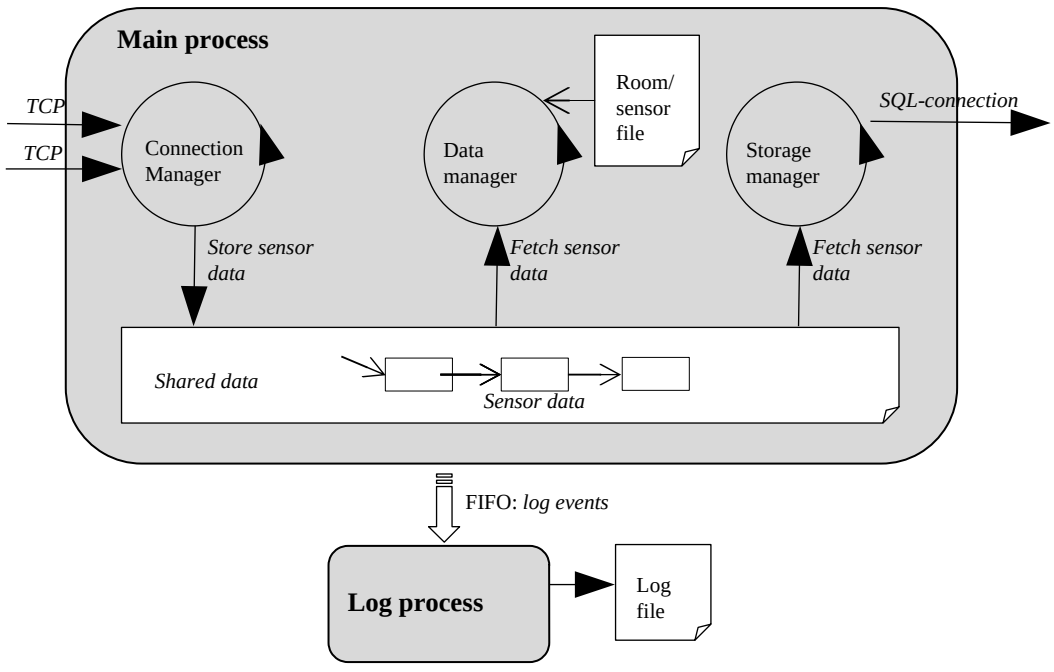


The sensor gateway may **not** assume a maximum amount of sensors at start up. In fact, the number of sensors connecting to the sensor gateway is not constant and may change over time.

Working with real embedded sensor nodes is not an option for this assignment. Therefore, sensor nodes will be simulated in software (see lab 7).

Sensor Gateway

A more detailed design of the sensor gateway is depicted below. In what follows, we will discuss the minimal requirements of both processes in more detail.



Minimal requirements

Req 1. The sensor gateway consists of a main process and a log process. The log process is started (with fork) as a child process of the main process.

Req 2. The main process runs three threads: the connection, the data, and the storage manager thread. A shared data structure (see lab 8) is used for communication between all threads. Notice that read/write/update-access to the shared data needs to be *thread-safe*!

Req 3. The connection manager listens on a TCP socket for incoming connection requests from **new** sensor nodes. The port number of this TCP connection is given as a command line argument at start-up of the main process, e.g. ./server 1234

Req 4. The connection manager captures incoming packets of sensor nodes as defined in lab 7. Next, the connection manager writes the data to the shared data structure.

Req 5. The data manager thread implements the sensor gateway intelligence as defined in lab 5. In short, it reads sensor measurements from shared data, calculates a running average on the temperature and uses that result to decide on 'too hot/cold'. It doesn't write the running average values to the shared data – it only uses them for internal decision taking.

Req 6. The storage manager thread reads sensor measurements from the shared data structure and inserts them in the SQL database (see lab 6). If the connection to the SQL database fails, the storage manager will wait a bit before trying again. The sensor measurements will stay in shared data until the connection to the database is working again. If the connection did not succeed after 3 attempts, the gateway will close.

Req 7. The log process receives log-events from the main process using a FIFO called "logFifo". If this FIFO doesn't exists at startup of the main or log process, then it will be created by one of the processes. All threads of the main process can generate log-events and write these log-events to the FIFO. This means that the FIFO is shared by multiple threads and, hence, access to the FIFO must be thread-safe.

Req 8. A log-event contains an ASCII info message describing the type of event. For each log-event received, the log process writes an ASCII message of the format <sequence number> <timestamp> <log-event info message> to a new line on a log file called "gateway.log".

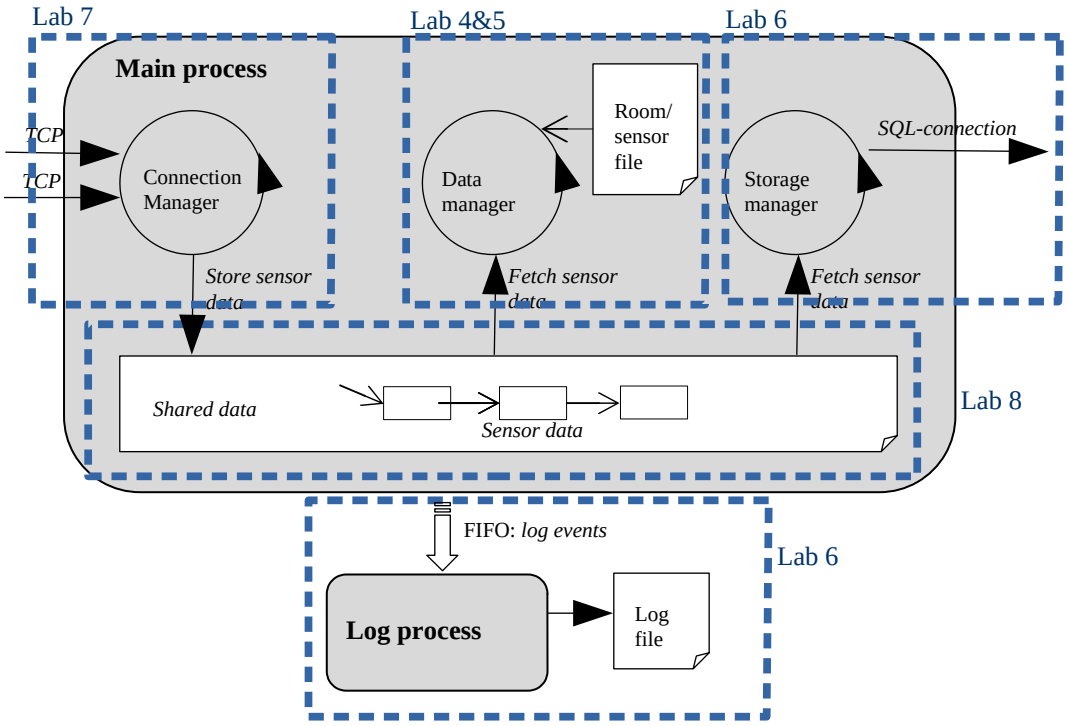
Req 9. At least the following log-events need to be supported:

1. From the connection manager:
   a. A sensor node with <sensorNodeID> has opened a new connection[1]
   b. The sensor node with <sensorNodeID> has closed the connection
2. From the data manager:
   a. The sensor node with <sensorNodeID> reports it's too cold (running avg temperature = <value>)
   b. The sensor node with <sensorNodeID> reports it's too hot (running avg temperature = <value>)
   c. Received sensor data with invalid sensor node ID <node-ID>
3. From the storage manager:
   a. Connection to SQL server established.
   b. New table <name-of-table> created.
   c. Connection to SQL server lost.
   d. Unable to connect to SQL server.

1 Remark that the sensor ID is only known after the first data packet is arrived.

How to start?

The sensor gateway is a kind of integration result of the code of lab 4 up to lab 8. The picture below shows the relationship between the different components of the sensor monitoring system and the lab sessions. It should be clear from the description of the requirements which pieces of code of these labs are required for the sensor gateway.



Deliverables and acceptance criteria

On **https://labtools.groept.be : System Software – 2021** you will find a description of what exactly and in which format (source code files, directory structure, make file, etc.) you need to prepare and upload your solution. Once you have finished the assignment, you upload your solution on **labtools.groept.be**. Please remember that this is not a compilation nor test tool. Write test code – also non-trivial test code – yourself and thoroughly test your code before uploading. For instance: test your code with multiple sensor nodes running concurrently using very small sleep times between 2 measurements. Include a debug mode.

Important remark about sbuffer-solution. Only 1 sbuffer can be used to share data between the 3 threads. Solutions where multiple buffers are used to share data between threads are not accepted to pass.

Your solution will only be accepted for grading if the following criteria are fulfilled:

- Your solution is available on **labtools.groep.be** before the deadline;
- Your solution includes the code self-review document in attachment. This is a critical reflection on your own code. Be honest with yourself! The evaluation on the exam will reveal the truth anyway.
- Your solution **compiles, runs and generates at least some meaningful output** (e.g. measurements in the database, some logs, ...);
- Your solution must be a reasonable try to implement the minimal requirements. This excludes, for instance, solutions that consist of an (almost) empty .c file, incomplete code, or code that has no or very little relationship to the exercise, code that implements a different assignment (e.g. from a previous academic year), etc.;
- Your source code is structured and readable. The following, for example, is not acceptable: you don't logically structure the code into source and header files, you apply bad naming of variables and functions, you don't use typedefs to create logical names, the code is not indented well, you write 'spaghetti-code' with goto's, ...;
- You must be able to present and defend your solution. During the defense of your solution, you are supposed to be able to answer on the technical questions of the evaluator. This includes technical questions related to programming (C, Linux system calls), source code building (preprocessor, compiling, linking, Valgrind, …) and Linux command line basics. The reference guide for the minimal knowledge on Linux are the Linux lab manuals. You are supposed to be able to work with the commands and tools introduced in these lab sessions. You should also be familiar with the usage of the programming tools introduced in the labs. More concretely, we target at least the following tools **and their options**:
    - gcc tool set (preprocessor, compiler, linker, assembly) and options (preprocessor symbols, code optimization, debug flag, ...)
    - creating static and shared libraries (ar, ldd, ldconfig, gcc), linking libraries (gcc)
    - valgrind
    - make tool and make files (you have to be able to compile and run your code using a simple make file)

Paper Assignment

Exercise 1

Draw the memory layout of your program when the program is in the following state:

- TCP connections are made on port 6543 and IP 127.0.0.1.
- The room-sensor map contains room numbers 501, 602, 703, 804 and 905 that map on sensor IDs 10, 20, 30, 40 and 50 respectively.
- Main process: all threads are created and the threads are in the following state:
  - Connection manager: sensors 20 and 50 have made a connection and each of these sensors have send 1 measurement (21°C).
  - Data manager: all received sensor data is parsed.
  - Storage manager: no sensor data is stored yet into the database. The storage manager is just starting to read the first sensor data from sbuffer.
- Log process: at least 1 log message is received and the process is waiting on the next log message.

Use the template in attachment as a starting point. This template shows the general structure for a multi-threaded process and is organized in such a way that each thread can easily access the heap. The template also includes a few examples that show how to represent the following variables and stack frames:

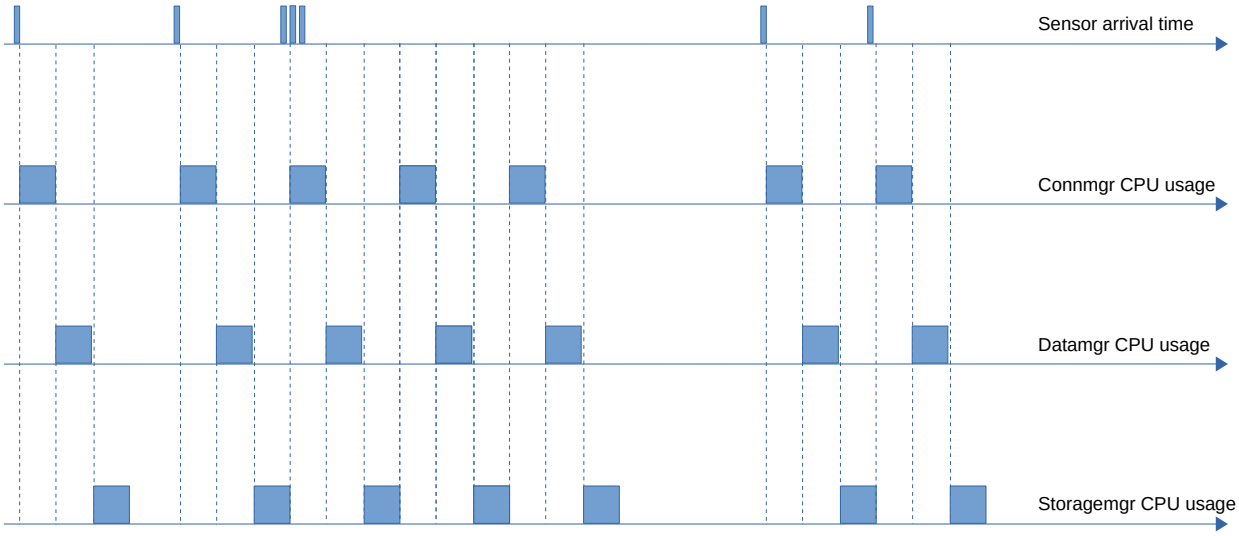
- int x = 7;
- struct { unsigned day, month, year; } d;
- file * fp = fopen(...); Notice that the real content where fp is pointing to is hidden.
- 'printf' stack frame: since 'printf' is a standard library function, you can indicate that the frame content is hidden. You can also do this for system calls and other library functions not implemented by yourself.
- <name> or <function>: replace this with the real thread or function name

Remark: for ease of reading, the function call stack grows up in this memory layout drawing.

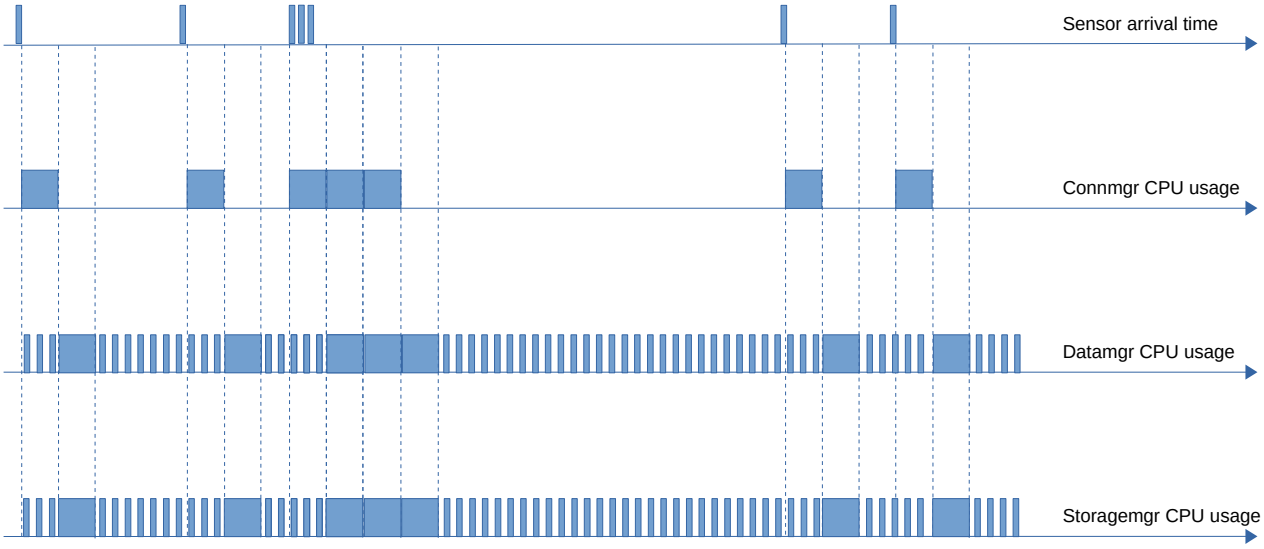
Exercise 2

The sbuffer data structure is shared between several threads and, as such, access to it must be protected. Different synchronization primitives could be chosen (mutex, semaphore, barrier, condition variable, r/w locks, ...) to do this job. What did you choose? And maybe more important: why? Write a critical reflection on your choice of synchronization primitive or combination of primitives.

- Motivate the correctness of your solution.
  - Is deadlock avoided?
  - How can you be sure that multiple writer-threads can't access sbuffer at the same time?
  - Is your synchronization solution 'fair'? For instance, if a writer or reader thread wants to access sbuffer, it should get it immediately if sbuffer is not locked by other threads. Of course, the starvation-problem should be avoided.
- Argue why your choice is the most optimal one. Does your choice guarantee efficient use of the CPU?
- Make a drawing that illustrates the general CPU usage of all threads in relation to received sensor data. Use the template below to do this (available in timing_template.odg). You draw a box to indicate the process state of the thread is 'running' (The other process states like blocked or waiting are not shown here.). You may not assume that the sensor data arrives with a fixed period, as clearly indicated in the template. A few example cases are added to illustrate the idea:



Case 1 : sequential



Case 2 : polling

Deliverables and acceptance criteria

Take the following thoughts into account when designing a solution:

- The drawings should represent the code of your solution, not some other code or some creative idea that you have in mind but you didn't implement.
- Be aware that your drawings become easily unreadable. Re-factoring your drawings at the end is a necessity.

Your solution will only be accepted for grading if the following criteria are fulfilled:

- A hard-copy (print on paper!) of your solution is available on the oral defense of your assignment. The result of exercise 1 should be printed on A3 format.
- The documents include a header containing your last name, first name and your student number.
- Your drawings are readable.
- The result of exercise 2 should not exceed two A4-pages.

Due Date of the Assignment

You must defend your solution on the date and location stated on the examination schedule. To create a more efficient planning, you must subscribe before the due date of this assignment to one of the events available on https://tolapps.kuleuven.be/Tolinto/event/6415537958_gII (same deadline as programming assignment).

Due date of the programming assignment for the 1st evaluation period: January 10, 2022 before 5 pm.

Solutions uploaded after the deadline will be rejected. The code used for the defense must be the same code as uploaded and tested on https://labtools.groept.be. No other code will be accepted.

Due date of the paper assignment: oral defense of your assignment

Calculation of the final mark

The final mark is computed as:

• 25% on the paper assignment
• 75% on the programming assignment.

Remark that a solution for the shared buffer based on polling or sequential execution of threads (cases 1 and 2 in exercise 2) will result in a lower mark (max. 15/20) on the programming assignment.