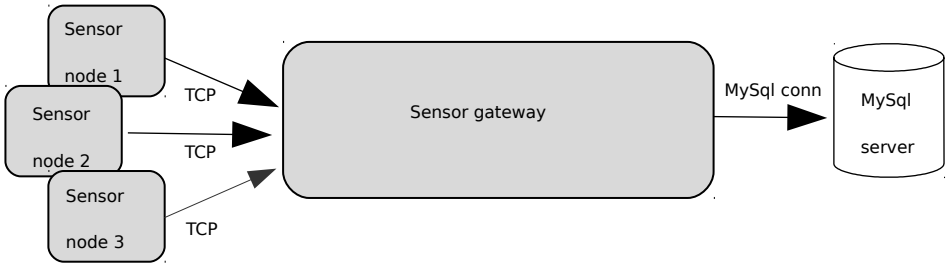
System Programming : Final Assignment

*This is an **individual** assignment and by no means meant to be "team work". It is only allowed to present **original work** implemented by yourself. It is not allowed to include substantial pieces of code or work from external sources (friends, internet, books ...). If two students present very similar solutions, no distinction will be made between the 'maker' and the 'copier'. When you have finished the assignment, you **upload your code on Toledo**. The evaluation of your work will take place before (make an appointment with your lab coach) or during the lecture exam of this course. During this evaluation, you present the source code and you are supposed to be able to answer on the technical questions of the evaluator. You must be able to compile and run your code on Linux **using a single makefile**. Basic knowledge on Linux, especially working with the command line (Bash shell) in a terminal session, is assumed. Gcc is used as default compiler and you must be able to **run gcc with command line options** to create the output of the different compilation stages (pre-processed code, assembly, object code, ...), to define preprocessor symbols, to set all warning/errors on, and to enable code optimization. **Gprof** is used as default profiler and you must be able to generate basic profiler statistics with gprof. If your implementation uses dynamic memory, **you need to show a print out of Valgrind.***

This document describes a minimal set of requirements of the sensor monitoring system. Solutions that limit their implementation to these requirements and nothing else, will be quoted at most 15/20. The remaining 5/20 can be gained by implementing at least 1 extra feature. Some examples of additional features are given, but you have the freedom to come up with your own ideas (If you hesitate about your idea, contact your lab coach for advise.).

Sensor Monitoring System

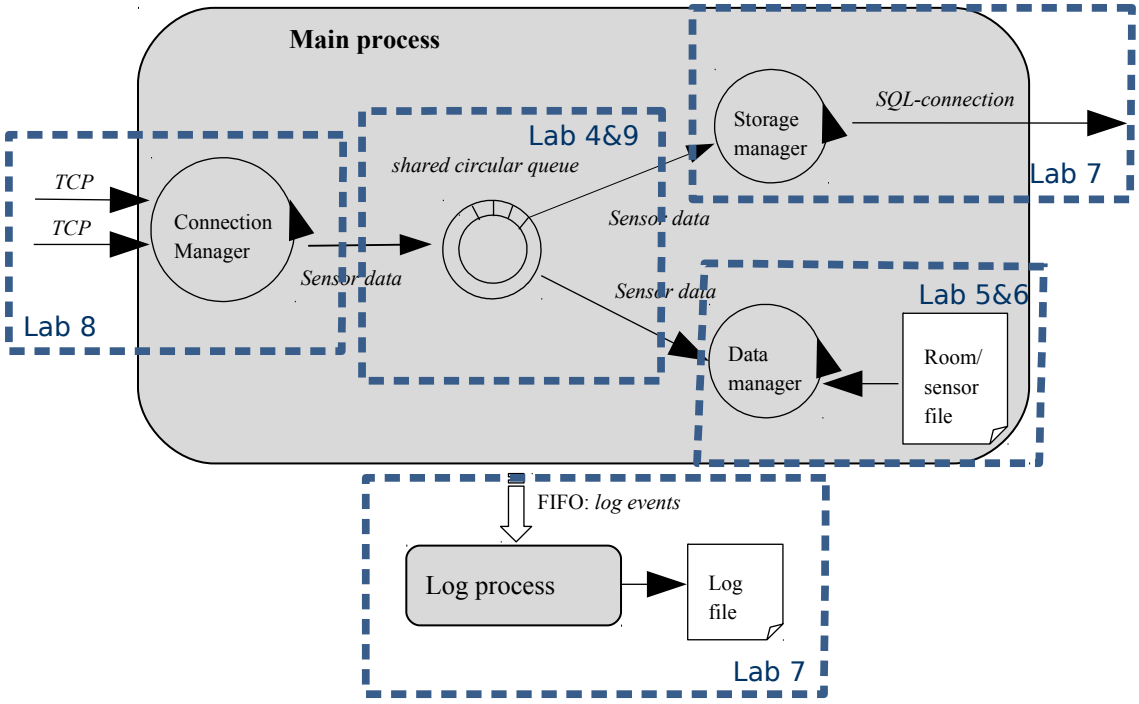
The sensor monitoring system consists of sensor nodes measuring the room temperature, a sensor gateway that acquires all sensor data from the sensor nodes, and an SQL database to store all sensor data processed by the sensor gateway. A sensor node uses a private TCP connection to transfer the sensor data to the sensor gateway. The SQL database is running on "studev.groept.be" (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 changes over time.
Working with real embedded sensor nodes is not an option for this assignment. Therefore, sensor nodes will be simulated in software. The source code for a sensor node is given in lab 8.

Sensor Gateway

The sensor gateway consists of a main process and a log process. 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 main process runs three threads: the connection, the data, and the storage manager thread. A circular queue as defined in lab 4 and lab 9 is used for shared communication between the connection (writer) and the data and storage (both readers) manager. Data can only be dequeued when both the data and storage manager have read the data. Notice that read/write/update-access to shared data needs to be *thread-safe*!

Req 2. 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.

Req 3. The connection manager captures incoming packets of sensor nodes as defined in lab 8. Next, the connection manager writes the data to the shared circular queue.

Req 4. The data manager thread implements the sensor gateway intelligence as defined in lab 6.

Req 5. The storage manager thread reads data from the shared circular queue and inserts them in the SQL database running on "studev.groept.be" as defined in lab 7. If the connection to the SQL database fails, the storage manager will wait a bit before trying again. If the connection to the SQL database is lost for a long time, the circular queue might become 'full'. In that case, the data manager is allowed to 'overwrite' the 'rear' packets in the queue. Hence, data will be lost but this solution avoids that the system will run out of memory.

Req 6. 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 7. 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 8. 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
   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
   b. The sensor node with <sensorNodeID> reports it's too hot
   c. Circular queue is full: started to overwrite data
3. From the storage manager:
   a. Connection to SQL server established.
   b. Connection to SQL server lost.
   c. Unable to connect to SQL server.

How to start?

The sensor gateway is a kind of integration result of the code of lab 4 up to lab 9. It should be clear from the description of the requirements which pieces of code of these labs are required for the sensor gateway.

Examples of additional features

1. Implement a solution that uses the SQLite (http://www.sqlite.org/) database engine as an alternative for MySQL.
2. Implement a user interface for the sensor gateway. This interface could allow the user to re-configure the gateway on-the-fly, get some statistics (How many sensors? How many packets? Running average temperature of a room? ...), access the status and log of the gateway, etc.
   a) You could use 'ncurses' to implement a UI for character-based terminals.
   b) You could use 'mongoose' (embedded web server) to implement a web-based UI that can be used from any browser.
3. Implement a more advanced / more efficient shared circular array / pointer list using fine-grained locking techniques and reader/writer locks.
4. *Add your own idea here ...*

Deliverables

Below you find a minimal set of deliverables you need to present as a result of this assignment.

Source code must be available of:

- implemented shared libraries (at least of the pointer list implementation!);
- sensor gateway;
- test program implementing an example test scenario with multiple sensor nodes
- 1 makefile.

Executable/compiled code must be available of:

- shared libraries;
- sensor gateway;
- test program implementing an example test scenario with multiple sensor nodes

The following documents must be available:

- Valgrind print outs of executables using dynamic memory.

Finally, take the following thoughts into account when designing a solution:

- Implement *efficient code* (e.g. optimize your algorithms, use efficient access techniques to shared data, etc.);
- Include a debug-mode.

Acceptance criteria

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

- The deliverables must be available on Toledo **48 hours** before the defense of the assignment.
- You must be able to present and defend your solution.
- Your solution must be a reasonable try to implement **all** 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.
- Your solution must compile without warnings, and must run without serious limitations.

Once your solution is accepted, it is still possible that you will be graded 0/20 when during the presentation and defence of your solution, one or more of the following pass/fail rules are violated.

- You have too little knowledge on the Linux command-line. The reference guide for the minimal knowledge on Linux is the lab manual of the first lab (i.e. the lab session on Linux). You are supposed to be able to work with at least all commands and tools introduced in that lab session.

- You have too little knowledge on 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
  - creating static and shared libraries (ar, ldd, ldconfig, gcc), linking libraries (gcc)
  - valgrind
  - static code checker
  - gprof
  - gdb (basics only)
  - make tool and make files

- Your solution has no or too little exception/error handling (e.g. malloc without NULL-test, empty list checks, switches without breaks, no asserts, etc).

- Your source code is unstructured code: you don’t use multiple source files and header files in a meaningful way. For example: all code belonging to a stack implementation should be organized in a .c and .h file.

- Your source code is unreadable code. Some examples of unreadable code: bad naming of variables and functions, not using typedefs to create logical names, not using comments, not indenting the code, etc.