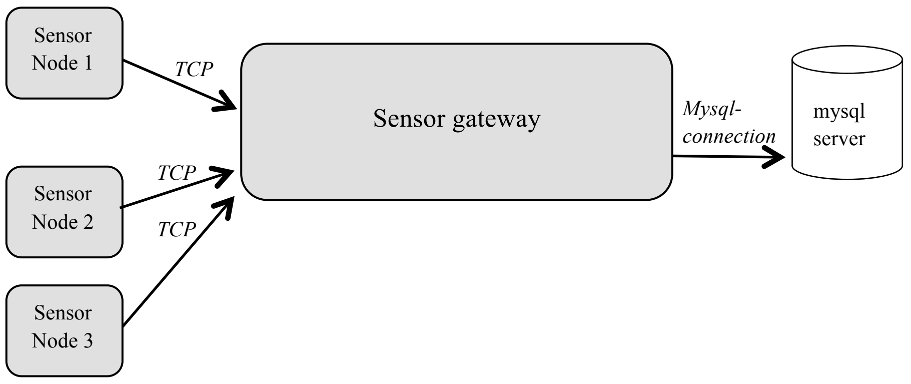
System Programming: Final Assignment

*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 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.***

Furthermore all rules mentioned in the Lab Evaluation Guidelines on Toledo also apply to this assignment. Violating these rules may lead to a non-acceptance of your code.

Sensor Monitoring System

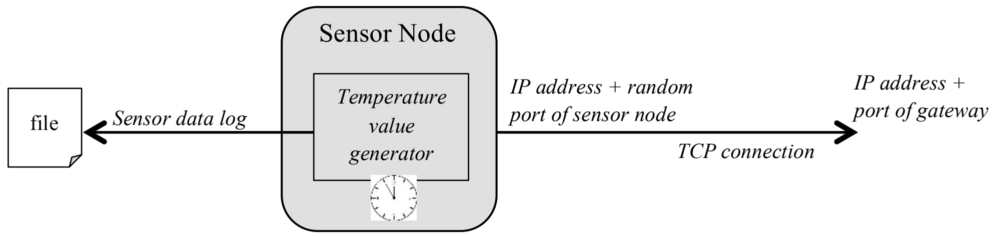
The sensor monitoring system consists of sensor nodes measuring the room temperature, a sensor gateway that acquires all sensor data from the sensor nodes, and an SQL database to store all sensor data processed by the sensor gateway. A sensor node uses a private TCP connection to transfer the sensor data to the sensor gateway. The SQL database is running on "studev.groept.be". 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.

Sensor Nodes

Working with real embedded sensor nodes is not an option for this assignment. Therefore, sensor nodes will be simulated in software, just as you did in lab 6. A more detailed design of a sensor node is depicted below. In what follows, we will discuss the minimal requirements of a sensor node in more detail.


Sensor Node
Temperature value generator
file
Sensor data log
IP address + random port of sensor node
IP address + port of gateway
TCP connection

Minimal requirements

Req 1. A sensor node is a stand-alone program that simulates an embedded system that measures the room temperature and communicates these values over a TCP connection to a sensor gateway. It should be possible to start up multiple sensor nodes on the same PC or on multiple PCs, and connect them all to the sensor gateway. You may assume that all sensor nodes can set-up a TCP connection to the sensor gateway without any network limitations such as firewall blockings, NAT problems, port forwarding issues, etc.

Req 2. A sensor node has a sensor node ID. This ID is defined at compile-time with the preprocessor directive SET_ID=<some_ID>. You may assume that the given sensor node ID is unique. Compilation should fail and an appropriate error message should be printed when this preprocessor directive is not set at compile-time.

Req 3. A sensor node generates random (but realistic) room temperature values at a pre-defined frequency, e.g. every minute. It should be possible to define the frequency at compile-time with the preprocessor directive SET_FREQUENCY=<some_value> where <some_value> is expressed in seconds. A default frequency is used if this preprocessor directive is not set at compile-time.

Req 4. A sensor node opens a (real) TCP connection to the gateway and this TCP connection remains open during the entire lifetime of the sensor node. All data is sent over this TCP connection. Only when the TCP connection is lost, a new TCP connection is set-up. A sensor node is started up with the IP-address and port number of the sensor gateway as command line arguments, e.g.:
> ./sensorNode 127.0.0.1 1234

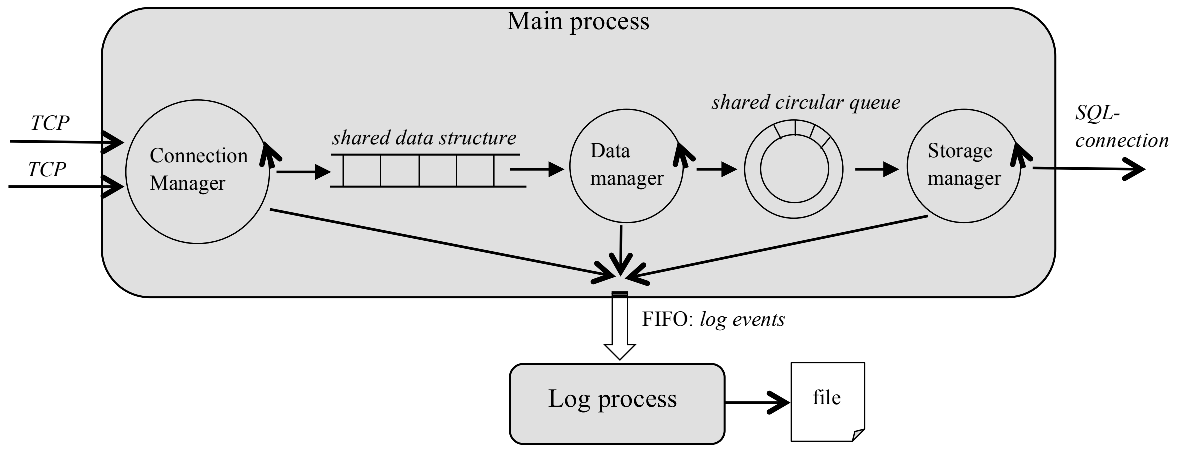
Req 5. After every temperature measurement, the sensor node sends a new data packet to the sensor gateway over the open TCP connection. The data packet uses the packet format as defined in lab 6.

Req 6. A sensor node logs a message of the format <sensor id> <sensor value> <timestamp> to a log file each time a sensor node sends a packet to the gateway (see also exercise 1 of lab 7). The name of the log file is "log<sensorNodeId>.msg", e.g. if the sensor node ID is 123, then the name of the log file should be "log123.msg". For simplicity, the log file is created in the same directory as the sensor node executable.

How to start? Start from the sensor node simulator code from lab 6. Next use the code of exercise 1 of lab 7 and the TCP client/server code from lab 8 to modify and extend the sensor node simulator code such that it satisfies all requirements.

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 7. The main process runs a pipeline model with three threads: the connection, the data, and the storage manager thread. A data structure as defined in lab 6 (i.e. a dynamic array with for every sensor node a separate pointer list) is used for communication between the connection and the data manager. Use a shared library for the pointer list implementation. A circular queue as defined in lab 4 is used for communication between the data and the storage manager. Notice that read/write/update-access to shared data needs to be *thread-safe*!

Req 8. 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. For each incoming request, it will create a new TCP socket that will be used by the connecting sensor node to send its sensor data to. Hence, the connection manager needs to monitor multiple sockets at the same time (Blocking on one socket is not an option!). Recall that a maximum amount of sensor nodes may **not** be assumed and that sensor nodes can open or close connections at any time.

Req 9. The connection manager captures incoming packets of sensor nodes (the packet format is defined in Req 5) and immediately attaches to every packet a time-stamp. The connection manager then performs a parity check and packets that fail this check will be discarded. Next, the connection manager writes the data (with timestamp) to the shared data structure as defined in lab 6.

Req 10. Whenever a sensor node goes off-line (closes his TCP connection to the sensor gateway) all its data will be deleted and the shared data structure will be updated accordingly. (*For simplicity, we assume that TCP connections are not closed because of network problems or any other connection errors.)*

Req 11. The data manager thread implements the sensor gateway intelligence. The data manager could implement data aggregation, data filtering and data compression algorithms, or apply decision logic to control, for instance, a HVAC installation. Obviously, controlling the HVAC installation of Group T is not an option (*and we are happy for that!*). For the sake of simplicity of this assignment, the data manager reads sensor data from the shared data structure and computes for every sensor node a running average over the last 10 sensor values before writing the sensor data to the circular queue. If this running average exceeds a minimum or maximum temperature value, a log-event (see further) should be generated indicating that it's too cold or too hot. It should 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.

Req 12. 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. The queue size should be possible to be defined at compile-time with the preprocessor directive SET_QUEUE_SIZE=<some_value>, otherwise the default value is used. 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 13. 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 14. 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 15. 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
   c. A data packet from sensor node with <sensorNodeID> failed the parity check
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. 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.

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:

– shared libraries (only of own development);
– sensor node;
– 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 node;
- 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.