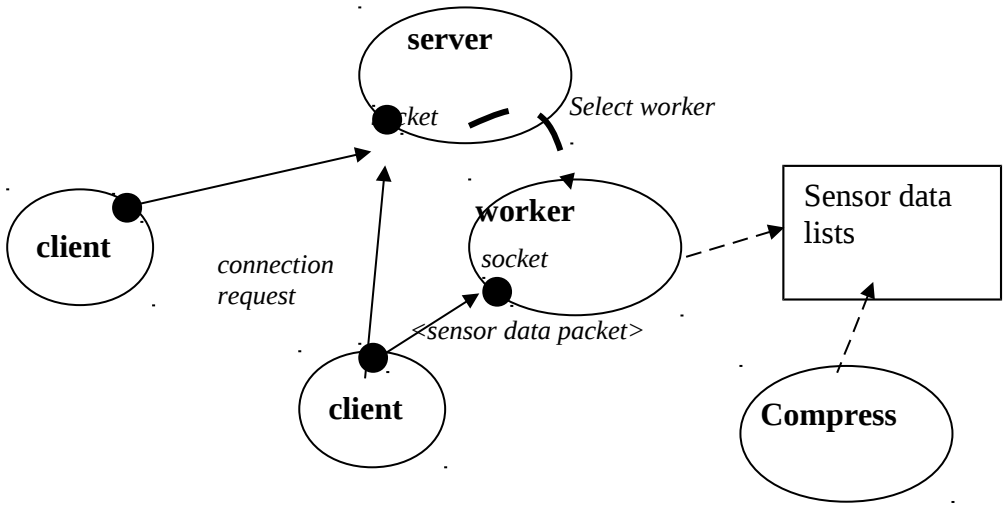
Programming with Threads

Implement a multi-threaded sensor data server. The server allows uploading sensor measurements by any sensor client using a TCP connection. Use the TCP/IP socket code from the previous labs. Notice that this server is different from the file server of the previous lab: the server is used to collect single sensor measurement and is not used to upload files containing sensor measurements. Actually, a sensor simulator was introduced in lab 6 that communicates data packets to the sensor simulator using a Linux pipe. In this exercise, the Linux pipe will be substituted by a real TCP connection.

The data server should be able to handle multiple connection requests at the same time. Again, the data server solves this problem by implementing the manager/worker multi-threading design pattern. The port on which the server is listening to incoming connections is given as a command-line argument at startup of the server. At startup, the server creates a static pool of worker threads. For each connection request, the server selects a free worker thread to handle the connection. The worker threads can update their status (free/busy) in a status array. Be aware that this array is shared between the main thread and the worker threads!

A sensor client will send messages with a packet structure as defined in lab 6. The client can send multiple packets during one connection. Also keep in mind that the main thread should 'wait' on the 'exit' of the worker threads.



The task of every worker thread is mostly the same as described in lab 6. For your convenience, the task description is repeated here. The worker thread captures incoming packets and immediately attaches to every packet a time-stamp. It also performs an even parity check and packets that fail this check will be discarded. For every sensor a separate pointer list is maintained that contains the latest measurements. You can use an array in shared memory ranging over all sensor IDs to maintain all these pointer lists. For each incoming packet, the sensor ID is read and the sensor data with time-stamp is stored in the corresponding pointer list. From time to time (e.g. this could be daily or weekly, depending on the measurement frequency of the sensors), a 'data compression' thread will wake up and compress the sensor pointer lists by substituting every 3 successive sensor measurements with only 1 measurement containing the average of all 3. Since this is repeated over and over again in time, it prevents that the memory used per sensor is growing out of proportion.