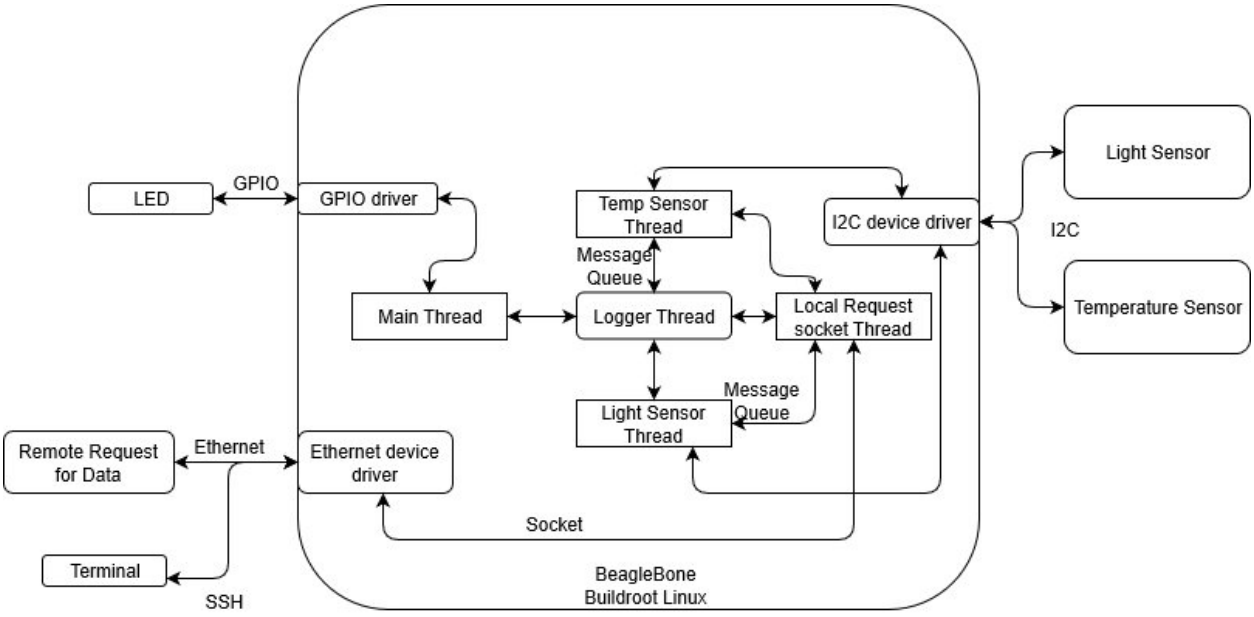
ECEN 5623 – Project 1 Architecture

Environment Monitoring Multithreaded Application

Vatsal Sheth and Sarthak Jain

3/31/2019

Software Design Diagram



Task name, software structure and responsibilities

1. Temperature and Light sensors: Two child threads spawned from main() will interact with the temperature and light sensor each. Startup tests will be run to check whether:
    a. Handshaking between main thread and child threads is functional via use of conditional variables. Heartbeat is implemented using pthread conditional variables. Monitor structure is implemented which contains conditional variable, mutex, timespec structure, and failure count. Each thread has a object of this structure. All threads keeps on signaling their respective conditional variable in a loop. To check the heartbeat timed conditional wait is used. If timeout occurs then it is asserted as a fail and its failure count is incremented. On three successive failure, thread is cancelled and created again. Meanwhile this failure is also indicated using an LED.
    b. The sensors' hardware connections are functioning. This can be checked by the response on I2C bus using Built in self tests. The tests check if data is being written to registers correctly by reading and exiting the program if not the same. If any error message is received, an error can be logged.
    c. Sensor initialization is complete.
    d. Log messages on successful boot. If unsuccessful, LED can be turned on.

The tasks will collect data from the sensors periodically based on a timer handler of 1 second, and the data will be logged. The API requesting the sensor data will return the data it has based on the last timer call. The API will also include an option to force returning the current value by taking a reading at that instant. For logging to text file, the task will write to message queue of the logger thread. This will include the ID of the called thread, log message, timestamp and log verbosity.

2. Synchronized Logger Task: We plan on implementing a separate thread for the logger task, so as not to have the clumsiness of a blocking mutex during file print. To increase the efficiency of logger thread, mq_notify is used. On receiving a notification handle set a flag, which thread polls and then it keeps on receiving from queue in a loop using timeout. If mq_receive timesout, it means the queue is empty and the flag is cleared and the queues is again set to notify. This thread will create a named semaphore, to allow synchronization between the log sources. This semaphore will be used to protect write/read of logger thread's message queue. Whenever this queue is non-empty, the logger thread will read from it, and depending on the verbosity, print to log. By default, the logging verbosity will be normal, and can be set to debug based on a run-time switch.
3. Remote Request Socket Task: We could spawn another thread and keep it open on a socket port. The thread waits for the user to input the type of data required and will accordingly open a socket connection to the main thread, passing on the data request. Main will filter the data type, retrieve it from the child threads, and will reply with the data type, which will then be printed on terminal.
4. Main thread: The main thread, will firstly, spawn off all child threads. Two child threads, a logger thread, and a socket thread will be created. Three different IPC channels will need to be established with main, one type with each sensor child thread (possibly shared memory access), one with the logger task, and one with the socket thread.

Functions are provided in code with sufficiently clear comments.