Project, Part 1 – Due Monday 10/7 11:59 PM

In this first part of the project you are going to implement a command interface for the **Telosb** nodes and the underlying service responsible for carrying out the commands. With this application, users can query each sensor directly or command the node to take regular samples, store them locally on flash, and retrieve them again later in a batch. Users can issue commands either manually through **picocom** or through some high-level application (which we will return to in Part 2 of the project).

Purpose

- To learn about TinyOS and the programming language nesC
- To understand UART communication
- To understand FLASH storage

Prerequisites

- C programming skills
- Debugging skills using text input/output and leds
- Read **TinyOS Permanent Data Storage TEP**, http://www.tinyos.net/tinyos-2.x/doc/html/tep103.html

Deliverables

- Code:

  A tar/zip containing the application folder, compilable when extracted to the cs450 directory.

- Report (at most 3 pages using 10pt type):

  A PDF containing: Design Choices, Bug Report, Evaluation, Future Work, Summary.

  Design Choices: You often have multiple choices when writing code. Let us know the reasoning behind the choices you made. Discuss the pros and cons.
  Bug Report: If something is not working, that you know of, let us know.
  Evaluation: Show us your application is working as intended, e.g., screen shots, logs, etc.
  Future Work: How would you improve the application? You are encouraged to include changes to the constraints set by the assignment.

  Remember, the only way we can know about any particularly clever solutions you've made, and give you credit for them, is if you tell us about them in the report. Also, if your code is not working properly, please document what you have done and achieved.

Sample Application

In the GIT cs450 folder you will find a new sample application, **tosProbe**. Compile, run it on a **Telosb** node, and connect to it with **picocom**. Keys **1-4** read the sensors and print the raw values together with a count (in binary milliseconds) of how long since the last boot-up. Key **e** erases the flash (useful for setting the flash in a known state), Key **a** writes a record of numbers to the flash, and Key **r** reads back the next record starting at the current read cookie.

Assignment

Use the **tosProbe** application as a starting point and implement the following three parts:

1. Change the **UartStream.receivedByte** and **uartTask** from taking a single character at a time into buffering a stream of characters and posting the **uartTask** to process the buffer. This stream of characters will be commands and parameters from either a human or another application, which means you will have to consider how to
    a. Determine when a new command begins and ends
    b. Handle corrupt/missing characters and interrupted commands

    You are allowed to make reasonable assumptions about the inter-character-interval.

2. Implement a command interpreter for the following commands:
    a. command: single read of *<sensor>.*
       output: timestamp and raw sensor value.

    b. command: continuous read of *<sensor>* every *<interval>.*
       output: continuous timestamp and raw sensor value output.

    c. command: stop all activity.
       output: acknowledgement of command.

    d. command: log all sensors to flash every *<interval>.*
       effect: read all sensors and store them to flash as one record every interval.
       output: acknowledgement of command.

    e. command: report current write-cookie.
       output: 32 bit write-cookie.

    f. command: read *<number of records>* from *<address cookie>.*
       effect: seek to the given address and read the number of records specified.
       output: timestamp and raw sensor values for each record.

    You are allowed to define how commands, parameters, and outputs are represented and coded in the UART stream. The *<sensor>* is one of the four sensors on the Telosb. The *<interval>* should

support the range from 1 second to 1 day. The *<number of records>* should at least support the range 1 to 3.

3. Implement the tasks and functions behind the commands above.

Grading

There are many different solutions to the assignment above and it is up to you to choose which one to implement and discuss your choice in the report. We will grade the assignment based on (1) how well it works, (2) how robust the implementation and communication protocol are, (3) the written report. We will ***not*** be grading the assignment on how efficient your implementation is (that will be in Part 2 of the project).

In order to improve robustness on the communication protocol it might be useful to draw inspiration from the methods used in wired networking that you learned in your networking class.