Laboratory assignment six
Introduction to Motes
ECE 397-1

Lab checked on or before 1 March
Lab report due during lab check
Prepared by Robert Dick

Please keep track of how long you spend doing this laboratory assignment. Specifically, how much time is needed to do the problems after studying enough to understand the concepts?

1 Goals

1. Develop priority-based cooperative scheduler for TinyOS that keeps track of the percentage of idle time.

   One possibility is to have each task call a scheduling routine at the end of execution. This scheduling routine may evaluate pending tasks and schedule the one with the highest priority. Please use the following priorities.

Task	Priority
Buzz	4
Send idle percentage	3
Send audio samples	2
Periodic temperature, light, and microphone level transmission	1

   Pending tasks are created by events such as data sensing timer ticks and incoming messages. Note that, in the next motes lab, it will be necessary to prioritize additional types of tasks.

   There are two straight-forward approaches to designing your scheduler. In the first, each task, upon completion, explicitly calls the scheduler, which evaluates the pending tasks and posts the task with the highest priority. In this case, the scheduler should be triggered by data sensing timer ticks and incoming messages. When the scheduler has no task to run, it should start a timer that keeps track of idle time. This timer's behavior should be modified when tasks are active in order to determine the percentage of idle time.

   Another alternative approach is using a background timer to trigger the scheduler. If a task is already running, do nothing. Otherwise, post the highest-priority pending task. With this approach, the scheduler's timer may be used to determine the percentage of idle time.

   Either approach might be extended to allow cooperative early task termination by adding checkpoints at appropriate points within long-running tasks. Of course, this is less elegant than preemptive multithreading.

   Extra credit: Use BDThreads (http://www.bdmicro.com/code/threads/) as a starting point to implement a preemptive, prioritized, multithreading executive within TinyOS.

2. Develop a tree routing algorithm for the sensor network.

   Use a periodic TinyOS beacon message from the root node (by convention, the node with an identification number of 0) that propagates through the network, allowing all nodes to determine the number of hops from the root node. When a message has a source that is more hops away from the root node than the receiving node, and this is the first time the message has been seen by the receiving node, rebroadcast. Otherwise do not rebroadcast. It may be necessary to stagger rebroadcast to avoid message collision. Please acknowledge message receipt for the sake of reliability. There is no need to pipeline message transmission in this lab. You may either change the size of TinyOS messages or join them at the root node for before transmission to PPCs.

   Commands coming from outside the network may be propagated to all network nodes but only the target node should respond.

3. Send temperature, light, and microphone data to a PPC, via the network root.

   Twice a second, send temperature, light, and microphone data to the root node, which then transmits this information to the Bluetooth-attached PPC, if any. The most straight-forward way of doing this is encapsulating a PPC message within a TinyOS message and removing the wrapper at the root node in order to send the PPC message to the UART. Unfortunately, the serialization and deserialization code for the PPC messages is not at all straight-forward to port to the motes. One of the main problems is that nesC and AVR-GCC were not built for use with C++ code. We spent a lot of time attempting to find a work-around (I'll spare you the gory details) but ultimately decided to rewrite the serialization code in ANSI C. Fortunately, this technique is straight-forward. However, it requires a lot of boiler-plate code we are still in the process of testing. We'll release serialization code on Wednesday and update it regularly. If you begin the assignment early on Wednesday, please start with the scheduling and routing portions of the assignment.

4. Have motes respond to *send audio samples* and *buzz* commands.

   Propagate PPC-originated command messages to the motes. Respond by taking and transmitting the appropriate number of audio samples at high frequency or by buzzing for the specified amount of time. If you haven't yet figured out how to sample good-quality 3 kHz audio data, please see me (R.D.).

5. Play back or display this data on PPCs to verify the that the system functions.

   In this lab assignment you needn't spend much time PPC code. However, it is necessary to form, serialize, deserialize, and display messages. Ideally, you should play audio data via the PPCs speaker.

You may find it useful to do this lab using the machines in Tech CG30. This will make it easier for me to help you during development, as my office is in the same building. If you want to use the machines in CG30 but don't yet have a Wilkinson lab account, please stop by Tech. M334 to have the account created. It should only take a moment.