

# CS 240E Assignment 1: Scheduler

Due: 11:59 PM, January 30, 2007

## Basics

In this assignment, you'll get up to speed using TinyOS and Telos motes by writing a priority scheduler. Rather than the standard TinyOS FIFO scheduling policy, your scheduler will have two task priorities: system and application. Within each priority level, the system schedules tasks using a FIFO policy, but it will always schedule a system task before an application task. You'll write an application that demonstrates this functionality.

## Installing TinyOS

To complete this assignment, you'll need to install TinyOS on a machine that has USB ports, so you can program your Telos nodes. You'll be using TinyOS 2.0. To download TinyOS and all of its tools (the nesC compiler, msp430-gcc, etc.), go to <http://www.tinyos.net/tinyos-2.x/doc/html/install-tinyos.html> and follow the instructions there.

If you're running Windows, you'll then also need to download support for the Telos motes. Go to [http://www.moteiv.com/community/Tmote\\_Windows\\_install](http://www.moteiv.com/community/Tmote_Windows_install) and follow the instructions there for downloading the FTDI serial drivers. Linux generally (2.4, 2.6 kernels) generally come with the kernel modules by default, but if not, you'll need to track down `ftdi_sio`.

## Getting Started

TinyOS 2.0 has four forms of documentation:

1. **Tutorials** are simple introductions to TinyOS 2.0 and how to program it. The tutorials can be found in the <http://www.tinyos.net/tinyos-2.x/doc/html/tutorial> directory.
2. The **Programming Manual** is a more in-depth discussion of how to program in nesC. It talks about the basic programming constructs and how to use them. You can download the manual: <http://csl.stanford.edu/pal/pubs/tinyos-programming.pdf>.
3. **TEPs** provide detailed descriptions of the various parts of a TinyOS system, including the boot sequence, communication, power management, timers, and sensing. While the programming manual tells you how to use nesC to write effective code, TEPs are more akin to a description of the TinyOS APIs. HTML of the TEPs can be found in the `tinyos-2.x/doc/html/` directory. TEP 106 covers the TinyOS scheduler.
4. **nesdoc** is the nesC equivalent of javadoc: it is direct documentation of the TinyOS source code. The best way to sift through nesdoc is to start at <http://www.tinyos.net/tinyos-2.x/doc/html/nesdoc/telosb>.

You might find the TOSSIM simulator useful. It currently only supports the micaZ platform, but as long as you are not using MSP430-specific interfaces (which have Msp430 in their name), this should not be an issue. The TOSSIM tutorial can be found in `tinyos-2.x/doc/html/tutorial/`.

The `tinyos-help` mailing list archives are an excellent resource for answering questions. If you can't find your answer there, don't be afraid to ask. This is the URL for the archives: <http://www.tinyos.net/scoop/special/support#mailing-lists>.

## Assignment

A very good resource for this assignment is TEP 106: Schedulers and Tasks. You can find it in the online TinyOS documentation. TEP 106 explains how the standard TinyOS scheduler works and gives an example of how one might extend it to support soft real-time through an earliest deadline first (EDF) scheduler.

The assignment is to write a new scheduler for TinyOS. The scheduler has two task priorities: system and application. System tasks are run in a FIFO order and application tasks are run in a FIFO order, but system tasks take precedence over application tasks. The nesC `task` and `post` keywords should be application tasks. System tasks require a separate, manual component wiring. System tasks should have the TaskBasic interface. You can find information on the task and post keywords in the TinyOS Programming Manual and tutorials.

Note that TEP 106 requires that a scheduler must not starve default tasks; that is, assuming that no task has an infinite loop, a posted task must run eventually regardless of what other tasks are posted. More specifically, even if an application has a self-posting system priority task, such as this:

```
uint8_t i;
event void Task.runTask() {
    i++;
    call Task.postTask();
}
```

then an application task should run eventually. If your scheduler had pseudocode like this:

```
schedule() {
    if there's a system task, run it
    else if there's an application task, run it
    else go to sleep
}
```

then the above system task would starve all application tasks.

However, system tasks should have higher priority; given a stream of pending system tasks and a stream of pending application tasks, system tasks must receive more than 80% but less than 95% of the scheduling events. Dividing the CPU share in terms of execution time rather than task execution counts is extra credit: TEP 102 specifies the timing interfaces in TinyOS, so might be a good resource for this.

You must also write an application that demonstrates that your prioritized task scheduler works. Your application should, by default, run a self-posting application task that continuously blinks the green LED at approximately 10Hz. You can achieve this by writing a task that runs for 5 milliseconds and toggles the LED every 20 executions. When the node receives input, then it should post a system task that blinks the red LED at approximately 10Hz for a total of 50 blinks. When the system task runs, you should see the user task blink rate slow down.

The input that causes the system task to run should be from a sensor. You may use any of the onboard sensors or the user button. TEP 109 describes several of the Telos sensors in its documentation. You should probably use one of the light sensors: covering the sensor triggers the system task to run.

Note that there currently is no driver written for the button. The user button is on MSP430 interrupt port 27. TEP 117 describes the TinyOS interfaces for controlling interrupts. You want to instantiate an instance of the component `Msp430InterruptC` and wire it to `HplMsp430InterruptC.Port27`. Depending on how you implement your system task interface, the “run once” semantics of TinyOS schedulers mean that you may not need to debounce the button.

## Handing In

All of your code should be in a single application directory. It should be reasonably documented (e.g., explain, at a high level, what each function does if it's not very simple). The directory must have a README that describes your scheduler, how it works, and how to trigger the system task. Send a tarball of this directory to Jung Woo.