

# CS 240E Assignment 3: Threads Design

Due: 11:59 PM, February 27, 2007

## 1 Basics

This assignment calls for a group of two to work together. You may work in the same group as you did for the previous assignment, or you may form a new group. As the assignment after this one is to implement the design you come up with in this assignment, you just be in the same group for this and the next assignment.

The hand-in for this assignment is a 3 page document describing a threaded programming API and a design of how you will implement a threading system for TinyOS.

## 2 The Problem

Some programmers like threads. For many tasks, they make programming simpler and easier. However, most sensornet operating systems are event-driven. This has led research groups to propose several approaches to incorporating threads into an event-driven operating system, and there are certainly more to come.

There are two parts to building a threading system on top of an event-driven system. The first is defining an API. Having threading greatly changes issues such as memory management, and this can in turn affect API decisions. A sensor application does four basic things: sense, store, communicate, and compute. The last is typically just an instruction stream and does not require special system calls.

In addition to system calls, there is also the question of what concurrency calls there are. TinyThread, for example, used cooperative threading; correspondingly, it needs a `yield()` call. TinyMOS, in contrast, uses preemptive threading with fixed priorities. It doesn't need a `yield()`, but it does need a way to assign thread priorities. When considering your API, you may use this example application to guide your decisions.

A sensor node samples its light, temperature, and humidity sensors every five seconds. Every minute, it puts these buffered readings into packets that it sends a burst to the next hop in a routing tree. It also logs these readings to non-volatile storage. When a node receives packets to forward up the routing tree, it examines the packets to maintain a histogram of the forwarded values. Every five minutes, it decays the histogram, dividing the counts by two. The histograms should have sixteen bins; with three sensors this will require  $48N$  state, where  $N$  is the size of the counter you use. When a node transmits packets, it should update the histogram values with its own readings. With respect to routing packets up a collection tree, please refer to TEP 119. The TEP is a bit out of date, but the two abstractions you are about are `CollectionSenderC.send` and `Intercept.intercept`. With respect to logging data to storage, please refer to TEP 103.

There are several ways to implement this application with threads. For example, you could:

- Have a first thread with an infinite loop that senses, sends, logs, and decays the histogram, sleeping to maintain time. Have a second thread that intercepts forwarded packets and examines their data. Use a mutex to protect the histogram (or have cooperative threading).
- Have one thread that senses and puts those readings onto a queue. Have a second thread that pulls readings off the queue to send and log them. You could either have the second thread wake periodically, or have it wait on the queue until it is the desired length (i.e., have a condition variable that the producer signals when the queue is long enough). Have a third thread that intercepts forwarded packets.
- Have every system call include a timeout and use a single thread. The thread waits for packets to intercept with a timeout. It checks the local time to decide if it is time to sample, send, or decay the histogram.

The second part is deciding how to actually implement the threads. Does the event driven operating system run as a thread (TinyMOS), or does it schedule threads (TinyThread)? Does your system allow dynamic thread allocation (TinyMOS), or are threads statically allocated? Deciding how thread scheduling interacts with the scheduling of the existing OS will greatly influence what kinds of threads you can support and how they can be implemented.

### 3 Writeup

The handin for this assignment is a 3-page paper, prepared in standard ACM conference format. Your paper should have three sections:

1. An overview of the design of your threading system and how it will interact with the TinyOS scheduler.
2. Your threaded programming API: how does a programmer create, manage, and write code for threads? An example `send`, `log`, and `send program` might be helpful here.
3. Your timeline for implementing and evaluating your approach as well as how you plan to evaluate it.

Think of this assignment as a proposal. Your goal is to clearly articulate the work you are going to do and how you are going to do it. The proposal has two purposes. The first is to give you an opportunity to think through the options and come up with a coherent plan and design. The second is to give me and Jung an opportunity to give you early feedback before you're halfway through the project.

You do not need to incorporate smart stack allocation tools. You can assume the presence of such tools and allocate stacks that are safely large enough (e.g., 512B).

With respect to implementing context switches, the man pages on `setjmp(3)` and `longjmp(3)` can be helpful. Your C compiler for the MSP430 has these in its standard library. And be mindful of the UNIX man page:

“If `longjmp()` or `siglongjmp()` are called even though `env` was never primed by a call to `setjmp()` or `sigsetjmp()`, or when the last such call was in a function that has since returned, absolute chaos is guaranteed.”

### 4 Handing In

The handin for this assignment is the three-page description of your design. The report must follow ACM formatting guidelines and must be a PDF.

### 5 Grading Criteria

Your handin will be graded on the following criteria:

- **Design (50%):** You should think through your design carefully, Try writing pseudocode to see how you might perform some basic operations. TinyMOS sheds some light on how simple decisions early on can make design more complex later in the process.
- **API (30%):** Your API should include functions for 1) sending single hop packets (TEP 116), sending packets up a collection tree (TEP 119), sampling the Telos sensors (TEP 109), logging data (TEP 103), sleeping, and getting the local time (TEP 102). You should include code for a simple application that samples a sensor, logs the value to flash, sends the packet up a collection tree, and sleeps for a few seconds before repeating.
- **Timeline and evaluation plan (10%):** The easiest way to run into trouble in this assignment is to not have a good timeline and plan for finishing it. You don't need to go into tremendous detail, but by starting thinking about evaluation early, it will give you a better idea of what things you want to focus on making efficient.

- **Writing (10%):** Writing things up at the last minute means it's unlikely that what you write will be coherent or well thought out. If your proposal is hard to understand, it will be hard for us to give you feedback.

As always, we encourage you to send email to the class list (be sure to use the -all list so it reaches staff too) to ask questions or discuss the assignment.