LiteOS, Contiki, MANTIS, & nesC

Presented by Nancy White, Chloe Norris, Catherine Greene, and Bretny Khamphavong

LiteOS

By Nancy White

LiteOS

* Similar to a unix operating system
* Fits on memory constrained motes such as MicaZ and IRIS
* Used for Wireless Sensor Networks
* LiteOS is not event driven

Features of LiteOS version 1.0

* Supports Windows XP, Vista and Linux
* Has both MicaZ and IRIS nodes
* Plug-and-play routing stack
* Lightweight event logging
* Multi-threading kernel
* Writes applications in C
* Hierarchical file system

Contiki OS

Chloe Norris

Radford CREU

The Beginning

- Released in 2001
- Specifically targeted and designed for WSNs
- Swedish Institute of Computer Science
  - Led by Adam Dunkles

Image Source: www.sics.se/~adam
Information Source: http://www.sics.se/contiki/

About the Operating System

* Open source
* Highly portable system
* 2 kilobytes of RAM
* 40 kilobytes of ROM
* Written in C
* Event driven kernel

Information Source: http://www.sics.se/contiki/

Contiki in Everyday Life

* Used in ships, satellites, oil drilling equipment, and more
* Designed for microcontrollers with limited memory

Image Source: http://www.sics.se/contiki/about-contiki.html
Information Source: http://www.sics.se/contiki/

Memory

* Lightweight protothreads
* Supports per-process optional multi-threading
* 3 different types of memory management:
  1. Malloc()
  2. Memory block allocation
  3. Managed memory allocator

Information Source: http://www.sics.se/contiki/

Communication

* IP communication for IPv4 and IPv6
* Rime low-power radio networking stack
* Interacting with the network of sensors via web browser

Image Source: http://www.sics.se/contiki/about-contiki.html
Information Source: http://www.sics.se/contiki/

Power Efficiency

- Power efficiency within most wireless sensors is always a paramount issue
- Software-based power profiling mechanism
  - keeps track of energy usage within each node

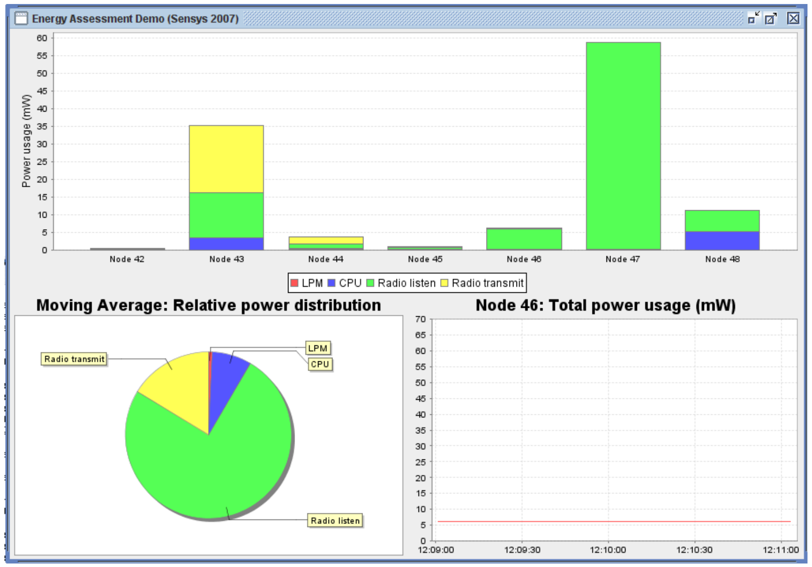

Image Source: http://www.sics.se/contiki/about-contiki.html
Information Source: http://www.sics.se/contiki/

MANTIS

By Catherine Greene

MANTIS Basics

- Name: **M**ultimod**A**l system for **N**e**T**works of **I**n-situ wireless **S**ensors
- Open source embedded multithreaded operating system for wireless micro sensor platforms
- Written in C
- Implemented in a lightweight RAM footprint
  - Fits less than 500 bytes of memory
    - Including: kernel, scheduler, & network stack

Source: http://portal.acm.org/citation.cfm?id=1160178

MANTIS Developments

* Developed by the MANTIS Group at CU Boulder
* 07/27/2005 MOS 0.5.5 released
* Aug/Sept 2005 MOS deployed in the Bitterroot National Forest (Idaho) in active wildfires to monitor weather conditions
* 5/30/2006 MOS TinyMO adds priorities and multithreading framework for evolving TinyOS
  * Ran TinyOS as a thread in MOS
* 10/19/2007 MOS 1.0 beta released
  * Improved Stability, bug fixes

Source: http://mantisos.org/index/tiki-index.php.html

MANTIS Qualities

- Has a power-efficient scheduler that sleeps the microcontroller
  - Sleep() function
- Flexible with cross-platform support
  - Ability to test on PCs, PDAs, and other sensor platforms
- Supports remote management of in-situ sensors through dynamic reprogramming & remote shells

Source: http://www.cs.colorado.edu/~rhan/Papers/MANTIS-MONET.external.pdf

MANTIS Downfalls

- Needs to work on improving:
  - Low power management
    - Though Sleep() helps conserve power, more power should be conserved
  - Demonstrating reliability or code updated over the network, optimizing the size of updates
  - Ensuring security & authenticity of updates
  - Etc..

Source: http://www.cs.colorado.edu/~rhan/Papers/MANTIS-MONET.external.pdf

nesC

By Bretny Khamphavong

nesC: The Basics

- Extension of the C programming language
- Components can be referred to as “motes”
- Designed and written researchers from UC Berkeley, Intel and Harvard University
- Authors include Eric Brewer, David Culler, David Gay, Phil Levis, Rob von Behren and Matt Welsh

Challenges of the OS

1. Motes are highly reactive in terms of their normal operation
   * React to their environment or to radio signals sent by other motes or by a controlling base station
   * Driven by events
   * Must process data as well as handle event arrival concurrently

Challenges Con’t

2. Motes have very limited hardware resources
   - Not expected to change
   - Emphasis is on the physical size of the mote rather than on memory or processing power
3. Software must enable highly available applications so as to reduce mote failure due to software
   - Language must be designed to minimize the number of run-time errors and detect potential issues at compile time

Components

* Basic building block of nesC applications; components both provide and use interfaces
* If a component provides an interface, then it must implement the commands for that interface.
* If a component uses an interface, then it must have code to properly handle events defined within that interface

Concurrency

- Two types of code: Asynchronous Code (AC) and Synchronous Code (SC)
- AC is code that is reachable from at least one interrupt handler and thus the timing of its execution cannot be perfectly predicted
  - Example: telephone
- SC is code that is only reachable from tasks, and therefore can never interfere with other synchronous code

Race Conditions and Atomic Reasoning

* Any shared state that is updated from an AC is a potential race condition; and by extension if that shared state is also updated by SC, it remains a race condition
* nesC compiler can identify data that is updated by AC and throw a compile-time error if that data is not inside an atomic section

In Conclusion…

* Concepts and principles could be applied to any imperative language
* Power and flexibility provided by interfaces make developing complex applications on these embedded systems reliable and high performing