Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors

Adam Dunkels, Björn Grönvall, Thiemo Voigt

Swedish Institute of Computer Science

IEEE EmNetS-I, 16 November 2004
OUTLINE

- Introduction
- System Overview
- Kernel architecture
- Services
- Communication support
- Discussion
- Conclusion
Introduction

- **Contiki** – an OS for sensor network nodes
- Ported Contiki to a number of platforms
  - MSP430, AVR, HC12, Z80, 6502, x86, ...
- Built a few applications for experimental network deployments
Introduction (cont.)

- Contribution
  - Downloading code at run time
    - Selective reprogramming
  - Portability
  - Event-driven systems
    - Event vs multi thread
System Overview

- Hardware target
  - “Mote”-class device
    - 10-100 kilobytes of code ROM
    - 1-10 kilobytes of RAM
    - Communication (radio)
  - ESB from FU Berlin
    - MSP430, 2k RAM, 60k ROM
System Overview (cont.)

- Contiki system consists of:
  - kernel, libraries, program loader, a set of processes

- Services can be dynamically replaced at run time

- Interprocess communication is done by posting event

- All processes share the same address space
Kernel architecture

- Event-driven vs multi-threaded
  - Event-driven (TinyOS)
    - low context switching overhead, fits well for reactive systems
    - Not suitable for e.g. long running computations
      - Public/private key cryptography
  - Multi-threaded
    - Suitable for long running computations
    - Requires more resources (stack)
- Trade-offs: preemption, size
Kernel architecture (cont.)

- Event-driven (TinyOS)
  - Processes do not run without events
  - Event occurs: kernel invokes event handler
  - Event handler runs to completion (explicit return; )
Multi-threaded
- Preemption
- Thread runs until next blocking statement
- Each thread requires its own stack
  - Larger memory usage
- Locking problems (race condition)
Kernel architecture (cont.)

- Combine **Event-driven** and **Multi-threaded**
Kernel architecture (cont.)

- **Contiki: kernel is event-based**
  - Most programs run directly on top of the kernel

- **Multi-threading** implemented as a library
  - Threads only used if *explicitly* needed
    - Long running computations, ...

- **Preemption** possible
  - Responsive system with running computations
Kernel architecture (cont.)

- Two level scheduling hierarchy
  - Event scheduler that dispatches event to running process
  - Periodically call processor’s polling handler

Control flow:
- Main()
  - initialize
  - ek_run()

Event queue:
- process
- App

Loop
- ek_process_event()
- Poll
  - driver1
  - driver2
  - driver3

Kernel: ek.c
Kernel architecture (cont.)

- Loadable programs
  - Run-time relocation function and a binary format that contain relocation information
  - Loader check sufficient memory space
  - Loader call initialization function

- Power save mode
Services

1. Services is called
2. Stub process service look up in services layer
3. Return pointer
4. Stub calls the implementation of the request function

An application function calling a service
Service replacement

- Kernel provides a special mechanism for replacing a process and retaining the process ID
Communication support

• Loosely coupled communication stack

1. Incoming packet
2. Put packet in buffer
3. Call services
4. Process the header

Post event

Out going packet
Discussion

- Reprogramming sensor nodes
  - 40 nodes dynamic distributed alarm system
  - Manual wired reprogramming complete system image
    - One node >> 30sec
    - 40 node >> 30 min
  - Over the air reprogramming a single component of application
    - 2 min
Discussion (cont.)

- Program typically much smaller than entire system image (1-10%)
  - Much quicker to transfer over the radio
Discussion (cont.)

- Code size
  - TinyOs < Contiki < Mantis

<table>
<thead>
<tr>
<th>Module</th>
<th>Code size (AVR)</th>
<th>Code size (MSP430)</th>
<th>RAM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>1044</td>
<td>810</td>
<td>10 +</td>
</tr>
<tr>
<td>Service layer</td>
<td>128</td>
<td>110</td>
<td>+ 4e + 2p</td>
</tr>
<tr>
<td>Program loader</td>
<td>-</td>
<td>658</td>
<td>8</td>
</tr>
<tr>
<td>Multi-threading</td>
<td>678</td>
<td>582</td>
<td>8 + s</td>
</tr>
<tr>
<td>Timer library</td>
<td>90</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>Replicator stub</td>
<td>182</td>
<td>98</td>
<td>4</td>
</tr>
<tr>
<td>Replicator</td>
<td>1752</td>
<td>1558</td>
<td>200</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3874</strong></td>
<td><strong>3876</strong></td>
<td><strong>230 + 4e + 2p + s</strong></td>
</tr>
</tbody>
</table>

- Size of the compiled code, in bytes
Conclusion

- Contiki – OS for “mote”-class sensor nodes
- Contiki explores trade-offs in
  - static vs dynamic
  - event-driven vs multi-threaded
- Loadable programs, works well
  - Static linking can be a problem
- Threads on an event-driven kernel
  - Multi-threading suitable for certain applications