TinyOS

Hardware Evolution
- Miniature hardware devices manufactured economically in large numbers
  - Microprocessors
  - Sensors
  - Wireless interfaces

Hardware Constraints
Severe constraints on power, size, and cost
- Slow CPU
- Low-bandwidth radio
- Limited memory
- Limited hardware parallelisms - CPU hit by many interrupts!
- Manage sleep modes in hardware components

Mica2 Mote
- Processor
  - Microcontroller: 7.4 MHz, 8 bit
  - Memory: 4KB data, 128 KB program
- Radio
  - Max 38.4 Kbps
- Sensors
  - Light, temperature, acceleration, acoustic, magnetic...
- Power
  - <1 week on two AA batteries in active mode
  - >1 year battery life on sleep modes!

Software Challenges
- Small memory footprint
- Efficiency - power and processing
- Concurrency-intensive operations
- Diversity in applications and platform → efficient modularity
  - Support reconfigurable hardware and software

Traditional OS
- Multi-threaded
- Preemptive scheduling
- Threads:
  - ready to run;
  - executing on the CPU;
  - waiting for data.
Pros and Cons of Traditional OS

- Multi-threaded + preemptive scheduling
  - Blocked threads waste memory
  - Context switch overhead
- I/O
  - Blocking I/O: waste memory on blocked threads
  - Polling (busy-wait): waste CPU cycles and power

Existing Embedded OS

<table>
<thead>
<tr>
<th>Name</th>
<th>Code Size</th>
<th>Target CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>pOSEK</td>
<td>2K</td>
<td>Microcontrollers</td>
</tr>
<tr>
<td>pSOSytem</td>
<td>286K</td>
<td>Pentium -&gt; ARM Thumb</td>
</tr>
<tr>
<td>VxWorks</td>
<td>&gt;100K</td>
<td>Pentium II -&gt; Strong ARM</td>
</tr>
<tr>
<td>QNX Nutrino</td>
<td>&gt;100K</td>
<td>Pentium II -&gt; SH4</td>
</tr>
<tr>
<td>QNX RealTime</td>
<td>100K</td>
<td>Pentium II -&gt; SH4</td>
</tr>
<tr>
<td>OS-9</td>
<td>10K</td>
<td>Pentium -&gt; SH4</td>
</tr>
<tr>
<td>Chorus OS</td>
<td>10K</td>
<td>Pentium -&gt; Strong ARM</td>
</tr>
<tr>
<td>ARIEL</td>
<td>19K</td>
<td>SH2, ARM Thumb</td>
</tr>
<tr>
<td>Creem</td>
<td>380 bytes</td>
<td>ATMEL 8051</td>
</tr>
</tbody>
</table>

- QNX context switch = 2400 cycles on x86
- pOSEK context switch > 40 µs
- Cream -> no preemption

TinyOS Solutions

- Efficient modularity
  - Application = scheduler + graph of components
  - Compiled into one executable
  - Only needed components are compiled/loaded
- Concurrency: event-driven architecture

Typical Application

- Events handle interrupts
  - Interrupts trigger lowest level events
  - Events can signal events, call commands, or post tasks
  - Tasks perform deferred computations
  - Interrupts preempt tasks and interrupts

Multiple Data Flows

- Respond quickly: sequence of event/command through the component graph.
  - Immediate execution of function calls
  - e.g., get bit out of radio hw before it gets lost.
- Post tasks for deferred computations.
  - e.g., encoding.
- Events preempt tasks to handle new interrupts.
Receiving a Message

Timing diagram of event propagation
(step 0-6 takes about 95 microseconds total)

Scheduling
- Interrupts preempt tasks
  - Respond quickly
  - Event/command implemented as function calls
- Task cannot preempt tasks
  - Reduce context switch → efficiency
  - Single stack → low memory footprint
- TinyOS 2 supports pluggable task scheduler (default: FIFO).
  - Scheduler puts processor to sleep when
    - no event/command is running
    - task queue is empty

Space Breakdown...
Code size for ad hoc networking application

Power Breakdown...

<table>
<thead>
<tr>
<th>Component</th>
<th>Active</th>
<th>Idle</th>
<th>Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>5 mA</td>
<td>2 mA</td>
<td>5 μA</td>
</tr>
<tr>
<td>Radio</td>
<td>7 mA (TX)</td>
<td>4.5 mA (RX)</td>
<td>5 μA</td>
</tr>
<tr>
<td>EE-Prom</td>
<td>3 mA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LED’s</td>
<td>4 mA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Photo Diode</td>
<td>200 μA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Temperature</td>
<td>200 μA</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Advantages
- Small memory footprint
  - Only needed components are compiled/loaded
  - Single stack for tasks
- Power efficiency
  - Put CPU to sleep whenever the task queue is empty
  - TinyOS 2 provides efficient power management for peripherals and microprocessors.
- Efficient modularity
  - Event/command interfaces between components
  - Event/command implemented as function calls
- Concurrency-intensive operations
  - Event/command + tasks

Time Breakdown...

<table>
<thead>
<tr>
<th>Components</th>
<th>Packet reception</th>
<th>Task overhead</th>
<th>CPU Utilization</th>
<th>Energy [nJ/Bit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0.06%</td>
<td>0.20%</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>Packet</td>
<td>1.12%</td>
<td>0.51%</td>
<td>7.58</td>
<td></td>
</tr>
<tr>
<td>Radio transfer</td>
<td>26.87%</td>
<td>12.16%</td>
<td>182.38</td>
<td></td>
</tr>
<tr>
<td>Radio decode thread</td>
<td>5.48%</td>
<td>2.48%</td>
<td>37.2</td>
<td></td>
</tr>
<tr>
<td>RFM</td>
<td>66.48%</td>
<td>30.08%</td>
<td>451.17</td>
<td></td>
</tr>
<tr>
<td>Radio Reception</td>
<td>5.48%</td>
<td>2.48%</td>
<td>37.2</td>
<td></td>
</tr>
<tr>
<td>Idle</td>
<td>100.00%</td>
<td>54.75%</td>
<td>2028.88</td>
<td></td>
</tr>
</tbody>
</table>

- 50 cycle task overhead (6 byte copies)
- 10 cycle event overhead (1.25 byte copies)
Disadvantages

- Lack preemptive real-time scheduling
  - Urgent task may wait for non-urgent ones
- Lack flexibility
  - Static linking only
  - Cannot change parts of the code dynamically
- Virtual memory?

More

- Multi-threaded vs. event-driven architectures
  - Lack empirical comparison against existing OSes
  - A "standard" OS is more likely to be adopted by industry
  - Jury is still out…
- Alternative: Native Java Virtual Machine?
  - Java programming
  - Virtual machine provides protection
  - Example: Sun SPOT

Reading

- J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, *System Architecture Directions for Network Sensors*