Critiques

- 1/2 page critiques of research papers
- Due at 10am on the class day (hard deadline)
- Email Yehan yehan.ma@wustl.edu in plain txt
- Back-of-envelop notes - NOT whole essays

- Critique #1
  - Due on 1/26 (Thursday)
IoT Operating Systems

Chenyang Lu
TinyOS and nesC

- TinyOS: OS for wireless sensor networks.
- nesC: programming language for TinyOS.
Mica2 Mote

- 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!

Epic Core

RAM 10 KB
Flash 48 KB
TI MSP430
Clock 4/8 MHz

I/O (some shared)
8 ADC (12 bit)
2 DAC (12 bit)
1 I2C
1 JTAG
1 1-Wire
2 SPI
2 UART
8 general, 8 interrupt, and 5 special pin connectors

CC2420 radio
802.15.4
6LoWPAN/IPv6

Typical sleep current 9μA at 3V, radio active ~20mA

Unique hardware ID

16 MB Flash memory

2.5 x 2.5 cm

3 V
TelosB

- Six major I/O devices
- Possible Concurrency
  - I²C, SPI, ADC
- Energy Management
  - Turn peripherals on only when needed
  - Turn off otherwise
Hardware Constraints

Severe constraints on power, size, and cost:
- slow microprocessor
- low-bandwidth radio
- limited memory
- limited hardware parallelism → CPU hit by many interrupts!
- manage sleep modes in hardware components
Software Challenges

- **Small** memory footprint
- **Efficiency** - power and processing
- **Concurrency-intensive** operations
- Diversity in applications & 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.
Limitations of Traditional OS

- Multi-threaded + preemptive scheduling
  - Preempted threads waste memory
  - Context switch overhead

- I/O
  - Blocking I/O: waste memory on blocked threads
  - Polling (busy-wait): waste CPU cycles and power
Example: Preemptive Priority Scheduling

- Each process has a fixed priority (1 highest);
- $P_1$: priority 1; $P_2$: priority 2; $P_3$: priority 3.
Context Switch

process 1
process 2

... memory

PC
registers

CPU
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>pSOSSystem</td>
<td></td>
<td>PII-&gt;ARM Thumb</td>
</tr>
<tr>
<td>VxWorks</td>
<td>286K</td>
<td>Pentium -&gt; Strong ARM</td>
</tr>
<tr>
<td>QNX Nutrino</td>
<td>&gt;100K</td>
<td>Pentium II -&gt; NEC</td>
</tr>
<tr>
<td>QNX RealTime</td>
<td>100K</td>
<td>Pentium II -&gt; SH4</td>
</tr>
<tr>
<td>OS-9</td>
<td></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>560 bytes</td>
<td>ATMEL 8051</td>
</tr>
</tbody>
</table>

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

TinyOS Solutions

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

D. Culler et. Al., TinyOS boot camp presentation, Feb 2001

application

routing

Messaging Layer

Radio Packet

Radio Byte (MAC)

RFM

sensing application

Temp

photo

ADC

i2c

clocks

SW

HW

Chenyang Lu
Two-level Scheduling

- 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 before it gets lost.

- Post tasks for deferred computations.
  - e.g., encoding.

- Events preempt tasks to handle new interrupts.

Chenyang Lu
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 $\rightarrow$ efficiency
  - Single stack $\rightarrow$ 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

- Interrupts
- Message Dispatch
- Initialization
- C-Runtime
- Light Sensor
- Clock
- Scheduler
- Led Control
- Messaging Layer
- Packet Layer
- Radio Interface
- Routing Application
- Radio Byte Encoder

Scheduler: 144 Bytes code
Totals: 3430 Bytes code
226 Bytes data

D. Culler et. Al., TinyOS boot camp presentation, Feb 2001
Lithium Battery runs for 35 hours at peak load and years at minimum load!
  - That’s three orders of magnitude difference!

A one byte transmission uses the same energy as approx 11000 cycles of computation.
### Time Breakdown...

<table>
<thead>
<tr>
<th>Components</th>
<th>Packet reception work breakdown</th>
<th>CPU Utilization</th>
<th>Energy (nj/Bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>0.05%</td>
<td>0.20%</td>
<td>0.33</td>
</tr>
<tr>
<td>Packet</td>
<td>1.12%</td>
<td>0.51%</td>
<td>7.58</td>
</tr>
<tr>
<td>Radio handler</td>
<td>26.87%</td>
<td>12.16%</td>
<td>182.38</td>
</tr>
<tr>
<td>Radio decode thread</td>
<td>5.48%</td>
<td>2.48%</td>
<td>37.2</td>
</tr>
<tr>
<td>RFM</td>
<td>66.48%</td>
<td>30.08%</td>
<td>451.17</td>
</tr>
<tr>
<td>Radio Reception</td>
<td>-</td>
<td>-</td>
<td>1350</td>
</tr>
<tr>
<td>Idle</td>
<td>-</td>
<td>54.75%</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100.00%</strong></td>
<td><strong>100.00%</strong></td>
<td><strong>2028.66</strong></td>
</tr>
</tbody>
</table>

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

- Small memory footprint
  - Only needed components are complied/loaded
  - Single stack for tasks

- Power efficiency
  - Put CPU to sleep whenever the task queue is empty
  - TinyOS 2 (ICEM) provides power management for peripherals.

- Efficient modularity
  - Event/command interfaces between components
  - Event/command implemented as function calls

- Concurrency-intensive operations
  - Event/command + tasks
Critiques

- No protection barrier between kernel and applications

- No preemptive scheduling → a real-time task may wait for non-urgent ones

- Static linking → cannot change parts of the code dynamically

- Virtual memory?