Operating Systems for Network of Sensors

Simon Han
simonhan@ee.ucla.edu
Embedded Software Development

• Cross-compiler
  – A compiler which runs on one platform and produces code for another.
  – Javac: compile java source into java bytecode
  – Avr-gcc: compile C source into avr instructions

• Tool-chain
  – Compiler + binutils (linker, assembler) + libraries.

• Development cycle
  1. Coding
  2. Cross-compile
  3. Upload binary
  4. Testing
Characteristics of Network Sensors

• Small physical size and low power consumption

• Concurrency-intensive operation
  – multiple flows, not wait-command-respond

• Limited Physical Parallelism and Controller Hierarchy
  – primitive direct-to-device interface
  – Asynchronous and synchronous devices

• Diversity in Design and Usage
  – application specific, not general purpose
  – huge device variation
  => efficient modularity
  => migration across HW/SW boundary

• Robust Operation
  – numerous, unattended, critical
  => narrow interfaces
Tiny OS Concepts

- **Scheduler + Graph of Components**
  - constrained two-level scheduling model: threads + events

- **Component:**
  - Commands,
  - Event Handlers
  - Frame (storage)
  - Tasks (concurrency)

- **Constrained Storage Model**
  - frame per component, shared stack, no heap

- **Very lean multithreading**
- **Efficient Layering**
Application = Graph of Components

Example: ad hoc, multi-hop routing of photo sensor readings

3450 B code
226 B data

Graph of cooperating state machines on shared stack
Programming TinyOS

- TinyOS 1.0 is written in an extension of C, called nesC
- Applications are too!
  - just additional components composed with the OS components
- Provides syntax for TinyOS concurrency and storage model
  - commands, events, tasks
  - local frame variable
- Rich Compositional Support
  - separation of definition and linkage
  - robustness through narrow interfaces and reuse
  - interpositioning
- Whole system analysis and optimization
Event-Driven Sensor Access Pattern

- clock event handler initiates data collection
- sensor signals data ready event
- data event handler calls output command
- device sleeps or handles other activity while waiting
- conservative send/ack at component boundary

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 200);
}

event result_t Timer.fired() {
    return call sensor.getData();
}

event result_t sensor.dataReady(uint16_t data) {
    display(data)
    return SUCCESS;
}
TinyOS Commands and Events

```c
{
    ...
    status = call CmdName(args)
    ...
}
```

```c
command CmdName(args) {
    ...
    return status;
}
```

```c
event EvtName)(args) {
    ...
    return status;
}
```

```c
{
    ...
    status = signal EvtName(args)
    ...
}
```
TinyOS Execution Contexts

- Events generated by interrupts preempt tasks
- Tasks do not preempt tasks
- Both essential process state transitions

Hardware

Interrupts

Commands

Events
TASKS

• provide concurrency internal to a component
  – longer running operations
• are preempted by events
• able to perform operations beyond event context
• may call commands
• may signal events
• not preempted by tasks

```c
{ 
  ...  
  post TskName();  
  ... 
}
```
Components

A component specifies a set of interfaces by which it is connected to other components
  – provides a set of interfaces to others
  – uses a set of interfaces provided by others

• Modules
  – provide code that implements one or more interfaces and internal behavior

• Configurations
  – link together components to yield new component

• Interface
  – logically related set of commands and events
  – Interfaces are bi-directional (include commands and events)
Example Application: Blink

module BlinkM {
    provides {
        interface StdControl; }
    uses {
        interface Clock;
        interface Leds; }
}

// Continued next page
Module implementation

implementation {
    bool state;
    command result_t StdControl.init() {
        state = FALSE;
        return SUCCESS;
    }
    command result_t StdControl.start() {
        return call Clock.setRate(TOS_I1PS, TOS_S1PS);  }
    command result_t StdControl.stop() {
        return call Clock.setRate(TOS_I0PS, TOS_S0PS);  }
    event result_t Clock.fire() {
        state = !state;
        if (state)
            call Leds.redOn();
        else
            call Leds.redOff();
        return SUCCESS;
    }
}
Top level configuration

- Configuration wires all components together
- Arrows bind interfaces (on the left) to implementation (on the right)

configuration Blink {
// this module does not provide any interface
}
implementation {
    components Main, BlinkM, ClockC, LedsC;
    Main.StdControl -> BlinkM.StdControl;
    BlinkM.Clock -> ClockC.Clock;
    BlinkM.Leds -> LedsC.Leds;
}
Compiling, Installing and Running

• Compiling:
  \% make mica

• Installing:
  \% make mica install

• Running:
  \% Live Demo!
TinyOS: Pros and Cons

• Pros
  – Static memory allocation => resource guarantee.
  – Non-preemptive scheduling => minimal memory requirement.
  – Modulized language => allow independent software development.

• Cons
  – A brand new programming language
  – Static memory allocation => resource over subscription.
  – Not really an OS.
    » Blur hardware abstraction.
    » No resource management.
  – Simple concurrency model => lots critical sections.