CS649
Sensor Networks
Lecture 6: OS and Language Support

Andreas Terzis
http://hinrg.cs.jhu.edu/wns05/
Outline

- Operating Systems for Sensor Nodes
  - TinyOS
- Language support for Sensor Nodes
  - nesC
Characteristics of Network Sensors

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: tasks + events
- Component:
  - Commands
  - Event Handlers
  - Frame (storage)
  - Tasks (concurrency)
- Constrained Storage Model
  - Frame per component, Shared stack, No heap
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

Execution driven by interrupts
TOS Execution Model

- Commands request action
  - ack/nack at every boundary
- Call cmd or post task
- Events notify occurrence
  - HW intrpt at lowest level
  - May signal events
  - call cmds
  - post tasks
- Tasks provide logical concurrency
  - preempted by events
- Migration of HW/SW boundary

Diagram:
- application comp
  - data processing
  - message-event driven
  - event-driven packet-pump
- active message
  - crc
- Radio Packet
  - event-driven byte-pump
- Radio byte
  - encode/decode
  - event-driven bit-pump
- RFM
Programming TinyOS - nesC

- TinyOS 1.x 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 variables
- 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

```c
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;
}
```

- 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
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)  
    ...  
}
```
Split-phase abstraction of HW

- Command synchronously initiates action
- Device operates concurrently
- Signals event(s) in response
  - ADC
  - Clock
  - Send (UART, Radio, ...)
  -Recv – depending on model
  - Coprocessor
- Higher level (SW) processes don’t wait or poll
  - Allows automated power management

- Higher level components behave the same way
  - Tasks provide internal concurrency where there is no explicit hardware concurrency
  - Components (even subtrees) replaced by HW and vice versa
TinyOS Execution Contexts

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

- Passed as arguments to command or event handler
  - Don’t make intra-node communication heavy-weight
- If queuing is appropriate, implement it
  - Send queue
  - Receive queue
  - Intermediate queue
- Bounded depth, overflow is explicit
  - Most components implement 1-deep queues at the interface
- If you want shared state, created an explicit component with interfaces to it.
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();  
  ...  
}
```

```c
task void TskName {  
  ...  
}
```
Task Scheduling

• Currently simple FIFO scheduler
• Bounded number of pending tasks
• When idle, shuts down node (except clock)

• Uses non-blocking task queue data structure

• Simple event-driven structure + control over complete application/system graph
  • instead of complex task priorities and IPC
Communication

• Essentially just like a call
• Receive is inherently asynchronous
• Don’t introduce potentially unbounded storage allocation
• Avoid copies and gather/scatter (mbuf problem)
Tiny Active Messages

- Sending
  - Declare buffer storage in a frame
  - Request Transmission
  - Name a handler
  - Handle Completion signal
- Receiving
  - Declare a handler
  - Firing a handler
    - automatic
    - behaves like any other event
- Buffer management
  - strict ownership exchange
  - tx: done event => reuse
  - rx: must rtn a buffer
Sending a message

```c
bool pending;
struct TOS_Msg buf;
command result_t IntOutput.output(uint16_t value) {
  IntMsg *message = (IntMsg *)buf.data;
  if (!pending) {
    pending = TRUE;
    message->val = value;
    message->src = TOS_LOCAL_ADDRESS;
    if (call Send.send(TOS_BCAST_ADDR, sizeof(IntMsg), &buf))
      return SUCCESS;
    pending = FALSE;
  }
  return FAIL;
}
```

- Refuses to accept command if buffer is still full or network refuses to accept send command
- User component provide structured msg storage
Send done event

```c
event result_t IntOutput.sendDone(TOS_MsgPtr msg,
                                  result_t success)
{
    if (pending && msg == &buf) {
        pending = FALSE;
        signal IntOutput.outputComplete(success);
    }
    return SUCCESS;
}
```

- Send done event fans out to all potential senders
- Originator determined by match
  - free buffer on success, retry or fail on failure
- Others use the event to schedule pending communication
Receive Event

```c
event TOS_MsgPtr ReceiveIntMsg.receive(TOS_MsgPtr m) {
    IntMsg *message = (IntMsg *)m->data;
    call IntOutput.output(message->val);
    return m;
}
```

- Active message automatically dispatched to associated handler
  - knows the format, no run-time parsing
  - performs action on message event
- Must return free buffer to the system
  - typically the incoming buffer if processing complete
A Complete Application

- SenseToRfm
- IntToRfm
- Timer
- photo
- phototemp
- SW
- HW
- ClockC
- ADC
- Mica2 stack
- Interpostioning
- ChipCon

- generic comm
- AMStandard
- RadioPacket
- UARTPacket
- CRCfilter
- noCRCPacket
- MicaHighSpeedRadioM
- SecDedEncode
- ChannelMon
- RadioTiming
- SPIByteFIFO
- RandomLFSR
- UART
- SlavePin
- bit
- byte
- packet
- Spring 2006
- CS 649
Composition

- 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
- Interfaces are bi-directional
  - include commands and events
- Interface methods are the external namespace of the component

```
provides
  interface StdControl;
  interface Timer:
uses
  interface Clock
```
Components

- 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

```plaintext
StdControl.nl

interface StdControl {
    command result_t init();
    command result_t start();
    command result_t stop();
}

Clock.nl

interface Clock {
    command result_t setRate(char interval, char scale);
    event result_t fire();
}
```

Spring 2006

CS 649
Example top level configuration

configuration SenseToRfm {
    // this module does not provide any interface
}
implementation {
    components Main, SenseToInt, IntToRfm, ClockC, Photo as Sensor;

    Main.StdControl -> SenseToInt;
    Main.StdControl -> IntToRfm;

    SenseToInt.Clock -> ClockC;
    SenseToInt.ADC -> Sensor;
    SenseToInt.ADCControl -> Sensor;
    SenseToInt.IntOutput -> IntToRfm;
}
A Multihop Routing Example

```
Component          | Code size | Data size |
-------------------|-----------|-----------|
                  | inlined   | noninlined|
Application components
SurgeM             | 236       | 240       | 44       |
Multipart communication
AMPromiscuous      | 676       | 526       | 9        |
MultihopM          | 2104      | 2884      | 223      |
NoCRCPacket        | 162       | 484       | 50       |
QueuedSend         | 298       | 852       | 461      |
Radio stack
ChannelMonC        | 58        | 158       | 9        |
CrcFilter          | _         | 34        | 0        |
MicaHighSpeedRadioM| 1272      | 1250      | 61       |
PktM               | _         | 82        | 1        |
RadioTimingC        | _         | 56        | 0        |
SecSecrEncoding     | 196       | 684       | 3        |
SpiBytePico        | 172       | 352       | 2        |
Sensor acquisition
ADCM               | 238       | 260       | 2        |
PhotoTempM          | _         | 360       | 2        |
Miscellaneous
TimerM             | 1956      | 1734      | 118      |
NoLeds              | _         | 18        | 0        |
RandomLDRR          | 134       | 134       | 6        |
RealMain            | _         | 72        | 0        |
Hardware presentation
Ledsc              | _         | _94       | 1        |
HPLAOCC            | 80        | 188       | 11       |
HPLClock           | _         | 60        | 0        |
HPLinit            | _         | 10        | 0        |
HPLInterrupt       | _         | 22        | 0        |
HPLPktC            | _         | 66        | 0        |
HPLslavePinC       | _         | 28        | 0        |
HPLUARTM           | _         | 58        | 0        |
SlavePinM          | 36        | 124       | 1        |
UARTM              | 122       | 136       | 1        |
other              | 3206      | 2678      | 45       |
Totals:            | 11046     | 13714     | 1050     |
```
Given the framework, what’s the system?

- Core Subsystems
  - Simple Display (LEDS)
  - Identity
  - Timer
  - Bus interfaces (i2c, SPI, UART, 1-wire)
  - Data Acquisition
  - Link-level Communication
  - Power management
  - Non-volatile storage
- Higher Level Subsystems
  - Network-level communication
    - Broadcast, Multihop Routing
  - Time Synchronization, Ranging, Localization
  - Network Programming
  - Neighborhood Management
  - Catalog, Config, Query Processing, Virtual Machine
Typical Operational Mode

• Major External Events
  • Trigger collection of small processing steps (tasks and events)
  • May have interval of hard real time sampling
    • Radio
    • Sensor
  • Interleaved with moderate amount of processing in small chunks at various levels
• Periods of sleep
  • Interspersed with timer mgmt
Comments

- Positive

- Negative