Internet of Things Laboratory

November 13, 2015

P.Gjanci, G.Koutsandria, D. Spenza

Contacts

- Gjanci: gjanci@di.uniroma1.it
- Koutsandria: koutsandria@di.uniroma1.it
- Spenza: spenza@di.uniroma1.it
  - Tel: 06-49918430
  - Room: 333
  - Slides: www.dsi.uniroma1.it/~spenza/
- SENSES lab
  - http://senseslab.di.uniroma1.it

Lessons Schedule

09.11.2015
8:30-12:00
Lesson 1

10.11.2015
17:15-19:30
Tutoring session

13.11.2015
14:00-17:30
Lesson 2

23.11.2015
08:30-12:00
Lesson 3

24.11.2015
17:15-19:30
Tutoring/project?

27.11.2015
14:00-17:30
Final project

Outline

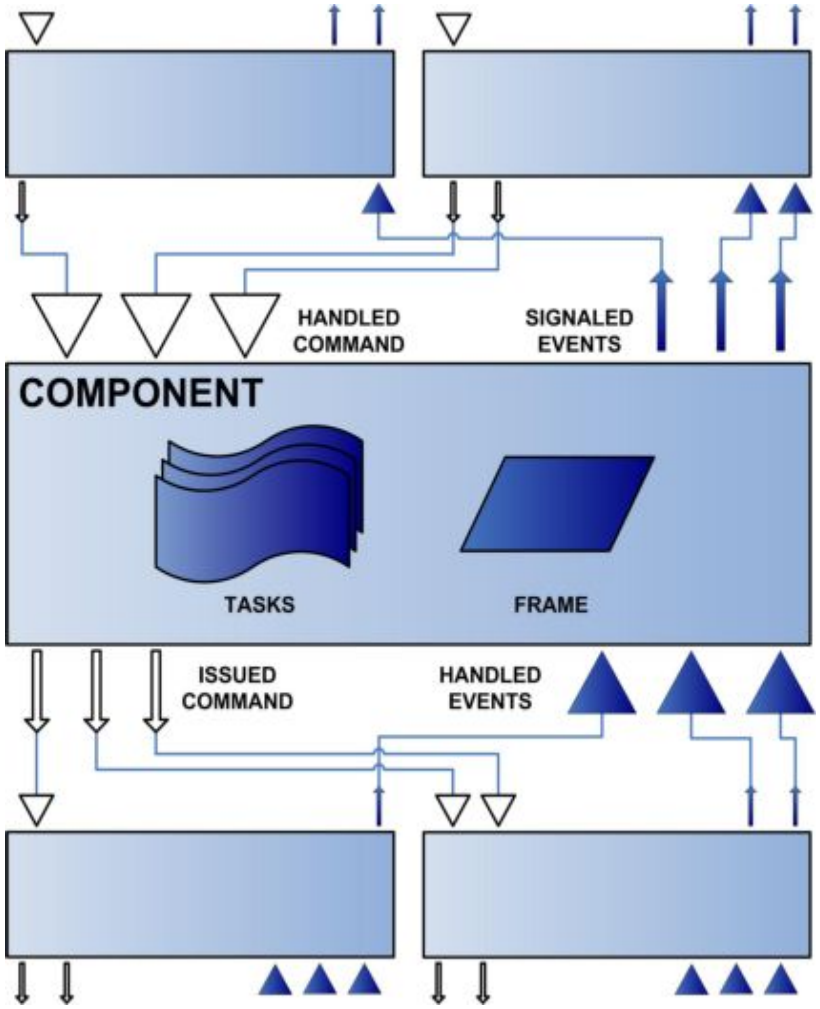
- Tasks and split-phase operation
- TinyOS Printf library
- The BlinkToRadio Application
- Mote-PC serial communication
- BaseStation
- SerialForwarder
- Oscilloscope

In the last episode..

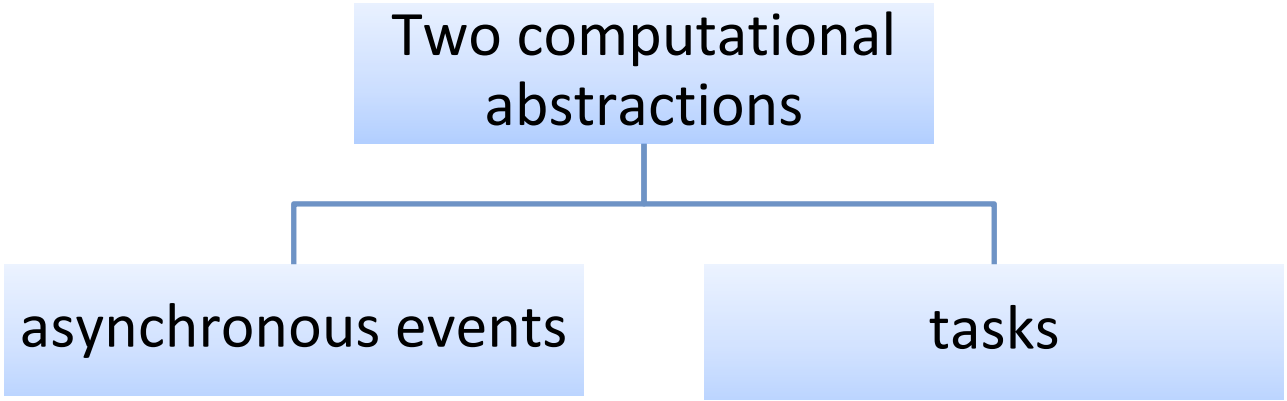
- **NesC**: C dialect
- **TinyOS**: event driven OS
- **split-phase**: call →callback (event)
- **kernel** with two hierachical levels: tasks and events
- **single stack system**: local variable in the stack, binary code are stored contiguously in memory

In the last episode

1. Application consists of *components* ***wired***
2. Two scopes:
   - Specification (interfaces' name)
   - Implementation
3. App ***provides*** and ***uses*** interfaces
4. Interfaces ↔ functionalities
5. Interfaces are bidirectional
   - Commands implemented by interface's provider
   - Events implemented by the interface's user



TinyOS Execution Model



- can run preemptively (async)
- interrupt handlers
- **race conditions**!

- schedule a function to be called later
- run in a single execution context
- no preemption!
- FIFO

Sync code should be kept short

Blink application: event handler for Timer0.fired()

```
event void Timer0.fired() {
 uint32_t i;
 call Leds.led0Toggle();
}
```

Let's introduce some latency..

```
event void Timer0.fired() {
    uint32_t i;

    for (i = 0; i < BIG_NUMBER;
        i++) { }
    call Leds.led0Toggle();
}
```

The `Timer` interface is synchronous → Long computations interfere with timers operations

Tasks

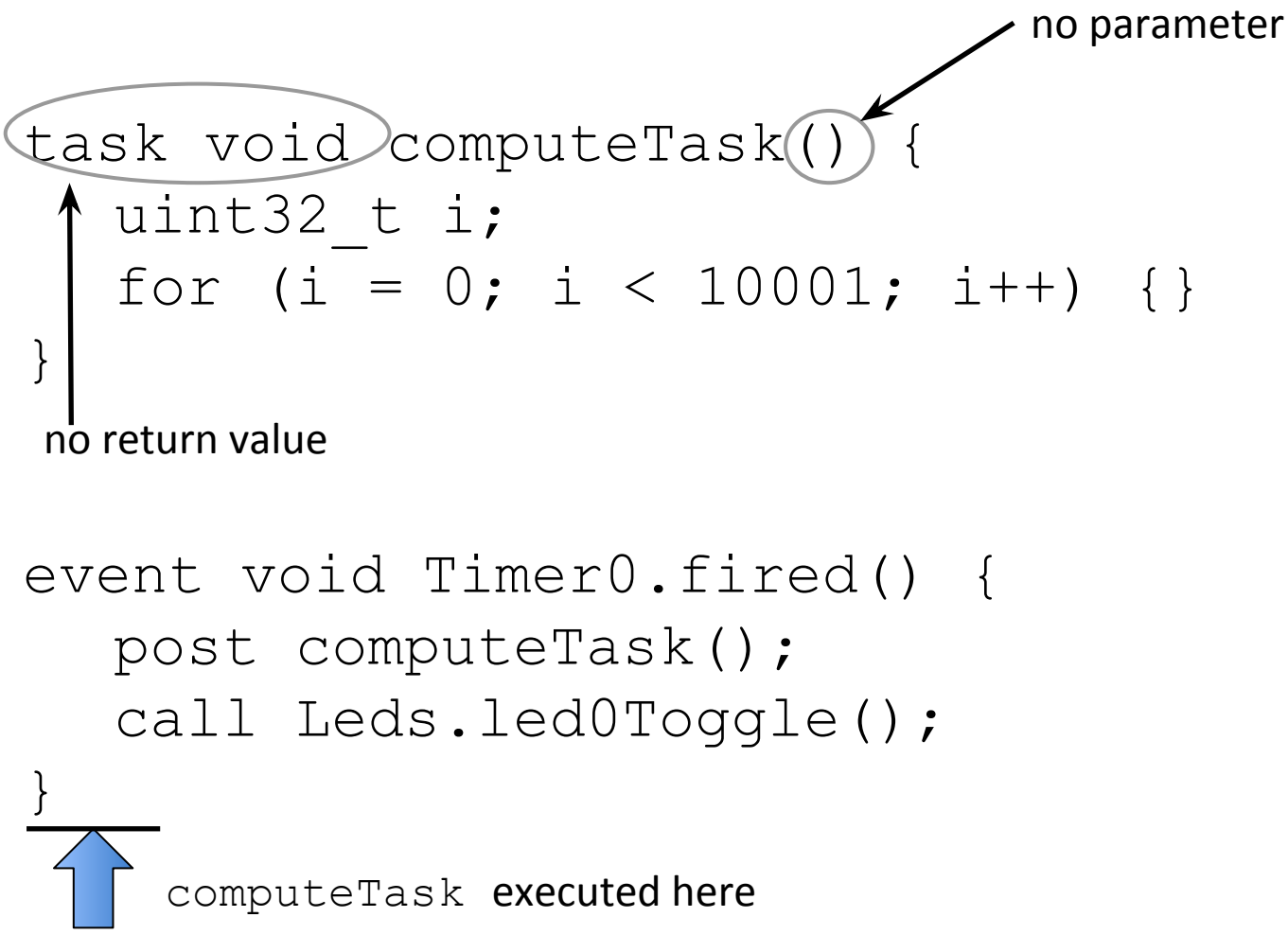
Usage:

```
task void computeTask() {
   uint32_t i;
   for (i = 0; i < 10001; i++) {}
}

event void Timer0.fired() {
   post computeTask();
   call Leds.led0Toggle();
}
```

- Dispatches a task for later execution
- Internal task queue processed in FIFO order
- Task cannot be poster more than once

Tasks Usage

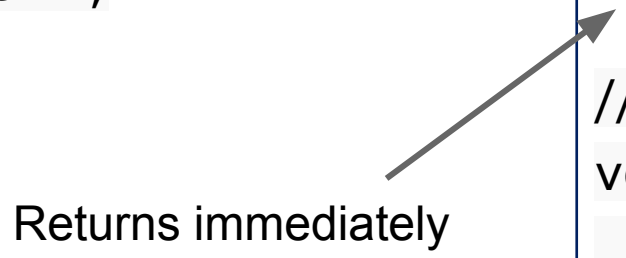

no parameter
task void computeTask() {
uint32_t i;
for (i = 0; i < 10001; i++) {}
}
no return value
event void Timer0.fired() {
post computeTask();
call Leds.led0Toggle();
}
computeTask executed here

Split-Phase

No blocking operations allowed.

Blocking	Split-Phase

`if (send() == SUCCESS) {`
`  sendCount++;`
`}`
	`// start phase`
	`send();`
	
	`//completion phase`
	`void sendDone(error_t err) {`
	`  if (err == SUCCESS) {`
	`  sendCount++;`
	`  }`
	`}`



Recap: Concurrency

- Commands by default are **sync**: no preemption, blocks.
- Also tasks are **non** preemptive.
- **But** interrupts block the execution of a code and starts running preemptively.
- Functions that can run preemptively are declare **async** (e.g., component LedsC).
- Commands and events of async functions are async as well.

The only way for an async command to call a sync function is via tasks.

- Posting a task is an async event, while executing it is sync.
- How to manage preemption? -> using the **atomic** keywords
  - TinyOs guarantee that atomic code is not modified during its execution

TinyOS Printf Library

- Located in tos/lib/printf
- Used to debug TinyOS applications by printing messages over the serial port
- Reference: http://tinyos.stanford.edu/tinyos-wiki/index.php/The_TinyOS_printf_Library
- How to use it:
  - include component PrintfC in the top-level configuration file
  - include “printf.h” header file in any component that calls it

- Start the PrintfClient by running the following command:

```
java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSBXXX:telosb
```

Tinyos Printf Library

- Include the #include "printf.h" header file in every component in which you would like to call the printf() command
  - In the implementation file
    - #include "printf.h"
- In the Makefile add:
  - The tos/lib/printf/2_0_2 directory must be in your include path
    - CFLAGS += -I$(TINYOS_OS_DIR)/tos/lib/printf/2_0_2
  - Define the size of the printf buffer
    - CFLAGS+=-DPRINTF_BUFFER=6042
  - Configuration file:
    - #define NEW_PRINTF_SEMANTICS
    - components PrintfC;

Exercise

Modify the blink application to print every time that a timer fires

BlinkToRadio

Today Application: BlinkToRadio

A one-timer version of Blink application that sends the counter value over the radio channel.

First Step:

Implement a version of Blink using a single timer and the `set` function (look at Leds component implementation).

BlinkToRadio Application

- A counter is incremented every second
- Whenever the timer fires, the value of the counter is sent over a radio message
- Whenever a radio message is received, the three least significant bits of the counter in the message payload are displayed on the LEDs

Sending the counter value

- Define a message format to send data over the radio

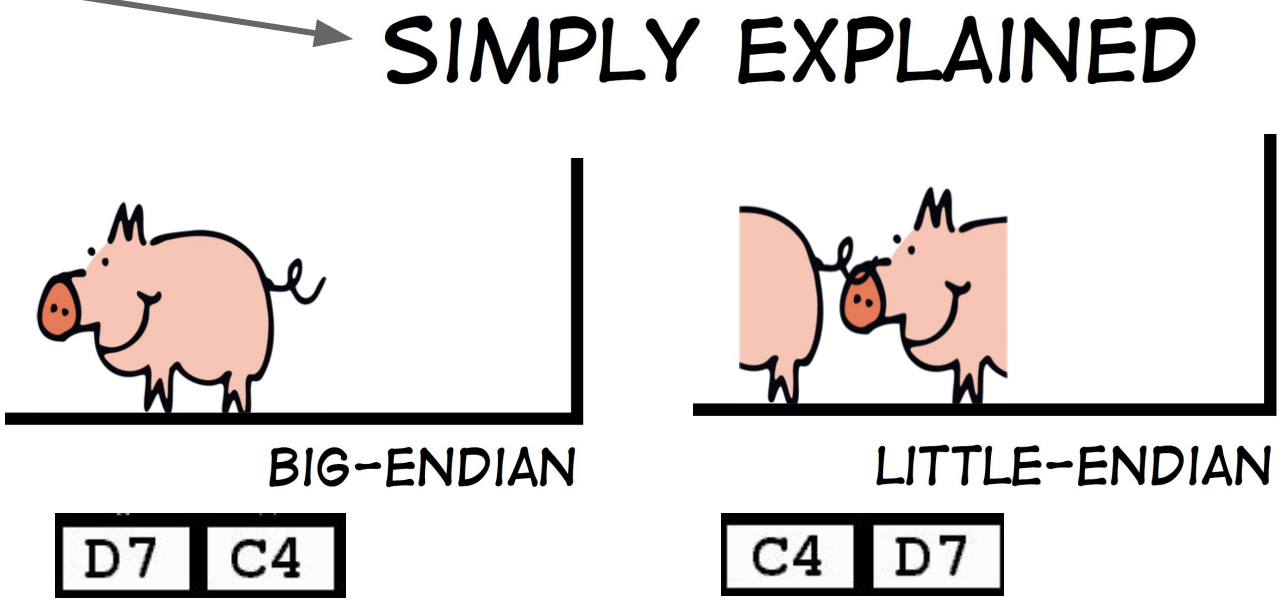
```
typedef nx_struct BlinkToRadioMsg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
} BlinkToRadioMsg;
```

- Why a struct?

```
uint16_t x = data[0] << 8 + data[1])
```

nesC external types

- nx_ data types
- Have the same representation on all platforms
- No need to manually address alignment and **endianness**



Communication Interfaces

- TinyOS provides high-level communication interfaces
  - Similar for radio and serial communication
- Basic interfaces:
  - **Packet**: Set/get payload of TinyOS message_t pckts
  - **Send**: Send packet by calling send() command
  - **Receive**: Reception of pckts signaled by receive() event
- Active Message interfaces allow for multiplexing:
  - **AMPacket**: Provides source/destination address to pckt
  - **AMSend**: Send packet to destination address

Communication Interface

TinyOS provides:

i. *Communication services by means of interfaces*
ii. *Components implementing such interfaces*
iii. *Abstract data type* **`message_t`**

```
typedef nx_struct message_t {
    nx_uint8_t header[sizeof
(message_header_t)];
    nx_uint8_t data
[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof
(message_footer_t)];
    nx_uint8_t metadata[sizeof
(message_metadata_t)];
} message_t;
```

Read and Write through accessors and mutuator functions

Communication Interface

Basic Communication Interfaces in /tos/interfaces:

- **Packet**, Send, Receive, PacketAcknowledgments, RadioTimeStamping

- Active Message Interfaces: **AMPacket**, **AMSend**

- Basic components In /tos/system:
  1. **AMReceiverC**
  2. **AMSenderC**
  3. AMSnooperC
  4. AMSnoopingReceiverC
  5. **ActiveMessageAddressC**

ActiveMessageC for the telosb are all implemented by:
CC2420ActiveMessageC

Tutorial Link

docs.tinyos.net/tinywiki/index.php/Mote-mote_radio_communication

Exercise 1

- Assign an unique ID to your nodes
- Filter radio messages that are not sent to you

Excercise 2

- Node A sends the value of its counter to node B
- Node B displays the three least significant bits of the counter on the LEDs, updates the value of the counter and sends it back to node A
- Node A displays the three least significant bits of the counter on the LEDs, updates the value of the counter and sends it back to node B
- ….