Proposal Presentation

- In class on 2/9

- 13 min per group
  - Including 3 min for questions and discussions
  - 5-7 slides

- Your elevator pitch!

- Email Haoran your slides in advance
  - All use computer in classroom → reduce context switches
Proposal

- One proposal/team, one page
  - Team members
  - Concise description of project
  - Responsibilities of each member
  - Specific equipment needed

- Written proposal due: 2/9, 11:59pm
  - Email to Haoran
Critique #2


- Due on 2/14 (Tuesday)
Integrating Concurrency Control and Energy Management in Device Drivers

Chenyang Lu
Overview

- **Concurrency Control:**
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O

- **Energy Management:**
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using asynchronous I/O
Overview

- **Concurrency Control:**
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O

- **Energy Management:**
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O

```
Application
  read()
  write()
  read()
  read()

OS Flash Driver
  read()
  write()
  setPowerState()

Physical Flash
```
Overview

- **Concurrency Control:**
  - Concurrency of I/O operations alone, not of threads in general
  - Synchronous vs. Asynchronous I/O

- **Energy Management:**
  - Power state of devices needed to perform I/O operations
  - Determined by pending I/O requests using Asynchronous I/O

The more workload information an application can give the OS, the more energy it can save.
Motivation

- Difficult to manage energy in traditional OS
  - Hard to tell OS about future application workloads
  - API extensions for hints?
Existing OS Approaches

- **Dynamic CPU Voltage Scaling**
  - Vertigo - Application workload classes
  - Grace OS - Explicit real-time deadlines

- **Disk Spin Down**
  - Coop-IO - Application specified timeouts
Existing OS Approaches

- **Dynamic CPU Voltage Scaling**
  - Vertigo - Application workload classes
  - Grace OS - Explicit real-time deadlines

- **Disk Spin Down**
  - Coop-IO - Application specified timeouts

*Saving energy is a complex process*
Existing OS Approaches

- Dynamic CPU Voltage Scaling
  - Vertigo - Application workload classes
  - Grace OS - Explicit real-time deadlines

- Disk Spin Down
  - Coop-IO - Application specified timeouts

Saving energy is a complex process
A little application knowledge can help us a lot
Sensor Networks

- Domain in need of unique solution to this problem
  - Harsh energy requirements
  - Very small source of power (2 AA batteries)
  - Must run unattended from months to years
Sensor Networks

- Domain in need of unique solution to this problem
  - Harsh energy requirements
  - Very small source of power (2 AA batteries)
  - Must run unattended from months to years

- First generation sensornet OSes (TinyOS, Contiki...)
  - Push all energy management to the application
  - Optimal energy savings at cost of application complexity
ICEM: Integrated Concurrency and Energy Management

- Device driver architecture that automatically manages energy

- Introduces Power Locks, split-phase locks with integrated energy and configuration management

- Defines three classes of drivers: dedicated, shared, virtualized

- Provides a component library for building drivers

- Implemented in TinyOS 2.0 -- all drivers follow it
Advantages of ICEM

- Energy efficient – 98.4% as hand-tuned implementation
- Reduces code complexity – 400 vs. 68 lines of code
- Enables natural decomposition of applications
The TelosB Platform

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

![Diagram of TelosB Platform](image)
Representative Logging Application

Producer
Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Consumer
Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Flash

Sensors

Radio
Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Producer

Consumer

Flash

Sensors

Radio
Representative Logging Application

**Producer**

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Consumer**

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

**Flash**

**Sensors**

**Radio**
Representative Logging Application

**Producer**

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Consumer**

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

**Flash**

**Sensors**

**Radio**
Representative Logging Application

**Producer**

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Consumer**

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

**Flash**

**Sensors**

**Radio**
Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Producer

Consumer

Sensors

Flash

Radio
Every 12 hours:
For all new entries:
  Send current sample
  Read next sample

Consumer

Every 5 minutes:
  Write prior samples
  Sample photo active
  Sample total solar
  Sample temperature
  Sample humidity

Producer

Flash

Sensors

Radio
Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Producer

Consumer

Flash

Sensors

Radio

Representative Logging Application
Representative Logging Application

Producer

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

Consumer

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

Flash

Sensors

Radio
Hand-Tuned Application

Every 5 minutes:
   - Turn on SPI bus
   - Turn on flash chip
   - Turn on voltage reference
   - Turn on I^2C bus
   - Log prior readings
   - Start humidity sample
   - Wait 5ms for log
   - Turn off flash chip
   - Turn off SPI bus
   - Wait 12ms for vref
   - Turn on ADC
   - Start total solar sample
   - Wait 2ms for total solar
   - Start photo active sample
   - Wait 2ms for photo active
   - Turn off ADC
   - Turn off voltage reference
   - Wait 34ms for humidity
   - Start temperature sample
   - Wait 220ms for temperature
   - Turn off I^2C bus
Hand-Tuned Application

Every 5 minutes:

- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I²C bus
- Log prior readings
- Start humidity sample
- Wait 5ms for log
- Turn off flash chip
- Turn off SPI bus
- Wait 12ms for vref
- Turn on ADC
- Start total solar sample
- Wait 2ms for total solar
- Start photo active sample
- Wait 2ms for photo active
- Turn off ADC
- Turn off voltage reference
- Wait 34ms for humidity
- Start temperature sample
- Wait 220ms for temperature
- Turn off I²C bus
Hand-Tuned Application

Every 5 minutes:
- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I$^2$C bus
- Log prior readings
- Start humidity sample
  - Wait 5ms for log
  - Turn off flash chip
  - Turn off SPI bus
  - Wait 12ms for vref
  - Turn on ADC
  - Start total solar sample
    - Wait 2ms for total solar
  - Start photo active sample
    - Wait 2ms for photo active
    - Turn off ADC
    - Turn off voltage reference
  - Wait 34ms for humidity
  - Start temperature sample
    - Wait 220ms for temperature
  - Turn off I$^2$C bus
Hand-Tuned Application

Every 5 minutes:
- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I\textsuperscript{2}C bus
- Log prior readings
- Start humidity sample
- Wait 5ms for log
- Turn off flash chip
- Turn off SPI bus
- Wait 12ms for vref
- Turn on ADC
- Start total solar sample
- Wait 2ms for total solar
- Start photo active sample
- Wait 2ms for photo active
- Turn off ADC
- Turn off voltage reference
- Wait 34ms for humidity
- Start temperature sample
- Wait 220ms for temperature
- Turn off I\textsuperscript{2}C bus

ICEM Application

Every 5 minutes:
- Log prior readings
- Sample humidity
- Sample total solar
- Sample photo active
- Sample temperature
Split-Phase I/O Operations

- Implemented within a single thread of control
- Application notified of I/O completion through direct upcall
- Driver given workload information before returning control
- Example: read() → readDone()

```
void readDone(uint16_t val) {
    next_val = val;
    read();
}
```
ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface
Virtualized Device Drivers

- Provide only a **Functional** interface
- Assume multiple users
- **Implicit** concurrency control through buffering requests
- **Implicit** energy management based on pending requests
- Implemented for higher-level services that can tolerate long latencies
Dedicated Device Drivers

- Provide **Functional** and **Power Control** interfaces
- Assume a single user
- **No** concurrency control
- *Explicit* energy management
- Low-level hardware and bottom-level abstractions have a dedicated driver

<table>
<thead>
<tr>
<th>Energy:</th>
<th>Concurrency:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit</td>
<td>None</td>
</tr>
</tbody>
</table>
Shared Device Drivers

- Provide **Functional** and **Lock** interfaces
- Assume multiple users
- **Explicit** concurrency control through Lock request
- **Implicit** energy management based on pending requests
- Used by users with stringent timing requirements
ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface

- Power Locks, split-phase locks with integrated energy and configuration management
Power Locks

- HW-Specific Configuration
- Power Control

Dedicated Driver

Functional Lock

Power Locks
Power Locks

- Dedicated Driver
- HW-Specific Configuration
- Power Control
- Functional Lock
Power Locks

- Dedicated Driver
- HW-Specific Configuration
- Power Control
- Functional Lock
Power Locks

- HW-Specific Configuration
- Power Control
- Dedicated Driver
- Functional

Lock
Power Locks

- Lock
- HW-Specific Configuration
- Power Control
- Functional
- Dedicated Driver
Power Locks

Lock

Power Locks

HW-Specific Configuration

Power Control

Functional

Dedicated Driver
Power Locks

- Lock
- HW-Specific Configuration
- Power Control
- Functional
- Dedicated Driver
Power Locks

- Lock
- HW-Specific Configuration
- Power Control
- Functional

Dedicated Driver
Power Locks

- Lock
- HW-Specific Configuration
- Power Control

Dedicated Driver

1
2
3
Power Locks

- Dedication Driver
- Power Control
- HW-Specific Configuration

Functional

Dedicated Driver
Power Locks

- HW-Specific Configuration
- Power Control
- Functional
- Dedicated Driver
Power Locks

HW-Specific Configuration

Power Control

Functional

Dedicated Driver
ICEM Architecture

- Defines three classes of drivers
  - Virtualized – provide only functional interface
  - Dedicated – provide functional and power interface
  - Shared – provide functional and lock interface

- Power Locks, split-phase locks with integrated energy and configuration management

- Component library
  - Arbiters – manage I/O concurrency
  - Configurators – setup device specific configurations
  - Power Managers – provide automatic power management
**Lock** interface for concurrency control (FCFS, Round-Robin)

**ArbiterConfigure** interface for automatic hardware configuration

**DefaultOwner** interface for automatic power management
- **Lock** interface for concurrency control (FCFS, Round-Robin)
- **ArbiterConfigure** interface automatic hardware configuration
- **DefaultOwner** interface for automatic power management
- Implement **ArbiterConfigure** interface
- Call hardware specific configuration from dedicated driver
- Implement **DefaultOwner** interface
- Power down device when device falls idle
- Power up device when new lock request comes in
- Currently provide **Immediate** and **Deferred** policies
Msp430 USART (Serial Controller)

- Three modes of operation – SPI, I²C, UART
Msp430 USART (Serial Controller)

- Three modes of operation – SPI, I²C, UART
**Shared Driver Example**

- **Msp430 USART (Serial Controller)**
  - Three modes of operation – SPI, I²C, UART
Msp430 USART (Serial Controller)

- Three modes of operation – SPI, I²C, UART
Virtualized Driver Example

- Flash Storage

Diagram:
- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager

Connecting_lines:
- Functional
- Lock
- Power Control
Virtualized Driver Example

- Flash Storage
Virtualized Driver Example

- Flash Storage

```
+-------------------+
| Log User          |
+-------------------+
    |                 |
    v                 
+-------------------+
| Log Virtualizer   |
+-------------------+
    |                 |
    v                 
+-------------------+
| Flash Driver      |
+-------------------+
    |                 |
    v                 
+-------------------+
| SPI User          |
+-------------------+
    |                 |
    v                 
+-------------------+
| Arbiter           |
+-------------------+
    |                 |
    v                 
+-------------------+
| Immediate Power Manager |
+-------------------+
    |                 |
    v                 
+-------------------+
| Functional        |
+-------------------+
    | Lock            |
    v                 
+-------------------+
| Power Control     |
```

CPSL
Cyber-Physical Systems Laboratory
Virtualized Driver Example

- Flash Storage
Virtualized Driver Example

➢ Flash Storage

- Log User
- Log Virtualizer
- Flash Driver
- SPI User
- Arbiter
- Immediate Power Manager

Functional
Lock
Power Control
Flash Storage

Virtualized Driver Example
Virtualized Driver Example

Flash Storage

- Block User
  - Block Virtualizer
- Log User
  - Log Virtualizer
- Flash Driver
  - SPI User
  - Immediate Power Manager
  - Arbiter

Control Paths:
- Lock
- Power Control
Virtualized Driver Example

Flash Storage

- Block User
  - Block Virtualizer
- Log User
  - Log Virtualizer
- Flash Driver
  - SPI User
  - Arbiter
  - Immediate Power Manager

- Power Control
- Lock
Applications

- **Hand Tuned** – Most energy efficient
- **ICEM** – All concurrent operations
- **Serial +** – Optimal serial ordering
- **Serial -** – Worst case serial ordering

---

**Producer**

Every 5 minutes:
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Consumer**

Every 12 hours:
- For all new entries:
  - Send current sample
  - Read next sample

**Flash**

**Sensors**

**Radio**
Average energy consumption for application operations
Application energy consumption with 5 minute sampling interval and one send batch every 12 hours.

- **Hand Tuned**
- **ICEM**
- **Serial +**
- **Serial -**
Application energy with 5 minute sampling interval and one send batch every 12 hours
Application energy with 5 minute sampling interval and one send batch every 12 hours
Overhead of ICEM to Hand-Tuned Implementation
= ADC Timeout + Power Lock Overheads
With 288 samples per day
≈ 2.9 mAs/day
≈ 1049 mAs/year

Insignificant compared to total
5.60% of total sampling energy
0.03% of total application energy
Expected Node Lifetimes

![Graph showing lifetime vs. sampling interval]

- **Lifetime (years)**
  - 0.5
  - 1
  - 1.5
  - 2
  - 2.5
  - 3

- **Sampling Interval (min)**
  - 0.01
  - 0.1
  - 1
  - 10

- Key points:
  - 1 sec: Lifetime = 0.5 years
  - 10 sec: Lifetime = 1 year
  - 50 sec: Lifetime = 2 years
  - 5 min: Lifetime = 3 years
Expected Node Lifetimes

![Graph showing the expected node lifetimes over different sampling intervals. The x-axis represents the sampling interval in minutes (0.01 to 10 min), and the y-axis represents the lifetime in percentages (% of hand-tuned). The graph indicates that the lifetime remains relatively constant across different sampling intervals.]
Expected Node Lifetimes

![Graph showing expected node lifetimes. The x-axis represents the sampling interval (min), ranging from 0.01 to 10 minutes, and the y-axis represents the lifetime (%) of hand-tuned nodes. The graph includes three lines, each representing different node types: Icem (diamonds), Serial+ (triangles), and Serial- (crosses). The graph highlights lifetimes at 1 second, 10 seconds, 50 seconds, and 5 minutes.]
Evaluation Conclusions

- Conclusions about the OS
  - Small RAM/ROM overhead
  - Small computational overhead
  - Efficiently manages energy when given enough information

- Conclusions for the developer
  - Build drivers with short power down timeouts
  - Submit I/O requests in parallel
Integrated Concurrency and Energy Management

- Device driver architecture for low power devices
- At least 98.4% as energy efficient as hand-tuned implementation of representative application
- Simplifies application and driver development
- Questions the assumption that applications must be responsible for all energy management and cannot have a standardized OS with a simple API