MOS (MANTIS OS)

Goals

- General-purpose sw/hw platform for sensor networks
- Simplify sensor networks for novices
- Flexible for advanced research
- Adapt to resource constraints

Why multimodal?

- Supports variety of apps, hw platforms, deployment scenarios
Overview

- Lightweight POSIX-like multithreading
- Simple, vanilla ANSI C API
- Modest learning curve
- Cross platform portability
- Preemptive time-sliced scheduling
- I/O sync via mutual exclusion
- Layered network stack, device drivers
- Dynamic reprogramming
- Flexible power management
- Build complex apps
Lightweight MOS kernel

- Preemptive multi-threading
  - Priority/round-robin scheduling
  - Fast context switching (~60 microsecs)
- 144 byte footprint for scheduler
- Static thread table (default 12 threads)
  - 10 bytes per thread entry
- Dynamic thread stack allocated on heap
- Thread context saved on thread stack
- Counting and binary semaphores
User Level network stack

- **Non-strict layered design**
  - Network, transport, app layers in user level
  - MAC protocol in “comm” layer

- **Implement in one or more threads**
  - Performance Vs Flexibility trade off
  - Easy to implement/experiment
  - Can implement different routing protocols
  - Cross-platform prototyping
MOS Comm/ Device Layers

- **Comm Layer**
  - Interface to communication devices
  - Manage shared pool of buffers

- **Device Layer**
  - Interface to all devices
Comm layer

- Blocking `com_send` / `com_recv` calls
- Interrupt-driven packet reception
  - Fills `comBuf` from a pool of `comBufs`
  - `com_recv` gets pointer to a full `comBuf` pkt
- Zero-copy on both send and receive
  - Downside- App receiving pkt must free `comBuf`
- `com_mode` for power management (on/off/idle)
- `com_ioctl` for comm specific functions
- MAC protocol implemented in radio device driver
Device drivers

- Each driver implements four functions
  - `dev_read`, `dev_write`, `dev_mode`, `dev_ioctl`
- Read/write calls are blocking, synchronous
- Each driver keeps a mutex for I/O sync
- Single static call table of driver function pointers
  - Devices addressed using index in this table
  - `dev_register()` - to initialize call table, mutex
  - `dev_mode` for power management (on/off/idle)
  - `dev_ioctl` for device specific calls
Power Management

- Traditional idle/active modes
  - No CPU throttling, voltage variation
- Apps have varying duty cycles
  - Let the app tell when it wants to sleep
- `mos_enable_power_mgt()`
  - Power save mode turned off by default to be compatible with UNIX sleep!
- `mos_thread_sleep(PERIOD)`
- Timer wakes processor on earliest deadline expiry
Energe aware scheduler
static comBuf send_pkt; //comBuf goes in heap

void send_thread();

void start(void)
{
    mos_thread_new(send_thread, 128, PRIORITY_NORMAL);
}

void send_thread()
{
    send_pkt.size=2; //2 bytes

    while(1)
    {
        mos_led_toggle(0);
        (uint16_t)send_pkt.data[0] = dev_get(DEV_MICA2_TEMP);
        com_send(IFACE_RADIO, &send_pkt);
        mos_thread_sleep(1000);
    }
}
#include "led.h"
#include "com.h" //give us the communication layer
#include "msched.h"

void receiver();

void start(void)
{
    comBuf *recv_pkt; //give us a packet pointer

    com_mode(IFACE_RADIO, IF_LISTEN);

    while(1)
    {
        recv_pkt = com_recv(IFACE_RADIO); //blocking recv a packet
        com_send (IFACE_SERIAL, recv_pkt); //send packet out over serial
        com_free_buf(recv_pkt); //free the recv'd packet to the pool

        mos_led_toggle(0);
    }
}
Application Integration

Application, e.g.
- Visualization app
- Bridging Gateway

Socket API  Database API  X Windows GUI

MANTIS System API

POSIX Shim Layer

UNIX

X86 Hardware
Virtual X MOS Nodes

Virtual Sensor Network Of X MOS Nodes

Visualization Application

Bridging X MOS Gateway

USB/
Serial

[Diagram showing a laptop connected to several XMOS nodes, with labels for virtual sensor network, visualization application, and bridging X MOS gateway.]
Dynamic Reprogramming

- Reprogram entire deployed node
  - MOS boot loader can re-flash entire OS
  - Load stored code image from EEPROM

- Source-independent reprogramming
  - Standard API to store code image to EEPROM
  - Reprogramming possible over arbitrary connection (multi-hop), or from application
  - Simple, flexible network management
Remote Shell/Command Server

- Remote “login” to nodes
- Debugging functions
  - Peek/poke
  - Kernel status info
- Configuration
  - Spawn threads
  - Call functions

RemoteShellCommandServer.png
Summary

- Lightweight, preemptive multi-threaded, cross-platform OS
- Easily integrates sensor networks with other apps
- User friendly
  - Familiar API and language
  - Powerful management tools
Discussion (1)

**Classic Threads Vs Events argument**

**Threads**
- + Ease of application development, relatively better application debugging, good for complex computations
- - Stack management, concurrency issues, cost of context switching

**Events**
- + Single stack – easy to manage, no concurrency issues, no context switching, good for simple computations
- - Hard to program, very hard to debug
Discussion (2)

- Context switching
  - How critical is this?
    - Paper says 60 microsecs, website says 200 microsecs

- Complex tasks in sensornets?

- How far are we?
  - Paper mentions compression/crypto algos