Politehnica University of Bucharest

Automatic Control and Computers Faculty

Computer Science Department

Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors

Advanced Topics in Distributed Systems – 21 December 2010

Presenter

Laura Gheorghe

laura.gheorghe@cs.pub.ro

Authors

Adam Dunkels
Bjorn Gronvall
Thiemo Voigt

Contents

- System overview
- Processes
- The core and programs
- The kernel
- Events
- Loadable programs
- Services
- Libraries
- Over-the-air programming
- Contiki vs. TinyOS
- Conclusions

Contiki

- Dynamically download code at run-time
  - Ability to load and unload applications and services
- Portability
  - The only abstraction provided by the base system
    - CPU multiplexing and support for loadable programs
  - Other abstractions implemented as libraries and services
- Hybrid system
  - Event-driven kernel
  - Preemptive multithreading as an application library
  - Optionally linked with programs

System overview

- A running Contiki system:
  - Kernel
  - Libraries
  - Program loader
  - A set of processes

- A process may be:
  - Application
  - Service - implements functionality used by applications

- Any process can be dynamically replaced at run-time

- Communication between processes – through the kernel

Processes

- A process is defined by:
  - An event handler function
  - An optional poll handler function
- Process state:
  - In the process private memory
  - The kernel keeps a pointer to the process state
- All processes run in the same address space
- Interprocess communication: posting events

The core and programs

- A Contiki system:
  - The core
  - Loaded programs

- The core:
  - Contiki kernel
  - The program loader
  - Libraries

- The core is compiled into a single binary

- Programs are loaded into the system through the program loader

The kernel

- Lightweight event scheduler:
  - Dispatches events to running processes
  - Periodically calls the polling handlers of processes
- Program execution is triggered by:
  - Events dispatched by the kernel
  - Polling mechanisms
- An event handler is not preempted once scheduled

Events

- Two kinds of events:
  - Asynchronous events
    - Enqueued by the kernel and dispatched to the target process
  - Synchronous events
    - Immediately causes the target process to be scheduled
    - Control returns to the posting process only after the event has been processed
- Polling for events:
  - Used to check status updates from hardware devices
  - Programs can implement a poll handler
- A single shared stack for all process execution
- Events can be preempted only by interrupt handlers

Loadable programs

- Implemented using
  - Run-time relocation function
  - Relocation information found in the binary format
- Steps:
  - First allocates memory based on the information in the binary
  - Program loaded in the memory
  - Loader calls program initialization function

Services

- A process that implements a functionality used by other processes
- Dynamically linked – replaced at run-time
- Examples: communication protocol stacks, sensor device drivers
- Services are managed by the service layer
- Service = service interface + interface implementation in the process
- Applications use a stub library to communicate with the service
- The stub library uses the service layer to find the service
- The lookup returns a pointer to the service interface
- The interface stub calls the implementation of the requested function

Service replacement

- Services can be dynamically loaded and replaced

- Service is identified by the process ID
- The process ID must be retained when replacing the service

- The kernel informs the running service by posting a special event
- The service must remove itself from the system

- Sometimes the state of the service must be transferred to the new one
- Produces a service description and passes a pointer to the new service

Libraries

- Base system: basic CPU multiplexing and program loader
- The rest of the system: libraries

- Programs can be linked with libraries in three ways:
  - Statically with libraries that are part of the core
  - Statically with libraries that are part of a loadable program
  - Programs can call services implementing a specific library
  - s
- Libraries implemented as services can be replaced at run-time

Communication support

- Implemented as a service -> Replaceable at run-time
- Multiple communication stacks loaded simultaneously

- Device driver reads incoming packet into communication buffer
- Calls the upper layer communication service
- The communication stack processes the headers
- Posts a synchronous event to the application program

Preemptive multithreading

- Is implemented as a library on top of the event-based kernel
- This library can be optionally linked with programs

- Each thread requires a separate stack
- Threads execute on their own stack until they are preempted

Application - Over-the-air programming

- Developed a simple protocol for over-the-air programming

- Sends a single binary to some concentrator nodes
- The nodes receive and store the binary in EEPROM
- Broadcast the binary to their neighbors

- Object code size of application – 6 KB
- Complete system size – 30 KB

- Reprogramming a node – 30 seconds
- Reprogramming 40 nodes – 30 minutes

Contiki vs. TinyOS

- Architecture:
  - TinyOS is a monolithic operating system – single static image
  - Contiki uses a modular approach – dynamically loading programs

- Code size:
  - Larger than the one of TinyOS

- Memory footprint
  - TinyOS has a smaller memory footprint than Contiki
  - TinyOS performs compile-time optimization

- Scheduler:
  - TinyOS provides a FIFO event queue scheduler
  - Contiki provides FIFO event and poll handlers with priorities

Contiki vs. TinyOS – Execution model

- TinyOS – event-based execution model:
  - Concurrency model based on commands, asynchronous events and tasks
  - Commands and events can post tasks
  - Tasks are non preemptive and run to completion
  - Disadvantages: low programming flexibility, non-preemption
- Contiki – hybrid execution model:
  - Combines the advantages of events and threads
  - Multi-threading provided as an optional library
  - Synchronous events scheduled immediately, asynchronous scheduled later
  - Polling mechanism to avoid race conditions
  - Service implementation can be changed at run-time

Contiki vs. TinyOS - Reprogramming

- TinyOS – application level reprogramming
  - Write new image on the mote
  - High communication overhead
  - Entire image must be re-written for a small change

- Contiki – modular level reprogramming
  - Service implementation can be changed at run-time
  - Dynamically loading and unloading of services
  - The code must be loaded at the same location in the memory
  - Causes memory allocation problems if the code size increases

Contiki vs. TinyOS – Power Management

- TinyOS:
  - Provides API for conserving and managing power consumption
  - The processor should be put in sleep when
    - the radio is off
    - the clock interrupts are disabled
    - SPI interrupt is enabled
    - Task queue is empty
- Contiki:
  - Does not provide explicit power management abstractions
  - Must be implemented by programmers
  - The system should sleep when no events are scheduled

Conclusions

- Contiki OS:
  - Based on an event-driven kernel
  - Preemptive multi-threading as an application library
  - The system is divided into core and loaded programs
  - Shared functionality implemented as services
  - Feasible for a resource-constrained systems
    - The base system is lightweight and compact
    - Services and programs can be loaded and un-loaded at run-time