Overview of Middleware for Embedded Devices

Maria Porcius

AgroSense

Outline

Å Introduction

Å Embedded Operating Systems

Å Protocol Stacks

Å Virtual Machines

Å Conclusions

AgroSense

Jožef Stefan Institute
Department of
Communication Systems

Introduction

- *"The main purpose of middleware for sensor networks is to support the development, maintenance, deployment, and execution of sensing-based applications"*

- Middleware
  - Software infrastructure that glues together
    - Network hardware
    - Applications

Outline

Å Introduction

Å Embedded Operating Systems

Å Protocol Stacks

Å Virtual Machines

Å Conclusions

Embedded Operating Systems (EOSs)

- EOS - OS running on constrained devices, with limited resources and capabilities

- Kernel
  - Functions
    - Resource management: processor, memory, I/O devices
    - Allow other programs to run and use these resources
    - Synchronization
    - Communication between processes

Embedded Operating Systems

Traditional OS

- Large memory requirements
- Multithreaded architecture
- I/O model
- Kernel and user separation
- No energy constraints
- Ample available resources

Desired features for EOS

- Small memory footprint
- Efficient computation
- Power aware communication protocols
- Easy interface to expose data
- Support diverse application design
- Simple way to program, update and debug network applications

Embedded Operating Systems

- Support for programming wireless sensor nodes
  - Over-the-air reconfiguration
    - Maintenance
  - Code update mechanisms
    - Reprogramming
- Virtual machines

Embedded Operating Systems

- Modular vs. monolithic images
  - Monolithic (TinyOS, FreeRTOS, eCOS, uC/OS-II, Nut/OS)
    - One system image: system kernel + other components compiled together
    - Efficient execution environment (optimization at compilation)
    - High energy costs for updating
  - Modular (Contiki, SOS)
    - Static image (kernel) + loadable component images
    - Lower execution efficiency (no global optimization at compilation time)
    - Updates are less expensive (smaller size) - energy and time

Embedded Operating Systems

Å Scheduling

- ï Event driven model (Contiki, TinyOS, SOS)
  - Å Handlers that run to completion
  - Å No locking - only one event running at a time
  - Å One stack – reused for every event handler
  - Å Requires less memory
- ï Thread driven model (FreeRTOS, eCOS, Nut/OS, eCOS)
  - Å Each thread has its own stack
  - Å Thread stacks allocated at creation time (Unused stack space wastes memory)
  - Å Locking mechanisms - to prevent modifying shared resources

Embedded Operating Systems

Name	Scheduling	Mem. Mgmt.	Kernel	Image	Footprint
Contiki	Event/Thread Hybrid	Single	Yes	Modular	Variable
TinyOS	Event	Single stack	No	Monolithic	Variable
eCOS	Thread, preempt	Multiple stacks	Yes	Monolithic	Variable
uC/OS-II	Thread, preempt	Multiple stacks	Yes	Monolithic	Variable
FreeRTOS	Thread, preempt	Multiple stacks	Yes	Monolithic	Variable
Nut/OS	Thread, preempt	Multiple stack	Yes	Monolithic	Variable
SOS	Event	Single	Yes	Modular	Variable

Outline

- Introduction
- Embedded Operating Systems
- Protocol Stacks
- Virtual Machines
- Conclusions

Protocol Stacks

- Communication standards:
  - IEEE802.15.4, ZigBee, ISA100.11a, WirelessHART, 6LoWPAN, Bluetooth, IEEE802.11
- Design models
  - Adjacent layers
  - Cross layer design
  - Vertical calibration
- Implementation
  - In Software (within the OS)
  - On microcontroller
  - Hybrid

Protocol Stacks

Å Challenges

ï Small size of the physical nodes

ï Limited available memory

  Å Code optimizations

  Å Reduced functionalities

ï Interoperability

  Å Modularity

Protocol Stacks

- Testing
  - Check actual functionality compared to the expected one
  - Implementation size (memory usage + code size)
    - Memory usage
      - Max number of processes
      - Max size of event queue
      - Size of thread stacks (for multi-threaded operation)

Protocol Stacks

Communication Standard	Protocol Stack Implementation
IEEE 802.15.4	"Implementation of IEEE 802.15.4 protocol stack for Linux"
ZigBee	Z-Stack, Open-ZB, FreakZ, Microchip Stack
6LoWPAN	NanoStack2.0, Mantus, µIPv6, BLIP (Berkeley Low-power IP)
WirelessHART	"WirelessHART- Implementation and Evaluation on Wireless Sensors", "WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control"
ISA100.11a	NISA100.11a
Bluetooth	TinyBT, Axis OpenBT, BlueZ, Affix
IEEE 802.11	smxWiFi

Protocol Stacks

Name	Memory
NanoStack 2.0	32-64 kB ROM, 4-8kB RAM
Matus	4KB RAM
µIPv6	11KB ROM, 1.8KB RAM
BLIP	4KB RAM
Open-ZB	3kB RAM
Microchip Stack	22.3kB ROM
“Implementation of the IEEE802.15.4 protocol stack for Linux”	32KB RAM
smxWiFi	25K - 49K ROM
“WirelessHART-Implementation and Evaluation on Wireless Sensors”	11KB ROM, 10KB RAM
“WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control”	60kB ROM, 4kB RAM
TinyBT	3KB ROM, 1KB RAM
Axis OpenBT	190KB ROM

Protocol Stacks

Operating System	Protocol Stack Implementation
TinyOS	TinyBT, Open-ZB, BLIP, MATUS
Contiki	µIPv6, FreakZ, "WirelessHART: Implementation and Evaluation of Wireless Sensors"
RTOS	NanoStack (FREERTOS), smxWiFi stack (SMXRTOS)
Linux	BlueZ, Axis OpenBT, Affix, "Implementation of the IEEE 802.15.4", NISA100.11a
Windows	NISA100.11a, Z-Stack, Microchip Stack

Å Programming languages

- ï C
- ï nesC

Outline

Å Introduction

Å Embedded Operating Systems

Å Protocol Stacks

Å Virtual Machines

Å Conclusions

Virtual Machines

- Reason
  - A large number of platforms and OSs for WSNs
- Abstracting the device hardware from an application
  - Applications run on heterogeneous platforms
  - Underlying physical is kept transparent to the user

Virtual Machines

- Types
  - Middleware level VM
    - Between OS and application
    - Can add capabilities to the underlying OS (e.g. multithreading)
    - Mate, Darjeeling, VM*, SwissQM, DVM
  - System level VM
    - No OS support
    - Some OS specific functions (resource management, concurrency, thread scheduling, interruption handling)
    - More available memory
    - Squawk, .NET Micro

Virtual Machines

- Features
  - Portability
  - Platform independence
  - Programming support
    - Concurrency, modular and distributed software
    - Bug fixing , reprogramming, adding new tags and functionalities to a node

Virtual Machines

- Trade-off between the resources needed and the services it provides
- Reduce the distribution energy costs for software updates
  - VM code smaller than native machine code
  - Simpler reprogramming process
- Additional overhead
  - Increased time and memory requirements for execution
  - Increased energy spent in interpreting the code

Virtual Machines

Name	OS	ASVM	Execution Model	Platform	Memory requirements	Multi-threading	Supported programming language
Mate	TinyOS	No	Stack-based	Rene2, Mica	600B RAM 7.5KB ROM	Yes	TinyScript
.NET Micro	With (TinyOS)/ Without OS	No	Stack-based	Imote2	512MB ROM 300kB RAM	Yes	C#
Darjeeling	TinyOS, Contiki, FOS	No	Stack-based	Tnode, Tmote Sky,Fleck3/Fleck3B	2K RAM	Yes	Java subset
Squawk	With(Solaris, Windows, MACOSX,Linux) /Without OS	Yes	Stack-based	SunSpots Java Card 3.0	Core:80KB ROM Libs:270kB ROM	Yes	Java
VM*	OS*	yes	Stack-based	Mica	6kB ROM 200kB RAM	No	Java
SwissQM	TinyOS	Yes	Stack-based	Mica2,TmoteSky	33kB ROM 3kB RAM	Yes	Java subset

Conclusions

Å Middleware

ï State of the art

Å Embedded Operating Systems

Å Protocol Stacks

Å Virtual machines

maria.porcius@ijs.si

AgroSense

Jožef Stefan Institute
Department of
Communication Systems