International Journal of Recent Trends in Engineering, Vol. 1, No. 2, May 2009

Operating Systems for Tiny Networked Sensors: A Survey

A. K. Dwivedi, M. K. Tiwari, O. P. Vyas
School of Studies in Computer Science
Pandit Ravishankar Shukla University, Raipur (C.G.), INDIA - 492010
Email: {anuj.ku.dwivedi, manojtiwari.rsu, dropvyas}@gmail.com

***Abstract-* Wireless Sensor Networks (WSNs) is an emerging area of research. WSNs face lot of problems that do not arise in other types of wireless networks and computing environments. Limited computational resources, power constraints, low reliability and higher density of sensor nodes (motes) are some basic problems that need to be considered while designing or selecting a new operating system for performance evaluation of wireless sensor nodes (motes). In this paper we have studied various operating systems with performance analysis of WSNs.**

Index Terms- Wireless Sensor Networks, Tiny Networked Sensors, Operating Systems, Preemptive, Virtual Machine

I. Introduction

WSNs are composed of large numbers of tiny-networked devices that communicate untethered. Operating systems are at the heart of the sensor node architecture. In terms of Wireless Sensor Networks we need these things in operating system architectures: Extremely small footprint, extremely low system overhead and extremely low power consumption. When designing or selecting operating systems for tiny-networked sensors, our goal is to strip down memory size and system overhead because typical sensor devices [13, 14, 15] are equipped with 8-bit microcontrollers, code memory on the order of 100KB and RAM is less than 20KB. In literatures [1, 2] three classifications of O. S. architectures are described for tiny-networked sensors: Monolithic, Modular, and Virtual Machine. After evaluating various survey papers, research papers and poster presentation we have identified 39 operating systems running on different wireless sensor nodes (motes). In Section-3 we focus on 5 operating systems based on some criteria and in Section-4 features of rest 34 is described briefly.

II. Problem Formation

After analyzing several recently deployed WSNs and operating systems for tiny-networked sensors we identify that three OS features – virtual memory, preemptive priority scheduling, and OS safety – will significantly improve WSN systems. However, it is impossible to implement these features with traditional OS design techniques, due to the current situation of embedded H/W platforms. Moreover, we envision that the H/W constraints will still exist for a long time for energy and cost efficient miniaturized computing devices.

III. Features Of Widely Used Operating Systems

In this section we present 5 operating systems for tiny-networked sensors among 37. These operating systems were selected based on a number of criteria including usability, published results, and interesting characteristics and features.

TABLE I. Widely Used O.S. And Thier Supported Platforms

S. No.	Operating System	Sensor Node Name or Supported Platform
1.	TinyOS	BTnode, EyesIF X v1, EyesIF X v2, IMote, IMote 1.0, IMote 2.0, Iris, KMote, Mica, Mica2, MicaZ, Rene, SenseNode, TelosB, T-Mote Sky, Shimmer
2.	Contiki	T-Mote Sky, TelosB, avr MCU, MSP430 MCU, x86, 6502
3.	Mantis OS	Mica2, MicaZ, Nymph, TelosB
4.	SOS	XYZ, T-Mote Sky, KMote, Mica2, MicaZ, TelosB, avrora, Protosb, Cricket, Cyclops, emu
5.	Microsoft .NET Micro	IMote 2.0

3.1 TinyOS

An open-source operating system designed for wireless embedded sensor network. More specifically, it is designed to support the concurrency intensive operations required by networked sensors with minimal hardware requirements.

3.1.1 TinyOS Features

- No Kernel: Direct hardware manipulation.
- No Process Management: Only one process on the fly.
- No Virtual Memory: Single linear physical address space.
- No S/w Signal or Exception: Function call instead.
- No User Interface, power constrained.
- Unusually application specific H/w and S/w.
- Multiple flows, concurrency intensive bursts.
- Extremely passive vigilance (power saving).
- Tightly coupled with the application.
- Simulator: TOSSIM, PowerTOSSIM
- Written in “nesC” Language, a dialect of the ‘C’ language.

- TinyOS uses multi-hop routing instead of point-to-point connections to save transmission power. Route discovery is done by 2-hop broadcast and topology discovery is based on shortest path from each node to the base station.
- The paradigm for network transmissions in TinyOS is active messaging. Messages contain a handler address and on arrival this handler is called.
- *3.1.2 TinyOS Architecture*

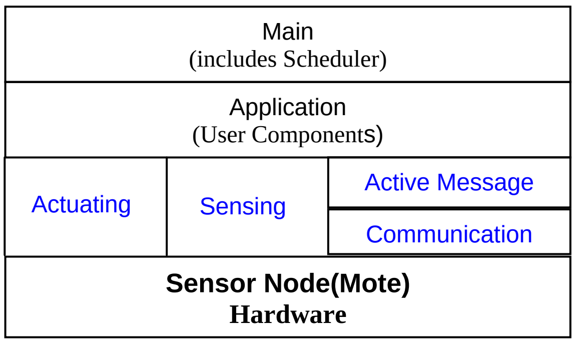
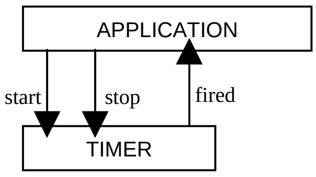


Fig. 1: Simplified TinyOS Architecture

- Component Based Architecture: enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks.
- Small footprint: fits in 178 Bytes of memory.
- Event based instead of threaded architecture.

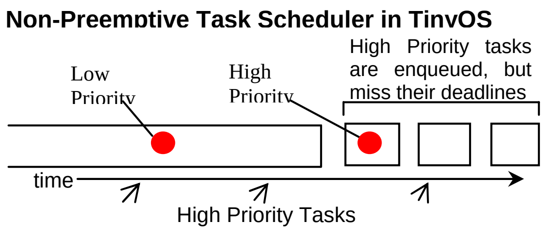


  - *start* and *stop* are commands.
  - *fired* is a event invoked by Timer.
  - The interface specifies: Timer must implement *start* and *stop*, Application must implement *fired.*

o Propagates events in time it takes to copy 1.25 Bytes.

o Switches context in the time to copy 6 Bytes of Memory.

- Radio and Clock have interrupts.
- Non-blocking/Non-preemptive Scheduling: Tasks will not preempts other tasks and run in FIFO order but only interrupts and associated events can preempt tasks. To avoid blocking scenarios, events and commands are expected to do only state transmissions and leave complex computations to tasks that can be preempted if necessary. This simple concurrency model is typically sufficient for I/O centric applications, but its difficulty with CPU-heavy applications has led to several proposals for incorporating threads into the OS.


Non-Preemptive Task Scheduler in TinyOS
Low Priority
High Priority
High Priority tasks are enqueued, but miss their deadlines
time
High Priority Tasks

- TinyOS has a Single Stack: TinyOS uses a simple queue with length 7 and a two level scheduling. Therefore, all I/O operations that last longer than a few hundred microseconds are asynchronous and have a callback. A TinyOS component can post a task, which the OS will schedule to run later.
- To support larger computations, TinyOS provides tasks, which are similar to a Deferred Procedure Call and interrupt handler bottom halves.

Latest version of TinyOS has numerous improvements such as: **Safe TinyOS**, a compile-time option that lets us incorporate run-time memory safety checks into oour application, **TOSThreads**, a threading library that runs on top of a standard TinyOS core etc.

3.2 Contiki

Contiki is a memory-efficient open source operating system for networked embedded devices. Contiki provide standard OS features like threads, timers, random number generator, clock, a file system and a command line shell. It includes an IPv4 stack (the original uIP stack), TCP and UDP support, as well as the Rime radio communication stack. It's latest version support IPv6 stack.

3.2.1 Contiki Features

- Event-driven Kernel: reduce the size of the system.
- Preemptive multi-threading support: an application library that runs on top of the event-driven kernel is optionally linked with applications that explicitly require a multithreaded model of computation.
- Simulator: COOJA
- Written in 'C' Language.

3.2.2 Contiki Architecture

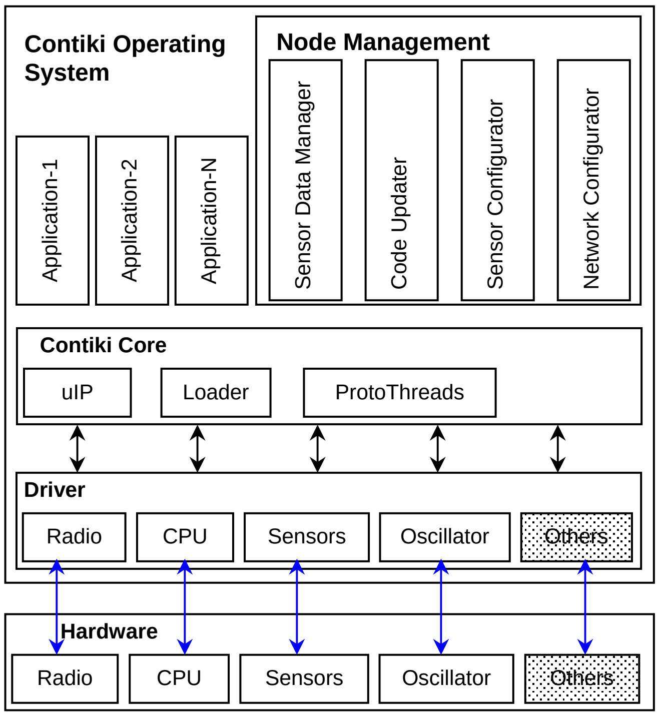


Fig. 2: Simplified Contiki kernel Architecture

- The Contiki kernel consists of a lightweight event scheduler that dispatches events to running processes and periodically calls processes polling handlers. All program execution is triggered either by events dispatched by the kernel or through the polling mechanism. The kernel does not preempt an event handler once it has been scheduled. The kernel supports two kinds of events: asynchronous and synchronous.
- It follows hybrid model, uses event-based model at kernel level and providing threading as application library.

3.3 MantisOS

An open source, light weighted and energy efficient multithreaded operating system for MultimodAl NeTworks of In-situ micro Sensor (MANTIS) nodes.

3.3.1 MantisOS Features

- Preemptive multi-threading support with standard I/O synchronization and a network protocol stack.
- To achieve energy efficiency, its power-efficient scheduler sleeps the microcontroller.
- It supports remote management of in-situ sensors via dynamic reprogramming and remote login.
- Provide cross-platform support across PC's, PDAs, as well as diverse micro sensor hardware platforms.
- MantisOS seamlessly integrates the virtual environment with the real deployment network.
- Written in 'C' Language.

3.3.2 MantisOS Architecture

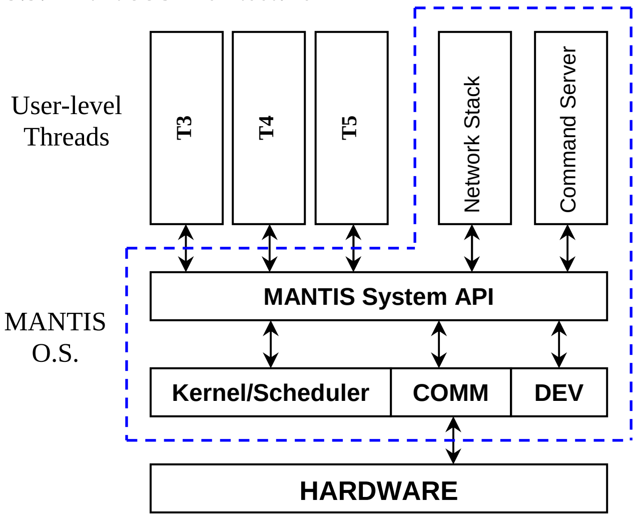


Fig. 3: Simplified MantisOS kernel Architecture

- Small footprint: fits in less than 500 Bytes of memory and 14 KB of flash including kernel, scheduler, and network stack.
- Kernel resembles classical, UNIX style schedulers. The kernel's main global data structure is a thread table, with one entry per thread. It uses subset of POSIX threads.
- Semaphores in it are 5-byte structures that are declared as needed by applications.
- Its communication COMM layer is designed for asynchronous I/O with the radio, serial, and achieves zero copy operation and zero polling.
- The device DEV layer is designed for synchronous I/O, e.g. sensor readings.

3.4 SOS (Sensor Operating System)

An OS whose primary goal is reconfigurability that is the ability to add, modify and delete the S/W modules at run-time.

3.4.1 SOS Features

- Static kernel interface for access system resources such as timers, memory, sensors, and actuators.
- Priority based asynchronous messaging for inter-node module communication.
- Type safe synchronous function call for intra-node module communication.
- Power management has not been considered explicitly, it provide special APIs for this.
- Inbuilt SOS simulation framework.
- Written in 'C' language.

3.4.2 SOS Architecture

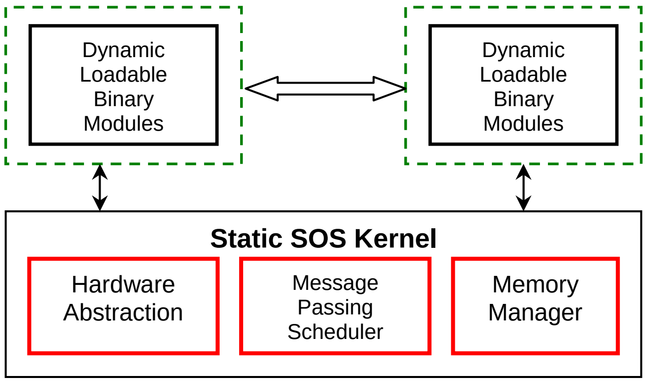

Dynamic Loadable Binary Modules
Dynamic Loadable Binary Modules
Static SOS Kernel
Hardware Abstraction
Message Passing Scheduler
Memory Manager

Fig. 4: Simplified SOS kernel Architecture

- Message Passing Scheduler for messages between parts of the system. Low layer H/W Abstractions for a given system platform. Fixed-partition dynamic memory mechanism to provide constant time dynamic memory allocation to the SOS kernel and modules.
- It uses modular approach at kernel as well as at application level. Application in SOS is built using set of interacting modules. These are similar to services in Contiki and tasks in TinyOS.
- Dynamic loading facility introduces significant problems related to memory allocation and message handling of modules.

3.5 Microsoft .NET Micro

The Microsoft .NET Micro is a bootable runtime module/framework that brings the advantages of .NET programming to devices too resource-constrained to run other Microsoft embedded platforms.

3.5.1 .NET Micro Framework Features

- The smallest .NET footprint yet (about 300 KB of RAM) able to run from ROM or flash memory.
- Full Visual Studio integration, including live debugging of code running on a tethered device

3.5.2 .NET Micro Framework Architecture

The architectural diagram shows the components that make up the .NET Micro Framework bootable runtime.

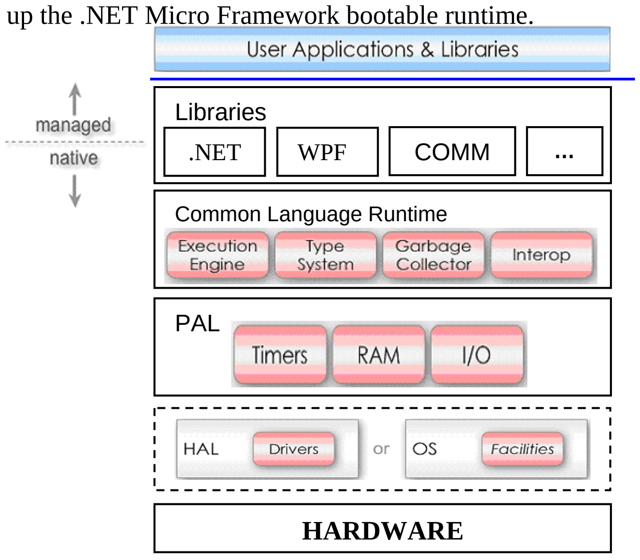


Fig. 5: Simplified .NET Micro Framework Architecture

- A version of the .NET class library tailored to embed applications, including GUI classes modeled on the Windows Presentation Foundation (WPF).
- A bootable Common Language Runtime (CLR) that can run directly on hardware without an operating system.
- Support for common hardware and interconnects (nonvolatile memory, GPIO, I2C, RS232, SPI).
- Managed device driver model for devices connected via the supported interconnects.
- Seamless persistent storage through extended weak references.
- Since the .NET Micro Framework separates all code that touches the hardware into a HAL. Changing any aspect of the underlying hardware (such as the LCD part being used) will typically require significant changes to the HAL, and rarely some changes to the PAL. Changes to components above these layers are generally not necessary.

TABLE II. .NET MICRO FRAMEWORK AT A GLANCE

Device Features	Connected, Small, Wearable, Graphical User Interface
Footprint	250-500KB managed code Full featured
Power	Very low power
CPU	ARM7, ARM9, No MMU
Real-time	Not Real-time
Managed vs. Native Code	Managed via .NET Micro Framework, native code through interoperability only.

IV. OTHER OPERATING SYSTEMS

4.1 YATOS (Yet Another Tiny OS): It is the first operating system for WSN with priority. It is event-driven, modular and easy-to-use for C programmers.

4.2 BTnutOS or NutOS: An open source specifically designed for BTnut nodes provide timer and dynamic memory allocation facilities.

4.3 PeerOS: An extension of EYES OS with real-time scheduling, memory management, and resource management. Offers priority based multitasking.

4.4 Embedded Linux: Embedded versions of Linux are designed for devices with relatively limited resources. The advantages of embedded Linux over other embedded operating systems include no royalties or licensing fees, a stable kernel, a support base and the ability to modify and redistribute the source code. The disadvantages include a comparatively larger memory footprint (kernel and root filesystem), complexities of user mode and kernel mode memory access and complex device drivers framework.

4.5 NanoRK: It is a fully preemptive reservation-based real-time operating system (RTOS) with multi-hop networking support. It includes a light-weight embedded resource kernel (RK) with rich functionality and timing support using less than 2KB of RAM and 18KB of ROM. Nano-RK supports fixed-priority preemptive multitasking for ensuring that task deadlines are met, along with support for CPU, network, as well as, sensor and actuator reservations.

4.6 µCOS: It is a licensed commercial embedded real-time OS, support preemptive multitasking in a lightweight RAM footprint of less than 4 KB.

4.7 Squawk VM: It was developed to run on the SunSPOTs, written in Java and does not need an OS but comes with OS functions of its own. Its memory demand is 149 KB for the VM, 363 KB for the VM suite, 158 KB for the library suite in RAM.

4.8 SensorOS: It supports preemptive priority-based scheduling, very fine-granularity timing, and message passing inter-process communication. It has been implemented for resource constrained Tampere University of Technology WSN (TUTWSN) nodes. In TUTWSN node platform with 2 MIPS PIC micro-controller unit, SensorOS kernel uses 6964 Byte code and 115 Byte data memory.

4.9 MagnetOS: It is a distributed operating system for ad-hoc and sensor networks whose goal is to enable power-aware, adaptive, and easy-to-develop ad-hoc networking applications. These properties are provided through a single system image of a unified virtual machine to applications over an ad-hoc collection of heterogeneous nodes. It automatically and transparently partitions applications into components and dynamically finds a placement of these components on nodes within the ad-hoc network to reduce energy consumption and increase system longevity.

4.10 CORMOS: It is a communication-oriented event based runtime system for sensor networks. It is tailored to provide easy-to-use abstractions and treat communication as a first class citizen rather than an extension deal with sensor limitations on concurrency, memory, and power.

4.11 Bertha: It uses modular approach.

4.12 kOS: An object based non real-time OS uses modular approach.

4.13 VMSTAR: A virtual machine with dynamic reprogramming capability.

4.14 Maté: A virtual machine architecture built on top of TinyOS to provide run-time reprogramming. The code fits in terms of capsules (24 instructions).

4.15 CVM: ContikiVM built on top of Contiki follows virtual machine architecture.

4.16 EYES: An event based non real-time OS for embedded devices. Efficient utilization of distributed resources in the network are handled by resource management and remote procedure call mechanisms.

4.17 SenOS: It is a finite state machine based OS. Its main components are event queue, callback library, and a state transition table. It provides power management as an application layer protocol for sensor network management. It is based on state based execution model with dynamic reprogramming facility.

4.18 DCOS: A real time lightweight Data Centric Operating System for embedded devices. The main features of this OS are real-time scheduling, dynamic configurations and the data-centric approach. Also supports dynamic allocation of memory to the modules and for various H/W devices.

4.19 t-Kernel: It employs a lightweight translation engine as a key part of the OS kernel, efficient load-time binary modification on a resource-constrained embedded processor with only 4KB RAM and 128K of Flash, and provides efficient virtual memory with neither hardware address translation support nor repeatedly writable external nonvolatile storage, and it guarantees OS protection for sensor network platforms without memory protection hardware.

4.20 Nano-QPlus: A real-time thread based OS for embedded devices. Provide priority based scheduling.

4.21 SmartOS: A small modular adept real-time OS for embedded systems.

4.22 AVRX: It is an open source embedded real-time operating system, support preemptive multitasking in a lightweight RAM footprint of less than 4 KB.

4.23 Pixie: The Pixie is based on dataflow programming model and is based on the concept of resource tickets, a core abstraction for representing resource availability and reservation. This OS enables resource aware programming model in which applications receive feedback, have explicit control over resource usage.

4.24 LiteOS: Unix Like OS for WSNs, built on Hierarchical File System and providing a wireless shell to interact using Unix like commands.

These Operating Systems are in progress **T2**, **OSSTAR** and **Jallad**. T2 and OSSTAR are non-real-time OS. T2 is component based whereas OSSTAR having hybrid execution model. Jallad OS is application specific and is exclusively for space applications. **CustomOS**, **GenOS** and **MoteWorks** are also identified currently working on GWnode, SenseNode, and Iris wireless sensors respectively. We also identified **NanoVM**, **ParticleVM**, **KVM** and **AmbiCompVM** these falls in virtual machine architecture category.

V. Results And Disscusion

As we mentioned earlier that virtual memory, preemptive priority scheduling and OS protection as well as interoperability with IPv6 would significantly improve WSN systems but currently used operating systems are not able to support all above described features this is due to limitation of computational resources, power constraints features. We hope this requirement is tackled in future.

VI. Conclusion

The objectives of this paper were to provide overview of various operating systems for Tiny Networked Sensors or WSNs and to present the significant features of each one. The objective of such survey was two-fold. First, knowing the strengths and weaknesses of a number of different operating systems is valuable because it allows wireless sensor network administrators to select the one most appropriate for their application. Second, the developers of new operating systems are well served knowing what has worked in previous operating systems and what has not. To these ends, we believe we have succeeded.

References

[1] Antônio Augusto Fröhlich and Lucas Francisco Wanner: “Operating System Support for Wireless Sensor Networks”. Journal of Computer Science 4 (4): 272-281, 2008. ISSN 1549-3636
[2] Adi Mallikarjuna Reddy, V AVU Phani Kumar, D Janakiram and G Ashok Kumar: “Operating Systems for Wireless Sensor Networks: A Survey-Technical Report”, IIT Madras, Chennai, India, May 3, 2007.
[3] Easwaran: “TinyOS”. http://www.i2r.a-star.edu.sg/icsd/SecureSensor/papers/TinyOS.pdf
[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pis-ter: “A System Architecture for Networked Sensors”, U.C. Berkeley. http://web.cs.berkeley.edu/tos
[5] V. Raghunathan: “TinyOS”. http://black.csl.uiuc.edu/vivek/talks/tinyostalk.pdf
[6] John Yannakopoulos and Angelos Bilas: “CORMOS: A communication-oriented runtime system for sensor networks”. In Proc. of the Second European Workshop on Wireless Sensor Networks (EWSN 2005), February 2005.
[7] http://www.microsoft.com/netmf/default.mspx
[8] http://linuxdevices.com/news/NS6488003219.html
[9] http://redwood.snu.ac.kr/

[10] http://senses.ucdavis.edu/
[11] EYES. http://www.eyes.eu.org/
[12] http://www.btnode.ethz.ch/
[13] http://www.cse.unsw.edu.au/~sensar/hardware/hardware_survey.html
[14] http://senses.cs.ucdavis.edu/resources.html
[15] http://www.snm.ethz.ch/Main/Homepage
[16] Baunach, Marcel Kolla, Mühlberger, Clemens: "Introduction to a Small Modular Adept Real-Time Operating System". Dept. of Comp. Engg., Am Hubland, University of Würzburg, Bavaria, Germany.
[17] Bjoern Saballus et al.: "Towards a Distributed Java VM in Sensor Networks using Scalable Source Routing". University of Karlsruhe, Karlsruhe.
[18] www.eecs.harvard.edu/~konrad/jobapp/research-lorincz.pdf
[19] http://www.sics.se/contiki/
[20] http://wireless.cs.uh.edu/presentation/2007/LiteOS.ppt