OPERATING SYSTEMS FOR WIRELESS SENSOR NETWORKS

by

Mtech

February 24, 2016

EXAMPLES OF OPERATING SYSTEMS

1. TinyOS
   - The design of TinyOS allows application software to access hardware directly when required
   - TinyOS is a tiny microthreaded OS that attempts to address two issues
     - how to guarantee concurrent data flows among hardware devices
     - how to provide modularized components with little processing and storage overhead.
   - TinyOS is required to manage hardware capabilities and resources effectively while supporting concurrent operation in an efficient manner
   - TinyOS uses an event-based model to support high levels of concurrent application in a very small amount of memory

Advantages

1. It requires very little code and a small amount of data
2. Events are propagated quickly and the rate of posting a task and switching the corresponding context is very high
3. It enjoys efficient modularity

Mate

1. Mate is designed to work on the top of TinyOS as one of its components
2. It is a byte-code interpreter that aims to make TinyOS accessible to nonexpert programmers and to enable quick and efficient programming of an entire sensor network
3. In Mate, a program code is made up of capsules. Each capsule has 24 instructions, and the length of each instruction is 1 byte
4. The capsules contain type and version information, which makes code injection easy.

MagnetOS

1. It is a distributed adaptive operating system designed specifically for application adaptation and energy conservation
2. Other operation systems do not provide a network-wide adaptation mechanism or policies for application to effectively utilize the underlying node resources
3. The burden of creating adaptation mechanisms (if any) is on the application itself. This approach is usually not energy efficient

1. The goals of MagnetOS
   - To adapt to the underlying resource and its changes in a stable manner
   - To be efficient with respect to energy conservation
   - To provide general abstraction for the applications
   - To be scalable for large networks

MANTIS

1. MANTIS is a multithread embedded operating system, which with its general single-board hardware enables flexible and fast deployment of applications
2. With the key goal of ease for programmers, MANTIS uses classical layered multithreaded structure and standard programming language
3. The layered structure contains multithreading, preemptive scheduling with time slicing, I/O synchronization via mutual exclusion, a network protocol stack, and device drivers
4. The current MANTIS kernel realizes these functions in less than 500 bytes of RAM. MANTIS uses standard C to implement the kernel and API

OSPM

1. OSPM (or dynamic power management, DPM), proposed in , is directed at power management techniques
2. The general dynamic power management is based on a greedy algorithm that will switch the system to a sleep state as soon as it is idle
3. It considers the following factors
   - Transitioning to a sleep state has the overhead of storing the processor state and shutting off the power supply
   - Waking up takes a finite amount of time
   - The deeper the sleep state, the less the power consumption will be lower and the wake-up time will take longer

EYES OS

1. EYES OS [10.7,10.8] uses an eventdriven model and task mechanism to realize these objectives
2. It works in a simple sequence as follows
   - Perform a computation
   - Return a value
   - Enter the sleep mode.
3. The task can be scheduled using a FIFO-, priority-, or deadline-based approach (such as EDF), and is triggered by events in a nonblocking manner
4. EYES OS defines an application programming interface (API) locally and for the network components

SenOS

1. It is a finite state machine (FSM)–based operating system
2. It has three components
   - A kernel that contains a state sequencer and an event queue. The state sequencer waits for an input from the event queue (a FIFO queue)
   - A state transition table that keeps the information on state transition and the corresponding callback functions. Each state transition table defines an application. Using multiple state transition tables and switching among them, SenOS supports multiple applications in a concurrent manner
   - A callback library of call functions. An incoming event will be queued in the event queue. The first event in the event queue is scheduled, which triggers a state transition and correspondingly, invokes the associated functions

EMERALDS

1. EMERALDS is an extensible microkernel written in Cþþ for embedded, real-time distributed systems with embedded applications running on slow processors (15 to 25 MHz) and with limited memory (32 to 128 kB)
2. It supports multithreaded processes and full memory protection, which are scheduled using combined earliest deadline first (EDF) and a rate-monotonic (RM) scheduler

PicOS

1. One property of OS microcontrollers with limited RAM is to try to allocate as little memory as possible to a process or thread.
2. PicOS is written in C for a microcontroller with limited on-chip RAM (e.g., 4 kB).
3. In PicOS, all tasks share the same global stack and act as coroutines with multiple entry points and implicit control transfer, which is different from classical multitasking approaches

CONCLUSION

1. In this chapter we discussed operating systems for wireless sensor networks and presented design guidelines and objectives for a WSN operating system
2. A survey of some existing operating systems is also provided
3. The major issues for the design of operation systems for WSNs are size (memory requirement), energy-efficient IPCs and task scheduling, effective code distribution and upgrades, and finally, generic application programming interfaces