International Journal of Multidisciplinary Research and Development

International Journal of Multidisciplinary Research and Development

Volume: 2, Issue: 6, 167-170
June 2015
www.allsubjectjournal.com
e-ISSN: 2349-4182
p-ISSN: 2349-5979
Impact Factor: 3.762

Nilesh Gujar
Department of Electronics & Telecommunication, RMDSSOE, Savitribai Phule Pune University, Warje, Pune, 411058, India.

Snehal Bhosale
Department of Electronics & Telecommunication, RMDSSOE, Savitribai Phule Pune University, Warje, Pune, 411058, India

Correspondence:
Nilesh Gujar
Department of Electronics & Telecommunication, RMDSSOE, Savitribai Phule Pune University, Warje, Pune, 411058, India.

RTOS in Wireless sensor network

Nilesh Gujar, Snehal Bhosale

Abstract
Wireless Sensor Networks (WSNs) are a valuable technology to support countless applications in different areas. Given the WSN nodes resource constrained characteristics, designing energy-aware applications, communication protocols and security mechanisms are critical. The operating system (OS) running on the WSN node also interferes with the node overall behavior, and its energy consumption. In view of a variety of WSN applications, there is a need of developing a self-adaptable and self-configurable embedded real-time operating system (RTOS). In recent years, the availability of cheap and small micro sensor node and low power wireless communication give a contribution of enhanced developments of wireless sensor network application in real society. A light-weight embedded resource kernel with rich functionality and timing support is practical and.

Keywords: Wireless sensor network, Sensor node, Real Time Operating system.

1. Introduction
A Wireless Sensor Network (WSN) is composed of a large number of small sensor nodes having limited computation capacity, restricted memory space, limited power resource, and short-range radio communication device. It has a base-station or sink, which does the functions of calculation and decision-making, and can be compared with the functionalities of server or in some cases as a gateway in a computer network. The nodes communicate wirelessly and often self-organize after being deployed in an ad-hoc fashion. In several WSN-based applications, the nodes are left unattended for their whole operational lifetime after deployment. Moreover, the sensing and processing tasks the node will execute, as well as the overhead introduced by the node's operating system (OS), must be accounted for in the energy consumption. Hence, understanding the impact of processing and communications tasks on the nodes' energy usage provides the necessary knowledge to choose the appropriate duty cycle. This approach allows the sensor network node to alternate between active, idle and low-power periods, thus saving energy.
In a sensor network, application-specific requirements drive the entire hardware design, from processing capabilities to radio bandwidth and sensor modules, thus requiring the hardware to be modular. However, these requirements have led to a huge variety of hardware components, making wireless sensor networks hardware not only modular, but also heterogeneous. In this scenario, a sensor application developed for a given platform will seldom be portable to a different one, unless the run-time support systems on those platforms deliver mechanisms that abstract and encapsulate the sensor platform in an adequate manner. At the same time, the limited resources typically found in sensor networks hardware require any runtime support for these systems to be efficient and not to use excessive resources. The need for connectivity, hardware abstraction and management of limited resources makes operating system support imperative for sensor network applications. Considering current research, technology and applications, we may list a series of operating system requirements for wireless sensor networks. Such a system should:

A. Provide basic operating system functionality
The functionality and portability of sensor networks applications, an operating system for such devices should provide traditional operating system services such as: hardware abstraction, process management, timing services and memory management.

B. Provide efficient power management mechanisms
Efficient power management in the sensor nodes is a determining factor for the network's life time. A runtime support system for sensor networks applications should provide power management mechanisms to the applications, as well as use as little power as possible to provide its services.

C. Provide field reprogramming mechanisms

An operating system for sensor networks should ideally provide total or partial field reprogramming mechanisms for deployed applications.

D. Abstract heterogeneous sensing hardware in a uniform fashion

The application-specific requirements of sensor networks make its hardware not only modular, but also heterogeneous. In this scenario, a sensor application developed for a given platform will seldom be portable to a different one, unless the runtime support systems on those platforms deliver mechanisms that abstract and encapsulate the sensor platform in an adequate manner.

E. Provide a configurable communication stack

Given the specific communication requirements of different applications, communication hardware for sensor networks should be widely configurable. The operating system should provide means to configure the communication protocol stack, starting from the medium access control protocols.

2. Real Time Operating System

In general, an operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that run on the computer. An RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability. This can be especially important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard. To be considered "real-time", an operating system must have a known maximum time for each of the critical operations that it performs. Some of these operations include OS calls and interrupt handling. Operating systems that can absolutely guarantee a maximum time for these operations are commonly referred to as "hard real-time", while operating systems that can only guarantee a maximum most of the time are referred to as "soft real-time". Real-time and embedded systems operate in constrained environments in which computer memory and processing power are limited. They often need to provide their services within strict time deadlines to their users and to the surrounding world. It is these memory, speed and timing constraints that dictate the use of real-time operating systems in embedded software.

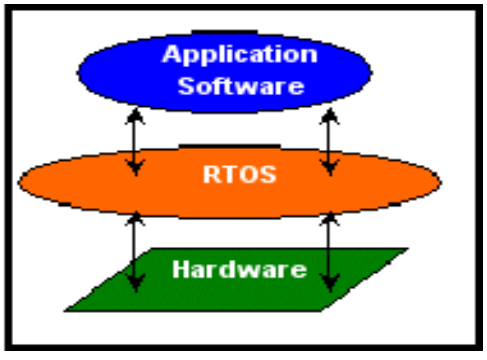

Fig 1: Firmware Architecture

The "kernel" of a real-time operating system ("RTOS") provides an "abstraction layer" that hides from application software the hardware details of the processor (or set of processors) upon which the application software will run. This is shown in Figure 1.In providing this "abstraction layer" the RTOS kernel supplies five main categories of basic services to application software, as seen in Figure 2.

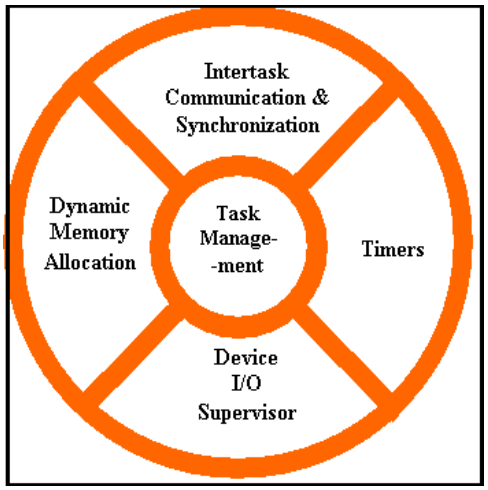

Fig 2: Services provided by RTOS

A. Task Management

The most basic category of kernel services, at the very center of Figure 2, is Task Management. This set of services allows application software developers to design their software as a number of separate "Task" of software, each handling a distinct topic, a distinct goal, and perhaps its own real-time deadline.

B. Inter-task Communication and Synchronization

These services make it possible for tasks to pass information from one to another, without danger of that information ever being damaged. They also make it possible for tasks to coordinate, so that they can productively cooperate with one another. Without the help of these RTOS services, tasks might well communicate corrupted information or otherwise interfere with each other.

C. Timers

Embedded systems have stringent timing requirements; most RTOS kernels also provide some basic Timer services, such as task delays and time-outs.

D. Dynamic Memory Allocation

Many RTOS kernels provide Dynamic Memory Allocation services. This category of services allows tasks to "borrow" chunks of RAM memory for temporary use in application software. Often these chunks of memory are then passed from task to task, as a means of quickly communicating large amounts of data between tasks. Some very small RTOS kernels that are intended for tightly memory-limited environments, do not offer Dynamic Memory Allocation services.

E. Device I/O

These services provide a uniform framework for organizing and accessing the hardware device drivers that are typical of an embedded system.

In addition to kernel services, many RTOSs offer a number of optional add-on operating system components for such high-level services as file system organization, network communication, network management, database management, user-interface graphics, etc.

3. RTOS for Sensor Node

A WSN is generally composed of a centralized station and tens, hundreds, or perhaps thousands of tiny sensor nodes. With the integration of information sensing, computation, and

wireless communication, these devices can sense the physical phenomenon, process the raw information, and share the processed information with their neighboring nodes. Typical sensor nodes are equipped with a sensor, a microprocessor or microcontroller, a memory, a radio transceiver, and a battery. Therefore, these hardware components should be organized in a way that makes them work correctly and effectively without a conflict in support of the specific applications for which they are designed. Each sensor node needs an OS that can control the hardware, provide hardware abstraction to application software, and fill in the gap between applications and the underlying hardware.

Traditional OSes are not suitable for WSNs because WSNs have constrained resources and diverse data centric applications, in addition to a variable topology. WSNs need a new type of operating system, considering their special characteristics. There are several issues to consider when designing sensor network OS.

A. Process Management and Scheduling

The traditional OS provides process protection by allocating a separate memory space (stack) for each process. Each process maintains data and information in its own space. But this approach usually causes multiple data copying and context switching between processes. This is obviously not energy efficient for WSNs. Sensor network OSes should provide efficient resource management mechanisms in order to allocate microprocessor time and limited memory.

B. Memory Management

Memory is often allocated exclusively for each process/task in traditional OSes, which is helpful for protection and security of the tasks. Since sensor nodes have small memory, another approach, sharing, can reduce memory requirements.

C. Kernel Model

The event-driven and finite state machine (FSM) models have been used to design microkernels for WSNs. The event-driven model may serve WSNs well because they look like event-driven systems. An event may comprise receiving a packet, transmitting a packet, detection of an event of interest, alarms about energy depletion of a sensor node, and so on. The FSM-based model is convenient to realize concurrency, reactivity, and synchronization.

D. Energy Efficiency

Sensor nodes provide very limited battery lifetime. On the other hand, guaranteeing sensor networks to operate for 3 to 5 years is a very desirable objective. Sensor network OS should support power management, which helps to extend the system lifetime and improve its performance..

E. Application Program Interface

Sensor nodes need to provide modular and general APIs for their applications. The APIs should enable applications access the underlying hardware. This may allow access and control of hardware directly, to optimize system performance.

F. Code Upgrading and Reprogramming

Since the behavior of sensor nodes and their algorithms may need to be adjusted either for their functionality or for energy conservation, the operating system should be able to reprogram and upgrade.

4. Existing Operating Systems for Wireless Sensor Networks

Over the years, we have seen various OSes emerging in the sensor network community.

A. TinyOS

TinyOS, developed in UC Berkeley, is perhaps the earliest sensor network OS in the literature. The design of TinyOS allows application software to access hardware directly when required.

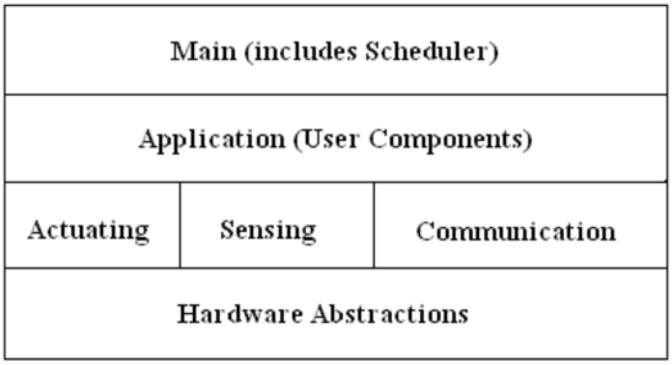

Fig 3: Sensor Node block diagram

TinyOS is a tiny micro threaded OS that attempts to address two issues: how to guarantee concurrent data flows among hardware devices, and how to provide modularized components with little processing and storage overhead. These issues are important since 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. Figure 3 illustrates the basic architecture of TinyOS. TinyOS includes a tiny scheduler and a set of components. C language was designed for programming based on TinyOS. TinyOS has a component-based programming model, codified by the nesC language. This WNS operating system defines three types of components: hardware abstractions, synthetic hardware, and high-level software components. Hardware abstraction components are the lowest-level components. They are actually the mapping of physical hardware such as Input/output (I/O) devices, a radio transceiver, and sensors.

B. Contiki Operating System

The Contiki operating system is an open source operating system for networked embedded systems in general, and wireless sensor nodes in particular. It is developed by a team of developers from the industry and academia. The Contiki project is led by Adam Dunkels. Typically, a running Contiki system consists of the kernel, libraries, the program loader, and a set of processes. Communication between processes always goes through the kernel, which does not provide a hardware abstraction layer, but let's device drivers and applications communicate directly with the hardware. A process is defined by an event handler function and an optional poll handler function. The process state is held in the process' private memory and the kernel only keeps a pointer to the process state. Inter-process communication is done by posting events. Looking at it from a higher perspective, the Contiki system is partitioned into two parts: the core and the loaded programs as shown in Fig. 4. Typically, the core consists of the Contiki kernel, the program loader, the most commonly used parts of the language runtime and support

libraries, and a communication stack with device drivers for the communication hardware. Programs are loaded into the system by the program loader. The program loader is in charge of loading/unloading the programs into the system either by using the communication stack or directly attached storage.

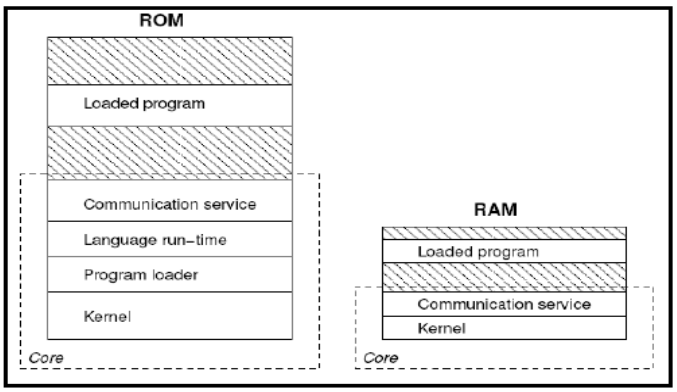

Fig 4: Contiki System

Both the Contiki system and applications for the system are implemented in the C programming language. Contiki is implemented in C, it is highly portable.

C. LiteOS

LiteOS, developed in the University of Illinois at Urbana Champaign, is designed to provide a traditional Unix-like environment for programming WSN applications. It includes: a hierarchical file system and a wireless shell interface for user interaction using UNIX-like commands; kernel support for dynamic loading and native execution of multithreaded applications

.

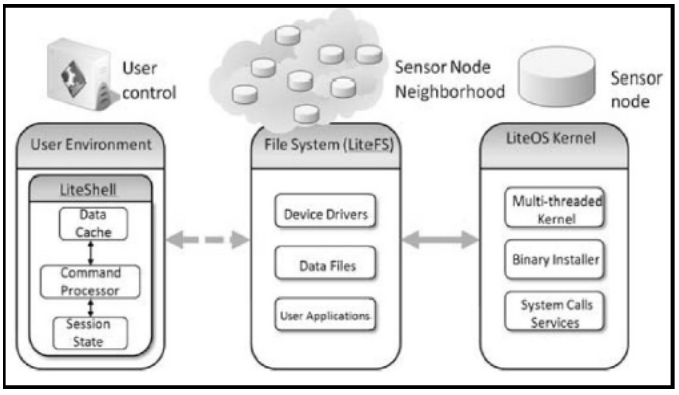

Fig 4: LiteOS Architecture

Above figure shows the overall architecture of the LiteOS operating system, partitioned into three subsystems: LiteShell, LiteFS, and the Kernel. Implemented on the base station PC side, the LiteShell subsystem interacts with sensor nodes (motes) only when a user is present. The interfaces of LiteFS provide support for both file and directory operations.

5. Conclusion

In this paper presented OS for WSNs and several major issues for the design of sensor network OS. By examining some existing sensor network OSes, we know the strengths and weaknesses of a number of different OSes. The contributions of this paper are twofold. First, we identify several major issues for the design of sensor network OS, such as memory requirement, process management and scheduling, kernel model, generic application programming interfaces, effective code distribution and upgrades, energy-efficient, real-time guarantee, and reliability. Second, our work may allow research community to know the features of a number of different OSes. This work is valuable with both OS developers and OS users. With OS developers, they will know what has worked in previous OSes. With OS users, they know the features of existing sensor network OSes, select a sensor network OS that is the most appropriate for their application.

6. References

1. Antônio Augusto Fröhlich and Lucas Francisco Wanner, Federal University of Santa Catarina, Brazil, “Operating System Support for Wireless Sensor Networks”, and Journal of Computer Science 4 (4): 272-281, 2008 ISSN 1549-3636 © 2008 Science Publications.
2. Thang Vu Chien, Hung Nguyen Chan, and Thanh Nguyen Huu, Thai Nguyen University of Information and Communication Technology, Vietnam, “A Comparative Study on Operating System for Wireless Sensor Networks”, Presented in Journal ICACSIS 2011 ISBN: 978-979-1421-11-9.
3. Lalit Saraswat, Pankaj Singh Yadav, Raj Kumar Goel Institute of Technology, Ghaziabad, India “A Comparative Analysis of Wireless Sensor Network Operating Systems”, Int J Engg Techsci Vol 1(1) 2010,41-47.
4. Nicolas Melot, “Study of Operating systems for embedded devices”.