Computers and Electrical Engineering 000 (2016) 1–10

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

A timer-based operating system for ZigBee sensor platforms ☆

Chia-Chi Chang*, Chuan-Bi Lin

Department of Information and Communication Engineering, Chaoyang University of Technology, Taichung 41349, Taiwan, R.O.C.

ARTICLE INFO

Article history:
Received 24 June 2015
Revised 24 November 2015
Accepted 26 November 2015
Available online xxx

Keywords:
Operating system
Event-driven
Timer-driven
Energy consumption

ABSTRACT

In recent years, resource constrained hardware makes the operating system simplicity a most crucial design criterion. The majority of research in such operating systems has focused on the reactive nature of an event-driven kernel. However, the event-based kernel may result in the latency of processing events and erroneous energy profiling. Most battery-driven wireless sensor nodes use the radio event to wake up sleeping nodes, but the radio always consumes more energy than other components during the transmission intervals. The contribution of this paper is that we present *EXOS*, a timer-driven operating system which can both process periodic events on time and obtain rapid responses to external signals. The system kernel can be easily ported to any other memory-constrained target platforms. EXOS is able to provide the detailed prediction of each component's energy consumption during program execution. The evaluation has demonstrated that EXOS can be practically integrated into wireless sensor networks.

1. Introduction

The continuing improvements in IC fabrication processes and Micro-Electro-Mechanical Systems technologies have led to many novel and fascinating applications in wireless sensor networks [1,2,3,4,5]. As system integration requirements increase, so does demand for small, low-cost, ultra-low power, and greater noise immunity wireless platforms. The construct of wireless sensor networks may be comprised of several thousands of the nodes. The nodes can ambitiously interface with unfamiliar surroundings through MEMs-based sensors and then forward packets containing necessary information to the neighboring nodes. The wireless sensor node commonly consists of the following components: an 8 or 16 bit microcontroller only has 4 K to 256 K bytes of on-chip program memory and 128 bytes to 10 K bytes of on-chip data memory, a low-power RF transceiver, a great diversity of sensors, and a limited amount of power supply. From the hardware design point of view, to cram everything into the smallest space as close as possible makes laying out a Printed-Circuit Board is a crucial design. The wireless sensor platforms always combine digital sections with analog sections. Spurious noise or reflections from wireless communication may induce fake signals that can cause false events to occur, or even damage the whole wireless system. From the software design point of view, resource constrained hardware makes the operating system simplicity a most crucial design criterion. Not only can the OS kernel deal with various timing constraints imposed on the rapid responses to external events, but it can save energy to extend the node lifetime [6]. When the OS kernel has no task scheduled, the node may even halt the system clock completely until the MCU is required, at which point an external event (such as the arrival of a RF packet) or an internal event (such as the overflow of a timer) reactivates the system clock. The

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. T-H Meen.

* Corresponding author. Tel.: +886-955-755-218.
E-mail address: ccchang@cyut.edu.tw, dyson0703@gmail.com (C.-C. Chang).

http://dx.doi.org/10.1016/j.compeleceng.2015.11.028
0045-7906/

node provides various power-save modes, each with a different level of power consumption and each requiring a different period time from the sleeping state to the first instruction executed [7]. Interrupts are a hardware mechanism used to notify the MCU that an event has taken place and need to process its service. It seems that event-driven Oss [8,9] can supply rapid responses to events. Nevertheless, some inherent drawbacks of event-driven OSs (such as interrupt overload, stack overflow, interrupt latency, and missed interrupt) will result in lack of accuracy, robustness and reliability [10,11]. As events manifest only rarely, it is also difficult to track down a variety of software errors. Most battery-driven applications use the radio event to invoke the sleeping node, but the radio transceiver always consumes a large amount of power during the transmission intervals [12]. For extending the node lifetime, it is indispensable to turn off the radio transceiver when not in use. The time synchronization protocol can use an asynchronous timer to wake up the sleeping node at regular intervals and then to turn on the radio for transformation. When the node is in power-save mode, the MCU basically halts all generated clocks, allowing operation of asynchronous timers only. The dedicated clock source is always from an external low-frequency crystal. Not only can it effectively extend the node lifetime but it can reduce the source of electromagnetic interference. It is essential that the OS kernel can possess periodic nature for deploying wireless sensor networks. The contribution of this paper is that we have implemented EXOS, a timer-driven embedded operating system for resource-constrained platforms. Contrary to the event-driven OS, the overflow of an asynchronous timer in EXOS is mainly used to divert the MCU from the power-save mode or the execution of the current sporadic task so that it can deal with the periodic task scheduled. Following are descriptions of some attractive features in EXOS.

Simplicity: For the limited code memory size, simplicity should be the primary design of the OS kernel. This can lead to two challenges. The first challenge is how to possess both the minimal code footprint of the OS and rapid responses to external events. EXOS removes unnecessary expensive services to acquire a tiny kernel which consists of a two-level scheduler, a power-save module, and an optional error detector. The scheduler can handle both periodic tasks and sporadic tasks. The second challenge is how to construct and debug a WSN application as readily as possible. All WSN applications running on EXOS can be divided into dozens of tasks, and each task should be linked together. Periodic tasks are dispatched on a regular basis between fixed time intervals. In contrast, sporadic tasks are activated by external signals or a change of some relationship.

Portability: EXOS is implemented in the standard C programming language, which can easily interface their C code to routines written in assembler. There are two occasions to call an assembly routine from the C program. First, the time-critical code is needed to execute quickly and efficiently. Second, to cram the C code into a small amount of memory is impossible. Typically, using in-line assembly with the C code can get the best of both worlds. The EXOS kernel is layered on Hardware Abstraction Layer. Once the hardware abstract layer has been ported to the target's MCU architecture, it can be easily ported to any memory-constrained platforms.

Reactivity: Unlike pure polling, even-driven OSs always use interrupts to reduce the overhead of detecting events. That is the reason why the event-driven OSs have rapid responses for external events. However, they may result in some inherent problems: interrupt overload, stack overflow, interrupt latency, and missed interrupt. EXOS is similar to Pont's system. Unlike Pont's system [13], the EXOS kernel does not limit the use of interrupts. In other words, EXOS is able to support both polling-like and event-driven.

Logical Concurrency: Most OSs put their emphasis on concurrency to improve system performance [14]. In general, interrupts are used to implement logical concurrency. The OSs use concurrency to represent logically parallel activities, even though these scheduled tasks are processed by a single MCU. As events arrive simultaneously, so does demand for more logically critical designs growth. However, a single MCU cannot execute two processes at the same time, even though interrupts can provide rapid responses to deal with sporadic events. In EXOS, the tasks will be staggered carefully to represent accuracy and logical concurrency. Thus, it can greatly reduce the OS overheads incurred by context-switching.

Predictability: Once sensor nodes are deployed in outdoor environments, it is nearly impossible to change batteries. Shutting down unused modules in the sensor node can obtain long node and network lifetime. Inefficient power-critical code blocks, like low-level communication and MAC protocols, may consume a large portion of the CPU processing time. It is indispensible to quantitatively predict the actual energy consumption of sensor nodes and to optimize the power management intended for all layers of the system. However, event-driven OSs are difficult to evaluate the energy consumption of specific code blocks due to interrupts with varying priorities. Erroneous energy consumption may result in the useless performance analysis and even a whole network crash [15]. Making a deep and accurate energy analysis during execution is inherent in the EXOS kernel. The energy profiling of EXOS can reveal which task or component consumes a large portion of the CPU load.

The rest of this paper is organized as follows: First, we discuss some inherent drawbacks in the event-based OSs in Section 2. Section 3 presents a detailed look at the EXOS system and demonstrates that EXOS is highly feasible for low-rate WSNs. In the following section, our experiences running on EXOS lend support to the claim in the earlier section. Finally, the paper is concluded in Section 5.

2. Related work

Polling is a simple programming style, which makes the MCU continuously detect whether the event takes place. It is highly portable but operates at active mode at all times. If the application has multiple interrupt sources, polling will not provide rapid responses for events. Interrupts are a hardware mechanism of diverting the MCU from the execution

Please cite this article as: C.-C. Chang, C.-B. Lin, A timer-based operating system for ZigBee sensor platforms, Computers and Electrical Engineering (2016), http://dx.doi.org/10.1016/j.compeleceng.2015.11.028