International Conference on Recent Innovation in Engineering, Science and Management (RIESM-19)
Osmania University Centre for International Program, Osmania University Campus, Hyderabad (India)
23rd November 2019 www.conferenceworld.in
ISBN : 978-81-943584-3-5

Reduced Energy Optimization in Operating System

Khalid Hassan

Research Scholar, Department of Mathematics, Magadh University, Bodh-Gaya

Abstract:

This paper reveals that the memory and energy optimization strategies are essential for the resource-constrained wireless sensor network (WSN) nodes. In this article, a new memory-optimized and energy-optimized multithreaded WSN operating system (OS) Live OS is designed and implemented. Memory cost of Live OS is optimized by using the stack-shifting hybrid scheduling approach. Different from the traditional multithreaded OS in which thread stacks are allocated statically by the pre-reservation, thread stacks in Live OS are allocated dynamically by using the stack-shifting technique. As a result, memory waste problems caused by the static pre-reservation can be avoided. In addition to the stack-shifting dynamic allocation approach, the hybrid scheduling mechanism which can decrease both the thread scheduling overhead and the thread stack number is also implemented in Live OS. With these mechanisms, the stack memory cost of Live OS can be reduced more than 50% if compared to that of a traditional multithreaded OS. Not is memory cost optimized, but also the energy cost is optimized in Live OS, and this is achieved by using the multi-core "context aware" and multi-core "power-off/wakeup" energy conservation approaches. By using these approaches, energy cost of Live OS can be reduced more than 30% when compared to the single-core WSN system. Memory and energy optimization strategies in Live OS not only prolong the lifetime of WSN nodes, but also make the multithreaded OS feasible to run on the memory-constrained WSN nodes.

Keywords: memory optimization, energy conservation, operating system, wireless sensor network, multi-core

Introduction:

A wireless sensor network (WSN) consists of distributed wireless sensor nodes which monitor the environmental conditions (temperature, sound, pressure, *etc.*) and send the collected data cooperatively through the network to a main location (e.g., the sink node) [1,2]. Currently, WSN technology has played a significant role in daily life [3,4] and is viewed as one of the key technologies of the 21st century [5].

For widespread use in different application domains, WSN nodes need to be small and inexpensive. Consequently, WSN nodes are mostly constrained in the memory resource (e.g., MicaZ node has only 4 KB RAM) and energy resource (most nodes are powered by small-sized batteries) [2]. Therefore, the memory and energy optimization strategy becomes essential for the WSN node. With these optimizations, the lifetime of the WSN nodes can be prolonged.

ISBN : 978-81-943584-3-5

Moreover, a software system, which has high memory cost, can become more feasible to run on the nodes.

An operating system (OS) is important for the WSN as it can manage the hardware resources and serve the application development. The current WSN OSes can be classified into two kinds: event-driven OS and multithreaded OS [6,7]. In the event-driven OS, preemption is not supported; one task can be executed only after the previous one runs to completion (Figure 1b). Due to this feature, memory cost of event-driven OS is low [8]. However, tasks in the event-driven OS cannot be blocked during the run-time, thus split-phase state-machine programming is commonly needed [9], and this is difficult for common users to manipulate. Moreover, the real-time performance of the event-driven system is poor, since the time-critical task cannot be executed immediately by the preemption. Multithreaded OS is different from the event-driven OS in that several tasks can run concurrently, and the split-phase programming is not necessary. Moreover, real-time scheduling can be achieved by using the thread preemption (Figure 1a). Since the preemption can be performed, each task in the multithreaded OS needs to have an independent run-time stack. Consequently, memory cost of the OS becomes high, and this makes the multithreaded OS not suitable to run on the severe resource-constrained WSN nodes. Therefore, the memory optimization mechanism becomes significant for the multithreaded OS, especially on the memory-constrained WSN nodes. Besides the memory optimization, energy optimization is also critical for a multithreaded WSN OS [2]. With the energy optimization, the lifetime of WSN nodes can be prolonged. As a result, the work of recollecting all the deployed nodes at intervals to recharge the energy can be eased. This is quite essential for the nodes deployed outdoors in harsh environments.

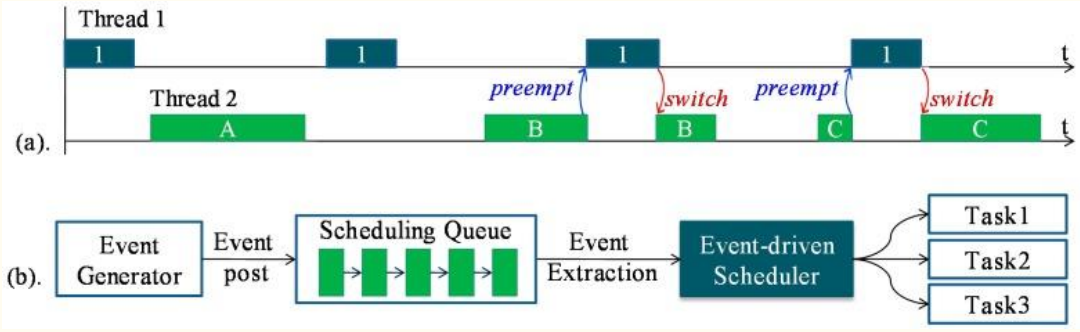

Figure-1

(a) Multithreaded scheduling system. Threads can preempt each other, thus each thread needs to have an independent stack; (**b**) Event-driven scheduling system. Events which are triggered will be posted into the scheduling queue, and then be withdrawn one by one by the scheduler. Tasks cannot preempt each other, and only one shared run-time stack is needed.

ISBN : 978-81-943584-3-5

In this article, a new multithreaded WSN OS LiveOS (former name MIROS) is designed and implemented. LiveOS aims to be a both memory-efficient and energy-efficient multithreaded WSN OS. To achieve the memory-efficient objective, LiveOS uses the stack-shifting hybrid scheduling mechanism. In contrast to a traditional multithreaded OS in which the stacks are allocated statically by the pre-reservation, stacks in LiveOS are allocated dynamically in terms of the run-time requirement. By doing this, memory waste caused by the static pre-reservation can be avoided. Since the dynamic-stack scheduling has higher run-time overhead, the hybrid scheduling mechanism, which can reduce the thread preemption frequency, is implemented in LiveOS. With the hybrid scheduling, the memory cost of the dynamic-stack scheduling can be decreased, while the performance of the dynamic-stack scheduling is maintained. To achieve the energy-efficient objective, LiveOS implements two energy-conservation approaches: the multi-core "context-aware" approach and the multi-core "power-off/wakeup" approach. As opposed to the other conservation approaches in which energy is conserved only from the software aspect, energy conservation in LiveOS is achieved from both the software aspect and the multi-core hardware aspect. By so doing, energy utilization efficiency is improved and the lifetime of WSN nodes can be prolonged.

The Stack-Size Analysis Approach

In a traditional multithreaded OS, the size of each stack is pre-reserved heuristically. If the reserved size is too small, memory overflow will occur. To avoid this problem, stack size is commonly reserved to a large value, e.g., 128 bytes in the mantisOS (on the 8-bit AVR microcontroller). However, memory waste will occur in this case. Ideally, the stack size will be reserved to the minimal but system-safe size, and this is the objective of the stack-size analysis approach. With the stack-size analysis approach, the thread execution process can be modeled as a control-flow graph. The thread functions and the local stack usages represent the nodes in the graph, whereas the branch instructions represent the edges in the graph (Figure 2). After this model is built, the stack usage of each thread can be calculated by the straightforward depth-first search. During the search process, if a "PUSH" or "CALL" instruction is observed, the stack usage value will increase. Otherwise, if a "POP" instruction is met, the stack usage value will decrease. By doing this, the stack size which will be required during the run-time can be computed, and the stack memory waste caused by the heuristic pre-reservation approach can be avoided.

ISBN : 978-81-943584-3-5

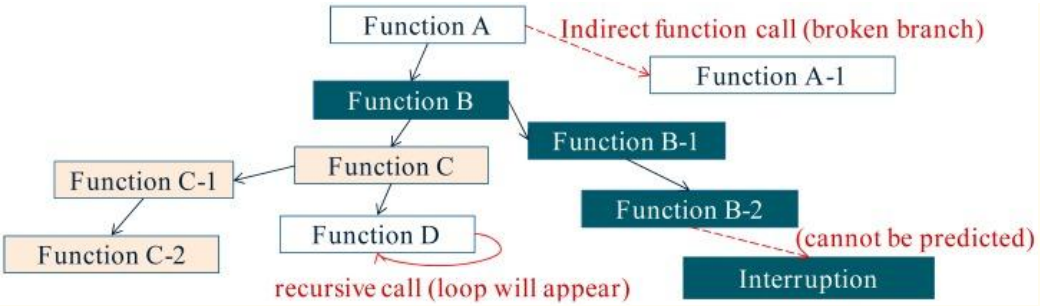

Figure-2

The thread control-flow graph modeled by the stack-size analysis approach. Stack usage can be calculated by the straightforward depth-first search in this graph.

With the stack-size analysis approach, the thread memory cost can be optimized. However, there are several cases in which the stack usage cannot be calculated, such as interruption reentrancy, recursive calls and indirect function calls. If interrupt is reenterable, the number of received interruptions during the run-time cannot be pre-known. If recursive calls exist, the cycle will appear in the control-flow graph. If indirect function calls exist, the disconnection will appear in the control-flow graph (Figure 2). To solve these problems, several solutions are proposed in the article [10]. However, the final evaluation results show that these solutions are not ideal, although the guesswork in determining the stack size is eliminated. As a result, stack-size analysis approach is not effective in some WSN systems.

Energy Conservation to the WSN Node

The energy conservation strategy is significant for the WSN nodes for two key reasons. First, many WSN nodes are deployed outdoors and need to be powered by small-sized batteries; the available energy resource is therefore limited. Second, WSN nodes are prone to be deployed in the harsh environment where human access is difficult; thus the recollection of the deployed nodes to recharge the energy is quite difficult. Since the lifetime of the nodes is determined by the energy supply, the energy conservation mechanism is essential to the WSN nodes. Currently, the "sleep/wakeup" approach is used popularly in most WSN OSes to conserve energy resources, and this approach has been achieved through two aspects: the OS scheduling aspect and the network protocol aspect. In TinyOS, Contiki and SOS, the "sleep/wakeup" mechanism is achieved from the scheduling aspect. A scheduling queue is polled in these OSes by the event-driven scheduler. In the case that no events are pending in the queue, the sleep directive will be called, and then the node will fall asleep to conserve energy. In openWSN, the "sleep/wakeup" mechanism is also applied, but it is achieved from the network protocol aspect. In openWSN, the IEEE 802.15.4e protocol is developed. With this protocol, the WSN nodes can wake up synchronously to communicate with each other within a given period, whereas fall asleep or power off the transceivers during the other periods.

ISBN : 978-81-943584-3-5

The "sleep/wakeup" approach can conserve the node energy resource in a degree. However, the conservation effect is not adequate. Currently, the energy constraint is still a critical challenge for the WSN nodes [2], and the research of the new conservation approaches is still significant.

Discussion

The mechanism of using the multi-core technique to achieve the energy conservation in LiveOS is presented. Besides the energy conservation, the multi-core technique can also be used to improve the OS real-time performance, the node reliability, *etc*.

The real-time performance can be improved in the multi-core node by distributing different tasks onto different microcontrollers to be executed concurrently. In the single-core node, if several real-time tasks become active simultaneously, these real-time tasks may not be schedulable even if an appropriate real-time scheduling algorithm is used (CPU computation resource is not enough). However, this problem can be solved on the multi-core node if these tasks are distributed.

In addition, the node reliability can be improved by using the multi-core WSN technique. If the single-core node is used, the node will fail once the microcontroller runs incorrectly. However, if the multi-core node is used, a high reliable auxiliary microcontroller can be used to manage the less reliable working microcontroller. Then, once the working microcontroller runs abnormally, the auxiliary microcontroller can catch this event and restart it. By doing this, the working microcontroller can be recovered from the faults and the WSN tasks can continuously be performed. Since the reliability of the auxiliary microcontroller is high, multi-core WSN node becomes more reliable if compared to the single-core node (auxiliary microcontroller can be high reliable as the program running on it is quite simple). Currently, this multi-core reliability approach has been applied to the LiveWSN node. With this approach, three deployed LiveWSN nodes have been working effectively for more than two years in the ISIMA garden.

Since many microcontrollers are equipped, as opposed to just one, the manufacturing price of the multi-core node will increase. However, the increase is not high. This is because most WSN microcontrollers are low-cost ones, e.g., the selling price of the AVR Atmega1281 microcontroller can be lower than five dollars (the selling price of some Ph sensors can be more than 100 dollars; the microcontroller is relatively cheaper). As a result, the price of the multi-core node is acceptable. More significantly, the integrated performance of the WSN node can be improved significantly by using the WSN multi-core technique (energy cost can be lowered, real-time performance and node reliability can be improved, *etc*.). Therefore, it is essential to use the multi-core technique in the WSN.

Conclusions

In this paper, the memory and energy optimization strategies for the multithreaded WSN OS LiveOS is presented. LiveOS uses the stack-shifting hybrid scheduling mechanism to achieve the

ISBN : 978-81-943584-3-5

memory optimization objective. By using the stack-shifting dynamic allocation, the size of each stack in LiveOS can be reduced. By using the hybrid scheduling, the number of the stacks in LiveOS can be decreased. As a result, LiveOS becomes memory-efficient and can cost the memory resource less than 50% of that in the traditional multithreaded mantisOS. With this memory optimization, LiveOS becomes feasible for running on the memory-constrained WSN nodes (TelosB, SenseNode, iLive, *etc.*). Not only optimized in terms of memory cost, LiveOS is also optimized for energy saving, and this is achieved by using the multi-core "context-aware" and multi-core "power-off/wakeup" energy conservation approaches. With these approaches, more than 30% of the energy resource can be conserved by LiveOS when compared to a typical single-core WSN system. With the energy optimization strategy, the lifetime of WSN nodes can be prolonged. Consequently, the WSN nodes become more competent to be deployed in harsh outdoor environments. Currently, an online demo about LiveOS can be accessed from the websites.

References:

1. Dargie W., Poellabauer C. Fundamentals of Wireless Sensor Networks: Theory and Practice. John Wiley & Sons; Chichester, UK: 2010.

2. Akyildiz I. F., Vuran M.C. Wireless Sensor Networks. John Wiley & Sons; Chichester, UK: 2010.

3. Sohraby K., Minoli D., Znati T. Wireless Sensor Networks: Technology, Protocols, and Applications. John Wiley & Sons; Chichester, UK: 2007.

4. Yick J., Mukherjee B., Ghosal D. Wireless sensor network survey. Comput. Netw. 2008;52:2292–2330.

5. Gross N. Ideas for the 21st century. Business Week. 1999 Aug 30;:78–167.

6. Ousterhout J. Why threads are a bad idea (for most purposes). Proceedings of the 1996 Usenix Annual Technical Conference; San Diego, CA, USA. 22–26 January1996;

7. Von Behren R., Condit J. R., Brewer J. Why Events Are a Bad Idea (for High-Concurrency Servers). Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX); Lihue, HI, USA. 18–21 May 2003; pp. 19–24.

8. Li S.F., Sutton R., Rabaey J. Compilers and Operating Systems for Low Power. Kluwer Academic Publishers; Norwell, MA, USA: 2003. Low Power Operating System for Heterogeneous Wireless Communication Systems; pp. 1–16.

ISBN : 978-81-943584-3-5

9. Levis P. TinyOS programming. 2006. [(accessed on 19 September 2014)]. Available online: http://www.tinyos.net/tinyos2.x/doc/pdf/tinyos-programming.pdf.

10. Torgerson A. Master's Thesis. University of Colorado; at Boulder, Boulder, CO, USA: May, 2005. Automatic Thread Stack Management for Resource-Constrained Sensor Operating Systems.

11. AbsInt Corporation Stack Analyzer: Stack Usage Analysis. [(accessed on 6 September 2014)]. Available online: http://www.absint.com/stackanalyzer/index.htm.

12. Bhatti S., Carlson J., Dai H., Deng J. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. ACM Kluwer Mob. Netw. Appl. J. 2005;10:563–579.