Micro real-time pre-emption operating system for industry wireless sensor networks

Kanghong Duan, Hongxin Zhang, Shilin Song, Peigang Wang*

North China Sea Marine Technical Support Centre of State Oceanic Administration, 22 Fushun Road,, Qingdao, China,266033

Received 1 January 2014, www.tsi.lv

Abstract

Event-driven systems and thread-driven systems are two major design philosophy of operating system in wireless sensor networks. Systems based on multi-threaded are more timeliness than the event-driven systems, which can meet the requirements of time-critical tasks by means of task pre-emption, while systems based on event-driven are more energy efficient. Furthermore, μCOS-II is a classical system, which combines benefits in both systems. Therefore, our recent work we have shown that a micro real-time pre-emption operating system has been proposed on the basis of μCOS-II. First of all, a clear hardware abstraction layer (HAL) is given to combine the kernel and hardware in the system architecture. Moreover, this system is more capable of fitting both sensor network design goals of energy efficiency and timeliness. We are dedicated to modify the existing system from the scheduling strategy and data structure aspects, which lead to the performance of the modified system largely improved. Above all, the performance of our operating system is better than the original μCOS-II and TinyOS from task switch time, FLASH usage and RAM usage perspectives.

Keywords: Wireless Sensor Network, Operating System, Pre-emption, Improvement

1 Introduction

With the progress of sensor technology, communication technology and computer networking technology, WSN (Wireless Sensor Network) has been developed and improved rapidly. As one of the front fields, it brings us a lot of challenges, and the embedded operating system is one of them.

The embedded operating system that we research on must have the key functions as task scheduling, I/O management, timer management and so on. At the same time, it should pay more attention to the professional filed to meet the need of WSN.

Besides the general features in sensor networks, WSN has some its own features, which are important for the Industry WSN OS (Wireless Sensor Network Operating System).

The capability of the hardware is limited. The WSN node usually takes AA battery as power supply, and its microcontroller usually is 8-bit or 16-bit. At the same time, many applications and services may be limited by its memory spaces. These limitations require the operating system to be small and efficient [1].

The network should be large-scale, self-organized, dynamic and reliable. The amount of the node is large and the network topology changes quickly. Therefore, the operating system should have good methods to enhance its haleness, fault-tolerance and self-repair [2].

Wireless sensor network has very strong application relativity, and different hardware platforms and software systems are needed under different application requirements [3]. So the operating system should be transplanted easily to meet the demands of different applications.

Many researchers has designed some WSN operating systems with high practical applicability. Based on different scheduling strategies, they can been divided into two kinds [3]:

- one is pre-emptive operating system oriented to hard real-time applications like MANTIS [4] and Contiki [5],
- the other is non-pre-emptive operating system oriented to soft real-time applications like TinyOS [6,7]. But these operating systems still have some disadvantages.

2 Study on μCOS-II

2.1 THE KERNEL OF μCOS-II

The kernel of μCOS-II is fully pre-emptive and real-time with multitask management; it can comparable on performance with most commercial kernels [8]. At the same time, it is only a Microkernel without many applications. Therefore, the architecture of it is easy to catch. FIGURE 1 shows the theoretical architecture of the kernel.

Most of the kernel's code is written by C language, only a few of the processor specific code is written by assembly language. And thanks to the clear hierarchical structure, the kernel gets the portal and scalable features which is crucial for WSN. And, the processor must

*Corresponding author - E-mail: dkhkidd@163.com satisfy the following requirements [9]:

1) C complier can generate reentrant code;
2) Interrupt can be disabled or enabled by C;
3) The processor must support interrupt and be able to generate timing interrupt;
4) The processor must support hardware stacks;

The processor must have instruction to load and store stack pointer and other registers to RAM.

2.2 SCHEDULE STRATEGY OF μCOS-II

μCOS-II uses the priority-based pre-emptive scheduling strategy. The base unit for scheduling in it is task. One task includes three main components: Program code in memory, TCB (Task Control Block) and stack space. Each task has its own priority and no task has the same priority while there are 64 priorities managed in μCOS-II. In order to ensure the schedule strategy, the kernel μCOS-II provides system services for tasks. FIGURE 2 shows the relationships between tasks and system services [9].

A task can request certain kinds of services of the kernel, then the kernel responses to corresponding requests. At the same time, the kernel executes the ready task with highest priority according to the state of the current task. The real-time is ensured by strategy of pre-emptive task scheduling.

2.3 THE MECHANISM OF TASK MANAGEMENT

After all codes downloading in the device, we can think that the whole system is ready to work. The first thing is starting the operating system and change the program to tasks.

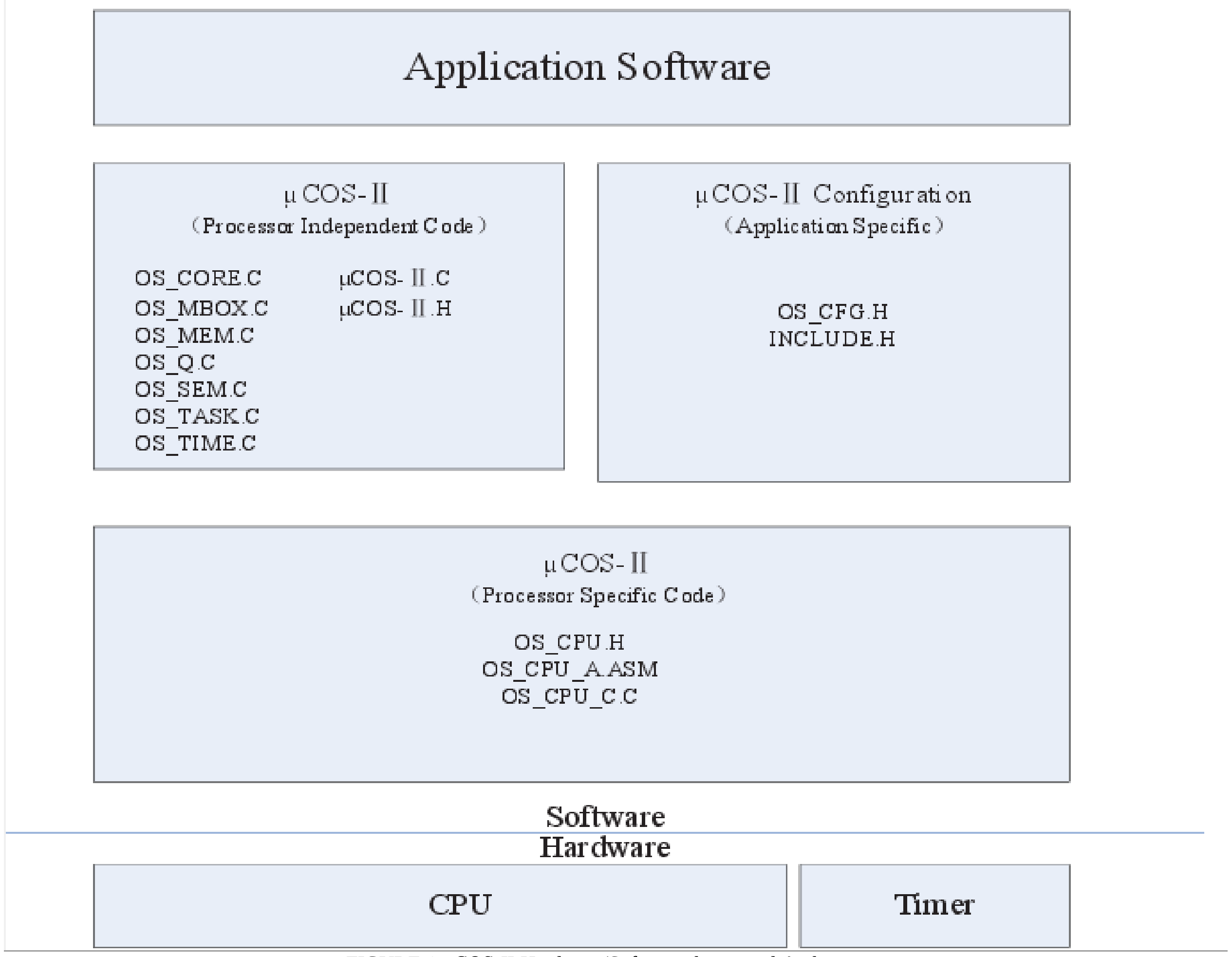


FIGURE 1 μCOS-II Hardware/Software theoretical Architecture

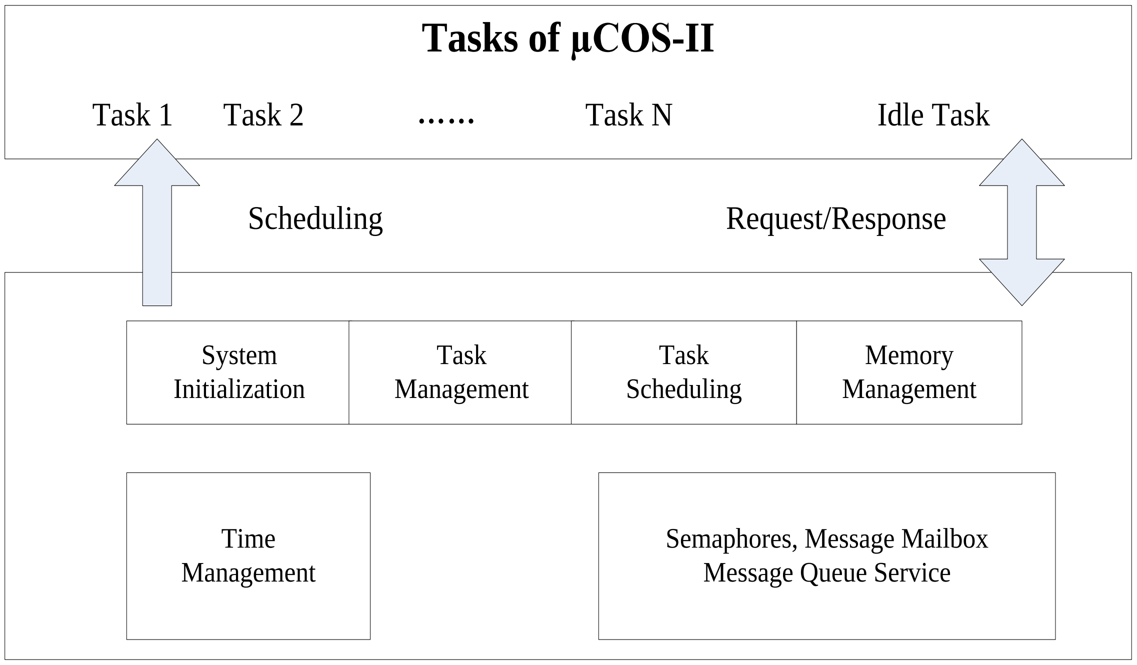


FIGURE 2 The relationship between tasks and system services

There are four states designed for every task: Dormant, Ready, Waiting and Running:

1) The dormant state is for tasks, which are deleted or not created. On this state, task does not have stack space and TCB while it cannot be scheduled. In other words, any dormant task is invisibility to the operating system.
2) The ready state is for tasks, which are ready for being scheduled. On this state, all the resources that task will use for running have been prepared well. And the task only need to wait until all the tasks who have the higher priority finish their work.
3) The waiting state is for tasks, which are waiting some resources necessary for running. On this state, task has stack space and TCB so that it can wait the necessary resources. After the resources are prepared, the waiting task will change its state to Ready.
4) The running state is for the task which is running at the moment. The running task has all the necessary resources and its priority is the highest.

In addition, there is ISR state for the interrupt. When a real-time call happens, the ISR will be triggered and OS will operate the new call first in order to ensure a strong real-time response.

The task management for μCOS-II is to ensure the following things:

1) Ensure the tasks in *Dormant* state occupying no RAM space.
2) Ensure the tasks in *Ready* state arranging in order.
3) Ensure the tasks in *Waiting* state preparing their resources in order.
4) Ensure the task in *Running* state running without disruption.

2.4 THE FEASIBILITY FOR TRANSPLANTING μCOS-II TO WSN

In the introduction we have given the three features which are important for WSN OS. Then we can research the feasibility based on the three features:

1) μCOS-II has a kernel perfect in function. But for WSN node, the memory capacity is a bottleneck for μCOS-II. Thanks to its scalable feature, we can improve some data structure and delete some useless functions for WSN in order to transplant μCOS-II.
2) μCOS-II has been used by hundreds of applications and its robust and reliable features are suitable for WSN.
3) Most of μCOS-II is written in highly portable ANSI C, with target microprocessor specific code written in assembly language. It determines the portable feature for μCOS-II. And the feature means we can transplant μCOS-II for WSN easily if we design a good platform for it.

The research tells us that μCOS-II is suit for WSN if the following things are done well:

1) Improve some data structure needing too much memory
2) Simplify some functions useless for WSN
3) Design a peripheral support platform for the μCOS-II

3 Target hardware platform for transplanting

Before improving μCOS-II for WSN, it is important to find a proper hardware platform for the research. In this paper, we use a normal WSN node hardware platform: ATmega128L micro-controller and CC2420 RF transceiver chip. The main hardware resources are as follows:

CPU: 8bit micro-controller with 133 powerful instructions and on chips 2-cycle multiplier.

Memory: 4KB RAM and 128KB Flash ROM on chips.

RF chips: True single-chip 2.4GHz IEEE 802.15.4 compliant RF transceiver with baseband modem and MAC support.

4 Improving μCOS-II for WSN

The trend of WSN's design is for specifically applications rather than the general applications. And different applications use different hardware platforms. Therefore, it is essential to change the details when you transplant μC/OS-II to different platforms. Nevertheless, the principle for transplanting is universal.

4.1 IMPROVE THE ARCHITECTURE

For μCOS-II only provides a portable kernel, so the first thing for the improvement is to combine μCOS-II with the actual hardware. Then we can improve the original architecture of μCOS-II for the actual use. FIGURE 3 shows the improved architecture.

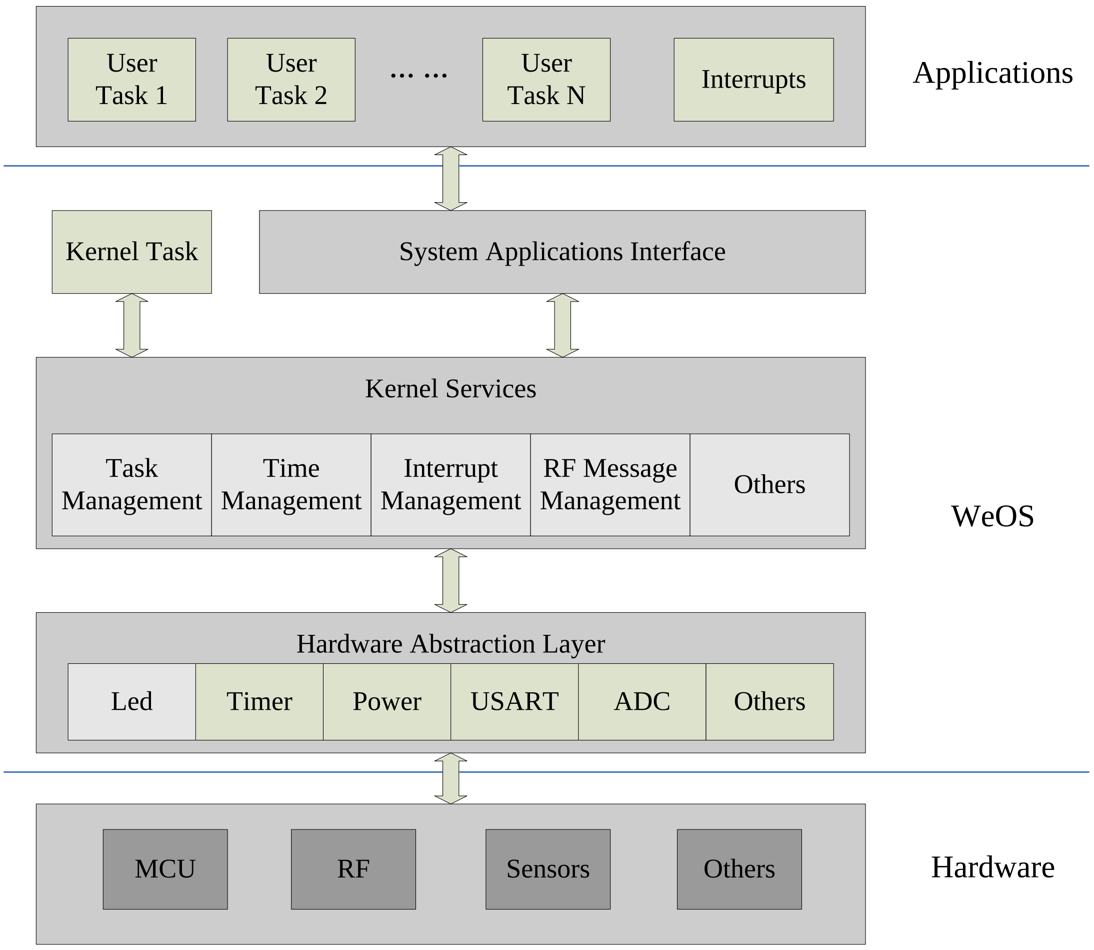


FIGURE 3 The improved architecture

There are two main improvements in the architecture:

1) Give a clear HAL (Hardware Abstraction Layer) to combine the kernel and hardware. And if any hardware should be added, moved or replaced, we only need to modify the corresponding code in the HAL without doing anything in the kernel and applications. By means of the improvement, we can transplant μC/OS-II to the other platform more easier especially when we use a new kind of WSN node.
2) Add a new kernel task. As mention above, the capability of the hardware especially the power supply is limited. In order to reduce energy use, we need to make the node sleep when there is no task need to be run. Therefore, we add an individual kernel task to finish the work. The kernel task has the lowest priority (always in Ready State) so that it is scheduled only when there is no user task ready. At the same time, there is no need to give any stack spaces to the task and the task can use the kernel service directly. By means of the improvement, the node can transform into sleep state by itself so that energy use will be reduced.

4.2 IMPROVE THE SCHEDULING STRATEGY

The scheduling strategy plays an important role in the performances of the real-time, power and reliability. μC/OS-II has a balanced scheduling strategy, but we can get a more suitable Scheduling Strategy for WSN by some improvements. FIGURE 4 shows the scheduling strategy.

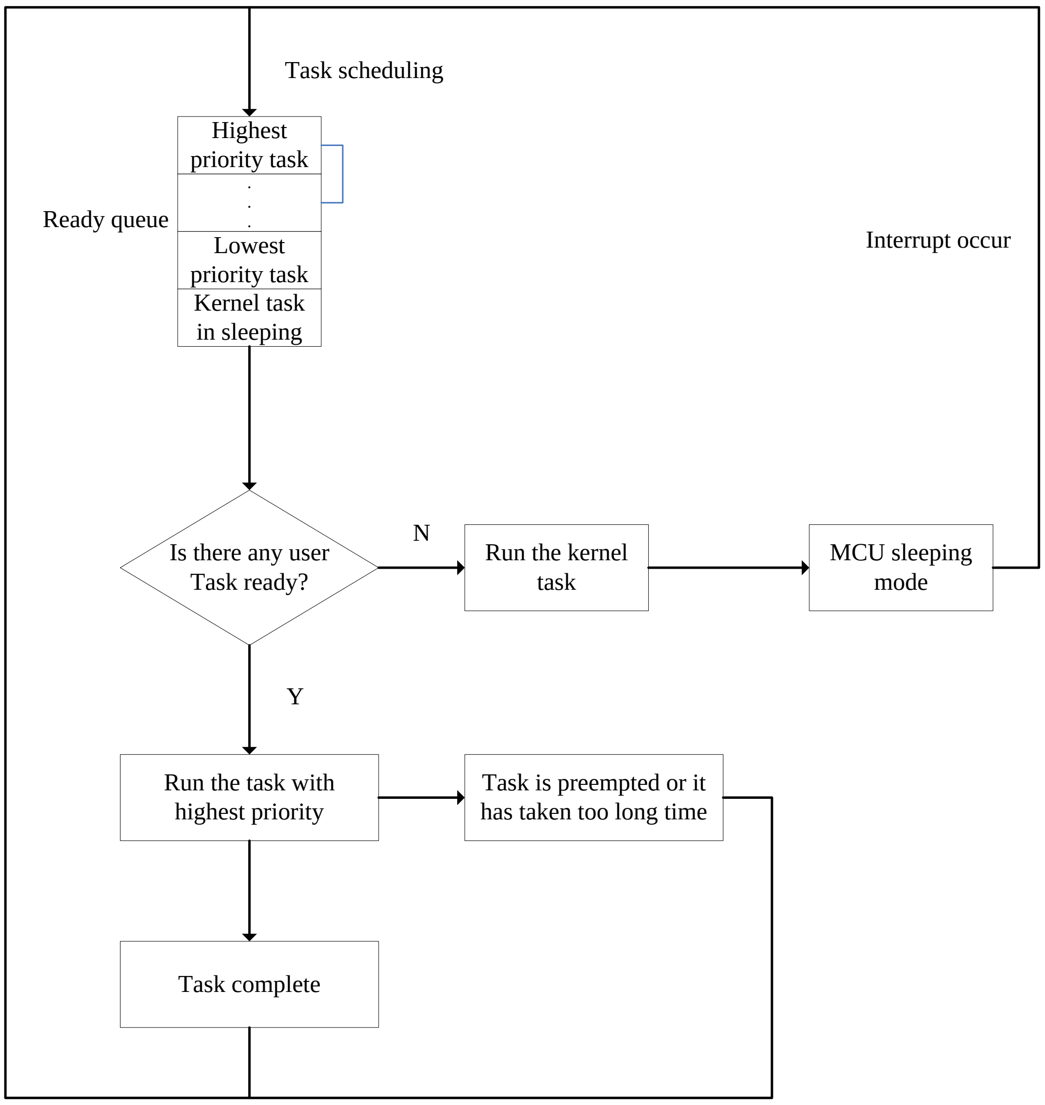


FIGURE 4 The improved scheduling strategy

There are two main important points in the improved strategy:

1) Modify the ready table. As mentioned above, we add a new kernel task with lowest priority and Ready State in order to make the node sleep when there is no user task ready. So we add a new judgement. When a procedure of task scheduling occurs, we first exam if there is any user task being in Ready State. If not, the kernel task will be scheduled and the MCU will get into Sleep Mode until an interrupt occurs.
2) Add a rotary-time scheduling method into the strategy. The original strategy takes the priority-based scheduling method and it has been proved excellent in performance. But for actual applications especially in WSN, it is easy for users to create a high priority task with endless loop. It is clear that the problem will cause huge harm to the whole system even influence the other nodes. Therefore, we add a rotary-time scheduling method with hardware timers. And in the actual use, we first use the priority-based scheduling method to ensure the high priority task run smoothly, and the rotary-time scheduling method only ensure the task with endless loop can be stopped by the scheduler to make the whole system run well.

We use the following symbols to model the task of our operating system:

$S=\{s_1,s_2...s_n\}$ is the set of task;

$B=\{b_1,b_2...b_n\}$ is starting time for each task;

$L=\{l_1,l_2...l_n\}$ is time slice for each task;

$P=\{p_1,p_2...p_n\}$is priority for each task;

$T=\{t_1,t_2...t_n\}$ is the ending time for each task;

Then we model the task of our operating system as (S, B, L, P, T).

For task s_i, its priority is pi. And when s_i is called, the bi will be recorded, and when the task is over (not include pre-empted by event), the ti will be recorded. If the task s_i, gets stuck in an infinite loop, when time slice li is over, s_i, will lose the control and change pi to the lowest priority.

So we can get the worst waiting time for s_i:

$$T_{Waiting} = T_{Initial} + \sum_{k}^{i-1} l_k \ . \tag{1}$$

The improved OS simplies many practical functions for applications in μC/OS-II and a new scheduling state method shown in FIGURE 5.

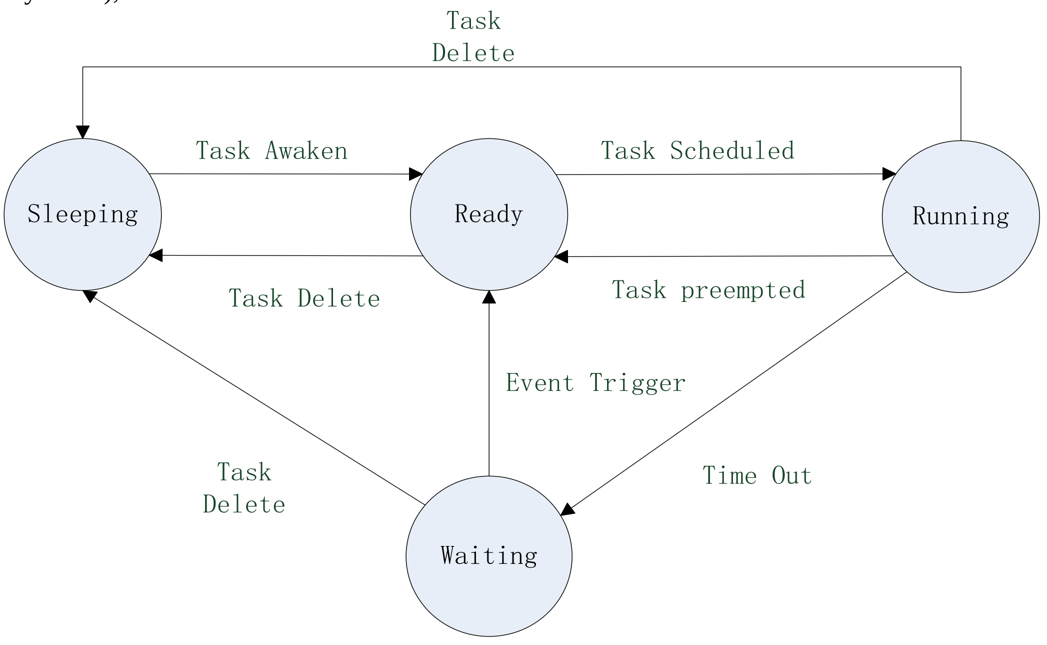


FIGURE 5 The improved task state

4.3 IMPROVE THE DATA STRUCTURE NEEDING TOO MUCH MEMORY

The limitation of memory capacity is the main problem for transplanting μCOS-II to WSN, especially the RAM space. So the main point that can be improved is to reduce the consume of RAM space for μC/OS-II.

The first thing we can do is to optimize the size and type of the global variable. In μC/OS-II, many global variables are defined as 32bits-integer data type which is usually useful for our WSN node. So we can change the 32bits-integer into 8bits-integer or 16bits-integer. At the same time, we can change the global variables which are not used frequently into extern global variables. So that these variables are not stored in RAM. And the RAM space can be saved a lot by the means above.

Another thing we can do is to enhance the table management. In WSN, the implementation of networking protocol usually uses many tables like routing table, neighbour table and so on. These tables usually are treated as temporary variables stored in RAM. Therefore, we can enhance the management of these tables by limiting table size and setting public table RAM spaces to reduce the RAM consume.

The main members of RAM are global variables and temporary variables. So improving the size and type of these two variables is principle for saving more RAM space.

4.4 IMPROVE THE DATA STRUCTURE NEEDING TOO MUCH MEMORY

μCOS-II has many practical functions for applications, but in WSN, we don't need so many functions, so it's useful to delete some functions in order to get a more efficient kernel:

1) Simplify the procedure of creating a new create. We delete some operations, which are useful for system programming but useless for applications such as creating a pointer to the bottom of task stack. Then we can save some memory spaces and creating time by means above.

2) Simplify the mechanism of task communication. µC/OS-II provides three kinds of methods for intertask communication: semaphores, message mailbox and message queue service. The functions of these three methods are similar. Therefore, we can choose one of them as our method. By this mean, we can get more space for applications instead of the operating system.
3) Change the method for handling interrupt. It is crucial for WSN node to response the external interrupt especially the interrupt from RF or sensors. Therefore, we get the interrupt handling out of the task state. Instead of that, we design the interrupt handling as a independent component in the whole system. By this way, we can get a well simplified kernel with a better capability for handling interrupt.

5 Performance analysis

In order to evaluate the improved µC/OS-II especially its improvement for memory saving. We compiled TinyOS, original µC/OS-II and our improved µC/OS-II and downloaded them into our hardware platform. Then, we measured some key data to analysis the performance.

FIGURE 6 shows the task switch latency when a higher priority task is ready. From the result, we can see that TinyOS has the worst performance to ensure the higher priority task to be scheduled when it is ready. At the same, our improved µCOS-II has the similar performance with the original µCOS-II. And they are about 5 times faster than TinyOS.

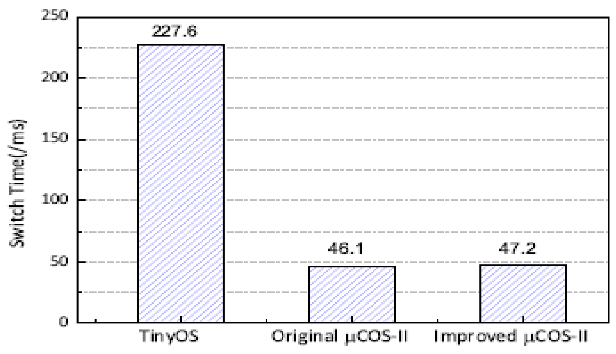


FIGURE 6 Task Switch Time

FIGURE 7 shows the Flash usage for each OS with different number and different size tasks. From the result, we can see that the original µCOS-II need about 40KB Flash space to hold the system itself. At the same time, our improved µCOS-II has similar performance with TinyOS, both of them only need less Flash space than half of the original µCOS-II.

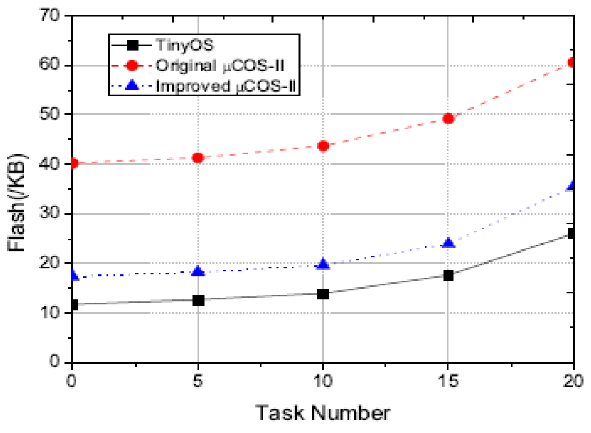


FIGURE 7 Flash Usage

FIGURE 8 shows the RAM usage for the original µC/OS-II and our improved µC/OS-II with different number tasks. From the result, we can see that the RAM space with different number tasks exhibit characteristic of linearity. The reason of the performance is that the RAM space is only relate to the initial system size and the size of task stack in µC/OS-II. The initial size for our improved µC/OS-II is about half of the original µC/OS-II. At the same time, it is easier to manage the RAM space for each task in µC/OS-II than in TinyOS.

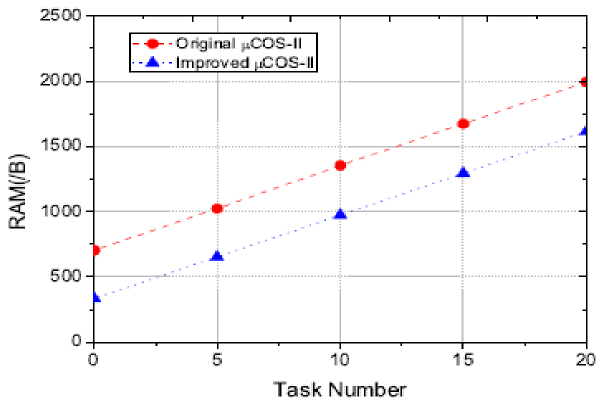


FIGURE 8 RAM Usage

Finally, we measure the performance for each OS with endless loop task. The result proves that our improved µC/OS-II can stop the task when it takes too long time to occupy the MCU.

From the measure, we can conclude that our improved µC/OS-II can provide a hard real-time scheduling for WSN with Relatively good space speeding.

6 Conclusion

This article analyses the limitations of WSN and studies the classic embedded operating system µC/OS-II detailed. The paper brings forward a micro real-time pre-emption Operating System for industry Wireless Sensor Networks and demonstrates the availability and performance of the improved operating system. It provides choices of more efficient system for the further study and application of Industry WSN.

References

[1] Akyildiz F, Su W, Sankarasubramaniam Y, Cayirci E 2002 *IEEE Communications Magazine* **40**(8) 102-14

[2] Minghai Y, Xiaoxiao Z 2007 *Wireless Communications, Networking and Mobile Computing* **1** 2803- 07

[3] Wei D, Chun C, Xue L, Jiajun B 2010 *IEEE Communications Surveys & Tutorials* **12**(4) 519-30

[4] Abrach H, Bhatti S, Carlson J, Dai H, Rose J, Sheth A, Shucker B, Deng J, Han R 2003 MANTIS: System Support For Multimodal Networks of in-situ Sensors *The 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)* **1** 50-9

[5] Dunkels A, Gronvall B, Voigt T 2004 Contiki:A Lightweight and Flexible Operating System for Tiny Networked sensors *Proceeings of the 29th Annual IEEE International Conference on Local Computer Networks* **1** 455-62

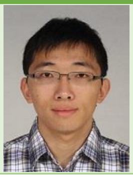
[6] Levis A 2006 TinyOS: An Open Operating System for Wireless Sensor Networks *The 7th International Conference Mobile Data Management* **1** 115-148

[7] Ying L, Hongyi Z 2007 *Transducer and Microsystem Technologies* **26**(03) 98-101

[8] Jean J L 2002 *µCOS-II, The Real-Time Kernel* Press: Higher Education Press, Chapter **6** 173–214 *(in Chinese)*

[9] Kai L, Hongzhe X, Weiran X, Hongbo H 2005 *Journal of Communication and Computer* **38**(6) 30-4

Authors



Kanghong Duan, 04 1987, Zhucheng City, Shanddong Province, P.R. China

Current position, grades: engineer, North China Sea Marine Technical Support Centre of State Oceanic Administration
University studies: Graduated from University of Science and Technology, master's degree in Computer Science
Scientific interest: Internet of Things, Marine Information, Embedded operating system.
Publications : More than 10 papers, 3 patents
Experience: an engineer in North China Sea Marine Technical Support Centre of State Oceanic Administration

Shilin Song, 11 1961, Qingdao City, Shandong Province, P.R. China

Current position, grades: professor Senior Engineer, chief engineer, North China Sea Marine Technical Support Centre of State Oceanic Administration
University studies: Graduated from Ocean University of China in 1983, Bachelor of Science in Marine Geology
Scientific interest: Marine Geology, Marine Information, Wireless Sensor Network.
Publications: more than 30 scientific research projects, more than 20 papers, 2 patents
Experience: as an engineer, Senior Engineer and professor Senior Engineer in North China Sea Marine Technical Support Centre of State Oceanic Administration

Hongxin Zhang, 01 1978, Qingdao City, Shandong Province, P.R.China

Current position, grades: Senior Engineer, vice Chief of Centre, North China Sea Marine Technical Support Centre of State Oceanic Administration
University studies: Graduated from Ocean University of China in 2000, Bachelor of Science in Physical Oceanography
Scientific interest: Physical Oceanography, Marine Information, ship information service system.
Publications: more than 10 scientific research projects, more than 20 papers, published, 6 patents
Experience: an engineer and Senior engineer in North China Sea Marine Technical Support Centre of State Oceanic Administration

Peigang Wang, 02 1962, Qingdao City, Shandong Province, P.R. China

Current position, grades: professor Senior Engineer, Chief of Centre, North China Sea Marine Technical Support Centre of State Oceanic Administration
University studies: Graduated from Ocean University of China in 1983, Bachelor of Science in Marine Biology
Scientific interest: Marine Biology, Marine Information, Marine Geology
Publications: more than 20 scientific research projects, more than 20 papers, 5 patents
Experience: an engineer, Senior Engineer and professor Senior Engineer in North China Sea Marine Technical Support Centre of State Oceanic Administration