A Dynamically Switchable Scheduling System for Operating Systems in Wireless Sensor Networks

Yoshiki Komachi[*]
Graduate School of Media and Governance[*], Keio University
ysk@sfc.keio.ac.jp

ABSTRACT

Operating Systems (OS) in wireless sensor nodes can be classified into event-driven systems or multithreaded systems. Most event-driven systems, such as TinyOS [4], drive down power consumption although context switching for real-time processing is not available. Among multithreaded systems, non-preemptive systems, such as Protothreads [2] in Contiki [1], often have lack of real-time processing capability. In Protothreads, if a higher-priority task was posted while a lower-priority task has been running, the lower-priority task cannot be preempted. Thus, one challenge is that without changing the semantics of Protothreads, how the system can be preemptive as well as lowering the power consumption for real-time tasks such as target tracking.

In this paper, we propose a dynamically switchable scheduling system for operating systems using Protothreads where events with time constraint have occurred. This system enables to trigger interruption, to process real-time tasks preferentially when real-time events occurred, and to save energy by executing tasks except real-time tasks as a standard event-driven system. Exeprimental results show that latency in Contiki is reduced by about 75% in the best case and is kept constant with power efficiency.

Keywords

Real-Time Processing; Low Power Consumption

1. INTRODUCTION

Along with cheaper and high performanced sensor nodes, Wireless Sensor Networks (WSN) are widely used in various application domains. There are various types of applications adapted to WSN, such as habitat monitoring, health monitoring, and target tracking. In target tracking applications, specific tasks (e.g., a task for detection of targets) must be processed with power efficiency as well as in real-time.

Operating Systems in wireless sensor nodes can be classified into event-driven systems or multithreaded systems. Not based on preemption, event-driven systems are built with low energy consumption. However, on event-driven systems, it is difficult to develop various applications and tasks are not processed in real-time. On the contrary to event-driven systems, most multithreaded systems which handle preemptive multitasking support real-time processing. Nevertheless, context switching makes multithreaded systems' power consumption higher.

Nano-RK [3], a multithreaded system in WSN, supports fixed-priority preemptive multitasking. This system enables hard and soft real-time processing by different two scheduling algorithms. In standard multithreaded systems (e.g., Nano-RK), variables' values are kept by being stored states of registers if interruption has occurred. The values are read again, and the processing is resumed after execution of the currently executed task was completed. Besides regular processing, these extra processing for context switching needs extra power. Therefore, power consumption of multithreaded systems is higher than other event-driven systems.

In Contiki, an event-driven system, Protothreads enables switching between tasks as same as other multithreaded systems, and processes will be executed if events are received. Events are inserted into an queue when the events occurred, and the events are kept in the queue until the events are accepted in First In First Out (FIFO) order. Unlike standard multithreaded systems, in Contiki using Protothreads, utilization of CPU will be abandoned by return values, which enables keeping power consumption lower. In case of switching between tasks, timers are not used for interruption in current Protothreads. Tasks are inserted into the queue when the timer's counts were larger than current time. The task that manages the timers is periodically executed, its priority is equal to the other tasks' priority.

2. PROBLEMS

There are tradeoffs between low power consumption and real-time processing. This is because event-driven systems are designed on the assumption that tasks are not interrupted, and context switching by saving states of registers needs much energy in multithreaded systems. However, both keeping power consumption lower under normal conditions and real-time processing in case of occurrence of real-time events are needed in the specific situations (e.g., target tracking).

Although Protothreads implemented on the event-driven system enables switching between tasks, this system is unable to interrupt an executed task, to switch to higher-priority tasks, and to execute these tasks when these tasks wait for execution. This is because the scheduler will not be executed unless an executed task issues a return instruction.

3. DESIGN

Our system consists of many event handlers and an event loop. The event loop waits for the arrival of events, and an event handler correlated with the event would be executed. During execution of the event loop, an executed task is interrupted, and the current thread is switched to a real-time thread when real-time event was issued. Execution of the

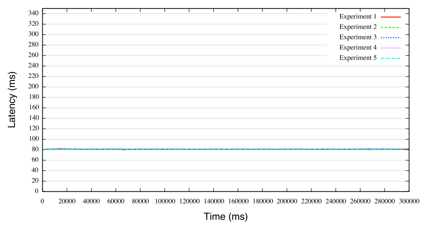

Figure 1: Latency in Our System

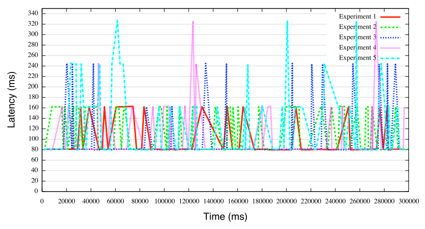

Figure 2: Latency in Contiki

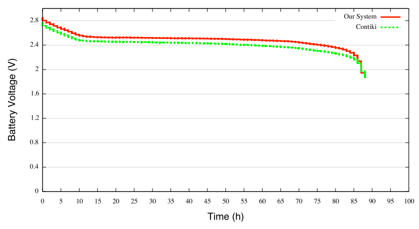

Figure 3: Power Consumption

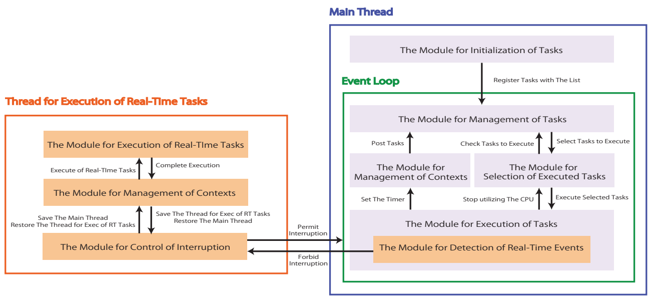
Main Thread
The Module for Initialization of Tasks
Register Tasks with The List
Event Loop
The Module for Management of Tasks
Post Tasks
Check Tasks to Execute
Select Tasks to Execute
The Module for Management of Contexts
The Module for Selection of Executed Tasks
Set The Timer
Stop utilizing The CPU
Execute Selected Tasks
The Module for Execution of Tasks
The Module for Detection of Real-Time Events
Thread for Execution of Real-Time Tasks
The Module for Execution of Real-Time Tasks
Execute of Real-Time Tasks
Complete Execution
The Module for Management of Contexts
Save The Main Thread
Restore The Thread for Exec of RT Tasks
Save The Thread for Exec of RT Tasks
Restore The Main Thread
The Module for Control of Interruption
Permit Interruption
Forbid Interruption

Figure 4: System Architecture

event loop is resumed again after execution of the real-time task was finished. Our system enables switching between tasks by Protothreads for general events except real-time events. Figure 4 shows system architecture of our system.

4. EVALUATION

4.1 Evaluation Environment

We evaluate the latency, time from occurrence of real-time events to being executed intended tasks, by increasing the number of tasks and an incidence of events little by little. In this evaluation, Cooja [5], a simulator designed for Contiki on virtual machines, are used.

In order to evaluate power consumption, an application that randomly sends current battery voltage as real-time events to the base station is deployed to actual equipment.

4.2 Discussion

In Contiki, if events waiting for execution have been already inserted at the moment of real-time events' occurence, high latency could be observed. This is because events are inserted in FIFO order. Against Contiki, when real-time events were issued, our system enables real-time tasks to be executed by interruption of current execution if a current task are being executed. This leads to lower latency.

According to Figure 1, the time from issue to execution is constant although our system needs as much as 80ms to execute real-time tasks. In wireless sensor nodes, since radio modules require the highest power, refraining from utilizing wireless communication per a unit leads to lower power consumption. Our system enables radio to be turned on periodically, which results in realization of power saving. Figure 3 shows that our system is comparable to Contiki in power consumption in spite of making Contiki preemptible.

In standard multithreaded systems, as each task has each priority, both interruption and context switching are needed whenever tasks which have higher priority than an executed task wait for execution. In our system, since general tasks except real-time tasks are handled like event-driven systems, the number of both interruption and context switching can be reduced.

5. CONCLUSIONS

Under normal conditions, in our system as the event-driven system, event handlers associated with occurred events are executed. When real-time events occurred, a current task is interrupted, and execution of the interrupted task will be resumed as soon as execution of the real-time task is completed. As a result of experiments, our system enables real-time tasks to be executed within 82ms, which is about 25% of the execution latency of Contiki. Furthermore, latency is kept constant without drastically increasing in power consumption.

6. REFERENCES

[1] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[2] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM.

[3] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-rk: An energy-aware resource-centric rtos for sensor networks. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, RTSS '05, pages 256–265, Washington, DC, USA, 2005. IEEE Computer Society.

[4] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[5] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648. IEEE, 2006.