# Performance Evaluation of the Nano OS Kernel based on System State-Monitor for Ubiquitous Sensor Network

Dong Myung Lee[1] and Kwangyong Lee[2]
[1]Department of Computer Engineering, Tongmyong University
[2]Embedded OS Research Team, Electronics and Telecommunications Research Institute

[1]179, Sinseonno, Nam-Gu, Busan, 608-711, Republic of Korea
[1]Tel: +82-51-629-1176, Fax: +82-51-629-1129
[1]dmlee@tu.ac.kr, [2]kylee@etri.re.kr

***Abstract* – Not only the MCU stopping and the auto-reset problems but also the dead end transition problems in the Nano OS kernel of the sensor modules are analyzed in this paper. In order to avoid and control these problems, the stack-safe Nano OS kernel suitable for USN and the system state-monitor mechanism in the Nano OS are suggested, and the performance is evaluated by a number of experimentation. The evaluation results show that the reliability and stability enhancements can be adapted and achieved by implementing thread-driven Nano OS kernel based on system state-monitor for USN.**

## 1. INTRODUCTION

The distributed computing platform of Ubiquitous Sensor Network (USN) is consisted of many a few autonomous nodes and no reusable resources. The resources of the platform - computing performance, memory and battery - are extremely dependent on the platform execution environments. So, the specific operating system (OS) should be required for wireless sensor modules to run as a low power system and light-weighted equipment and to tolerate against running condition and application program environments.

The Microprocessor Control Unit (MCU) stopping problem in the sensor modules by stack overflow and auto-reset problem by power degradation in sensor modules are occasionally appeared due to extremely limited memory space and low price MCU having non-memory protection function.

In this paper, not only the MCU stopping and the auto-reset problems but also the dead end transition problems in the Nano OS kernel of the sensor modules are analyzed. In order to avoid and control these problems, the stack-safe Nano OS kernel suitable for USN and the system state-monitor mechanism in the Nano OS are suggested, and the performance is evaluated by a number of experimentation.

## 2. RELATED STUDIES AND ISSUES

The TinyOS[1] is a representative open source OS suited for wireless sensor networks, and supports many components libraries effective for software developments. The SOS[2] is developed for mote based wireless sensor network environment, and provides messaging, dynamic memory, loading and unloading of module services in the kernel. The Mantis[3] is an UNIX style OS that is suitable for software development under UNIX environment using the common Application Programming Interface (API) independent of detailed hardware specification of sensor network. The TinyOS and the Mantis have no memory protection function.

The Nano OS what is called Nano Qplus[4, 5] is the module based hierarchical structure of Linux style for module reconfiguration in build time. It supports the FIFO or the preemption based schedulers, the IEEE 802.15.4 MAC protocol and the path selective star-mesh routing schemes in USN. Furthermore it provides various device drivers for running sensor modules such as illumination/temperature/ humidity/gas/infrared/ultrasonic/relay switch.

Unfortunately, the Nano OS doesn't have the memory protection function; it cannot cope with stack overflow problem. In TinyOS, the stack overflow can not be checked and recovered in the program running state, and it can be only checked by software simulation. The commercial OSs using in USN are shown to Figure 1.

| | TinyOS | MANTIS OS | SOS | Nano Qplus |
|---|---|---|---|---|
| Multitasking | X | O | O | O |
| Realtime | X | O | X | O |
| Low power Mode | O | O | O | O |
| Optimization/ Flexibility | O | O | O | O |
| Developing Simplicity | O | O | O | O |
| Static Re-setup | O | O | O | O |
| Dynamic Re-setup | X | O | O | O |
| Standardization | X | X | X | O |
| Memory Protection | X | X | O | X |
| Execution Model | Event | Thread | Event | Thread |

Figure 1. Commercial OSs Using in USN.

In this paper, we have suggested two mechanisms; one is the stack-safe Nano OS kernel suitable for USN using the Nano QPlus 1.6 version (qplusn–1.6.x). The kernel is released at present time for running application program to run safely even though the stack overflow problem under program execution is occurred in sensor module. The other mechanism is the system state-monitor mechanism that recoveries the modules to normal state even though the main module of kernel is transited to dead end or deadlock state.

## 3. DESIGN OF STACK SAFE-NANO OS KERNEL BASED ON SYSTEM STATE-MONITOR FOR USN

### 3.1 Analysis of Nano OS Kernel

In order to define the problem in existing Nano OS kernel exactly, we have built USN flatform using the Nano-24 network that is consisted of router node, sink node, sensor

node (illumination/temperature/humidity/gas) manufactured by Octacomm company to test the star-mesh network program. The kernel of Nano OS (Nano Qplus 1.6.1e version, qplusn -1.6.1e.tgz) is adapted to all of the sensor nodes.

The sensing time period is set to 3 seconds and battery power is used to sink node, and AC power is used to the sensor nodes and the router nodes to keep the source of electric power safely. If the sink node receives packets from sensor node, the red LED is emitted lights. If the node recognizes assigned values of gas and intensity of illumination sensor as soon as it checks the packet, the green and yellow LEDs are emitted lights at the same time. Also the green LED is designed to emit per every seconds for indicating the liveness of MCU.

The analyzed and conformed problems in Nano OS by experimentation are the MCU stopping problem due to the stack overflow and the auto-reset problem in sink node due to energy degradation.

### 3.2 MCU Reset Scheme

The MCU should be run at all the software environments all the time except to physical hardware faults without any the MCU stopping problem in sensor node. To satisfy this condition, the MCU reset scheme in the ATMega128 with the Watch Dog Timer (WDT) is designed and it is enabled at the resource initiation step of main program.

As shown to Figure 2, the maximum waiting time from the MCU to the WDT is set to 2 seconds to send a MCU RESET_SIGNAL. The MCU can resets the WDT per every 1 second periodically by timer interrupt because the timer0 interrupt in Nano OS is designed to activate per every 1 second periodically.

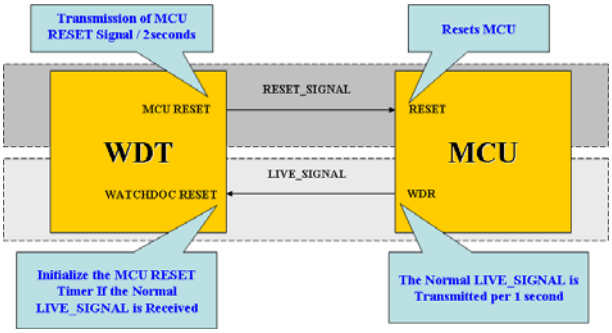


Figure 2. MCU Reset Scheme in ATMega 128.

If the MCU sends a WDR LIVE SIGNAL to the WDT per every 1 second, the WDT decides that the MCU is normal state and it waits for a WDR LIVE SIGNAL during 2 seconds. If the WDT doesn't receive a WDR LIVE SIGNAL, it sends a MCU RESET_SIGNAL to the MCU. Therefore this mechanism enables the MCU itself to be hardware based reset and to be restarted even though software faults are occurred.

### 3.3 Stack-Safe Nano OS Kernel

Interrupt service routines are divided into two groups such as the user level interrupt and the kernel level interrupt to maintain the safe of stack. Interrupt occurrence in the application (user level) thread stack which the memory size is smaller than that of the kernel thread causes to great probability of thread overflow or underflow problems in thread driven kernel, The kernel thread (main function), application threads and interrupt service routines run concurrently in thread driven kernel such as Nano OS environment. There are two important interrupt service routines in the Nano OS, one is the interrupt service routine for wireless communication service(SIGNAL(SIG_INTERRUPT0) and SIGNAL(SIG_OUTPUT_COMPARE 1A)) and the other is the timer based interrupt service routine related to thread wake-up in main scheduler(SIGNAL(SIG_OVERFLOW0)

The stack overflow problem is occurred when an interrupt service routine runs within the range of application thread stack in which the size is utmost 128 bytes. It is confirmed in our experimental analysis that the interrupt is occurred frequently and it causes the application thread stack to be increased continuously because the MAC scheduler interrupt service routine like the SIG_OUTPUT _COMPARE1A is run as low level interrupt mode. In worst case, the low level interrupt mode causes the stack overflow and results to program runtime errors.

To solve this problem, the stack-safe kernel algorithm is designed as shown to Figure 3. In this algorithm, the interrupt service routine can be run only within the main stack area in which the memory resource (size) are about 500 bytes.

```
1 SIGNAL(interrupt)
2 {
3   get SP;                    /* get stack pointer */
4   if (SP < (HEAP_TOP+32)) /* if thread stack area */
5     return;                  /* return;*/
6   else                       /* if SP within main stack area */
7     call ISR();              /* call interrupt service routine */
8 }
```

Figure 3. Stack-Safe Kernel Algorithm.

### 3.4 System State-Monitor Mechanism

The current Nano OS kernel is transited from normal state to dead end state and then it cannot be recovered to normal state completely. The system state-monitor mechanism is designed and applied to the Nano OS kernel to solve this severe problem. The dead end state in the Nano OS kernel can be generally occurred if the Zigbee RF module is come to wait for occurring RF interrupt during it receiving packets. It means that the sensor module is transited from normal state to dead end state, so it cannot receive packets during the sensor waiting for arriving and receiving packets by some problems. This causes quality (ratio of successful data communication) of the sensor node to be largely decreased even though it is in the liveness state and is exhausting its battery. Nevertheless we have suggested the solutions for the MCU stopping problem by stack overflow and the auto-reset problem by power degradation in sensor nodes above, the ratio of successful data communication of sensor node is largely decreased.

Thus, the system state-monitor mechanism is designed for solving this problem. In this mechanism, the state of main kernel module in Nano OS kernel is indicated to two types such as 'active state' to 1 and 'inactive state' to 0. The state of main kernel modules are [RX, T1, W, I].

- RX : Zigbee RF module for receiving sensing data
- T1 : Zigbee MAC scheduler module
- W : WakeUp module of Nano OS scheduler
- I : Idle thread module of Nano OS scheduler

The system state-monitor algorithm and macro procedure are shown to Figure 4 and 5 respectively. In this algorithm, the Zigbee RF module shall be ready for receiving sensing data, it sets the system state to 'active state' (1) and it shall informs the change of system state to the system monitor. If the above modules are not executed like the following error conditions since a given time period, the system-monitor decides the module to be transited from normal state to dead end state. The error conditions are :

- Err1 : (*, *, 0, 0) → Nano OS scheduler is not executed.
- Err2 : (0, *, *, *) → Zigbee RF module for receiving sensing data is not executed during a fixed time.
- Err3 : (*, 0, *, *) → Zigbee MAC scheduler module is not executed.

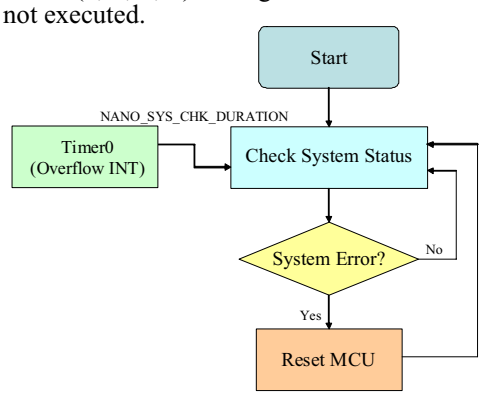


Figure 4. System State-Monitor Algorithm.

```
* sig_qplusn.h
SIGNAL( SIG_OVERFLOW0 ) /* Main timer interrupt */
{
  qplusn_system_status_checkpoint();
  tmr_ms += tmr_interval;        /*1000ms*/
  initialize SIG_OVERFLOW0 interrupt;
}
```

```
* Timer.c
void qplusn_system_status_checkpoint(void)
{
  if(tmr_ms >= next_tmr_system_checkpoint)
  {
    error = Check for the system status value,
      NANO_SYSTEM_STATUS (RX,T1,W, I);
    if(error is in [Err1, Err2 or Err3])
      NANO_RESET_MCU(); /* Reset & Recovery system */
    NANO_SYSTEM_STATUS = 0x00;
      /* Clear system status flag */
    next_tmr_system_checkpoint =
      tmr_ms + NANO_SYS_CHK_DURATION (5sec);
  }
}
```

```
* Initialize
  next_tmr_system_checkpoint =
NANO_SYS_CHK_DURATION (5sec);
  tmr_ms = 0;
```

Figure 5. Macro Procedure of System State-Monitor.

The system state-monitor forces the MCU to reset if it meets above error conditions, and it reboots and executes the kernel from the beginning phase, the module can escape from the system faults status. At this time, the error conditions Err1 and Err2 are checked per 5 seconds in the Nano OS version 2.x (qplusn-2.x), and the error condition Err2 that is not receiving RX data is checked per 120 seconds. That is to say, the kernel forces the system to reset automatically in the cases that the timer module or Nano OS scheduler is not executed during 5 seconds or the module cannot receive RX data during 2 minutes.

## 4. PERFORMANCE EVALUATION

### 4.1 Test Kernel Cases and Evaluation

The experimental environment for developing the Nano OS kernel based on system state-monitor is the USN flatform using the Nano-24 network as explained in Chapter 3. The major evaluating issues are 1) the influence of battery consumption to the sensor operation; and 2) the stack usages of threads according to kernel running time.

The experimentation data was acquired until two adapted AA batteries were vanished completely (about 2.3 voltage). The proposed mechanisms are evaluated with three test kernel cases that are shown to Table 1 and the performance evaluation metrics are shown to Table 2 respectively.

Table 1. Test Kernel Cases

| Case Type | Test Kernel Cases |
|---|---|
| Case 1 | the WDT based kernel |
| Case 2 | WDT + stack-safe based kernel |
| Case 3 | WDT + stack-safe based kernel + system state-monitor |

Table 2. Performance Evaluation Metrics

| Metric Types | Description |
|---|---|
| Metric 1 | Is the sensor node running continuously and can it be run even though adapted batteries are vanished completely? |
| Metric 2 | Are the stack of running threads maintained without any stack overflow or underflow problem completely? |
| Metric 3 | How is the performance results of the Nano OS kernel based on system state-monitor for USN? |

### 4.2 Performance Evaluation and Results

From the performance evaluation, we see that the battery usage (consumption) time for operating of normal sensor node is maintained about ten hours until the battery level is degraded from about 3.2 voltages to less than 2.7 voltages. Also the battery usage time is maintained about 20 hours until the battery level is degraded to less than 2.58 voltages. In addition to this, it is maintained about average 50~60 hours until the battery level is degraded to 2.3 voltages. The results are shown to Figure 6. As a result, it is evaluated that the sensor node is running normally until the battery is consumed completely, and it can not be stopped at all even though the system reset can be occurred in the sensor node.

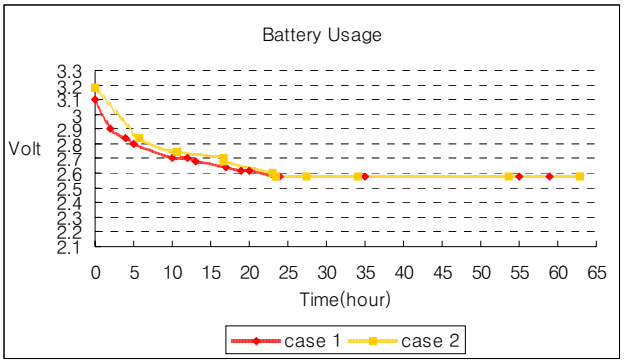


Figure 6. Battery Usage Ratio of Case 1 and Case 2.

In case 1 experimentation (WDT based kernel), the stack size is gradually decreased as time is proceed and as a result, the auto-reset event is occurred due to stack overflow as shown to Figure 7. In other words, as time is proceed, the stack is not used to threads no any more, thus the auto-reset by the WDT is occurred.

In case 2 experimentation (WDT + stack-safe based kernel), the stack usage ratio of threads is maintained constantly to less than 1 second by dividing the interrupts into the kernel level and use level interrupts as shown to Figure 8. This means that the stack is stabilized.

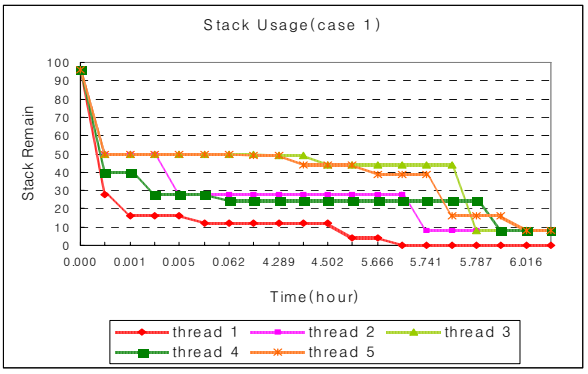


Figure 7. Stack Usage of threads (case 1).

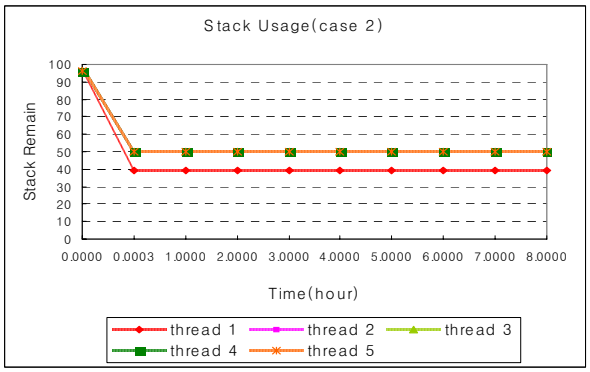


Figure 8. Stack Usage of threads (case 2).

The number of the WDT reset trials in case 1 (WDT based kernel) is increased because the stack overflow is occurred frequently in a short time as shown to Figure 9. But the number of the WDT reset trials in case 2 (WDT + stack-safe based kernel) is decreased to less than 1/3 compared with case 1. As a result, we see that the kernel of case 2 is more robust and stabilized than that of case 1.

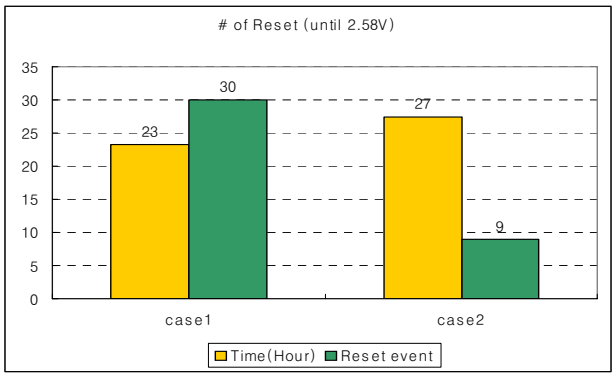


Figure 9. Numbers of WDT Reset Trials.

In case 3 experimentation (WDT + stack-safe based kernel + system state-monitor), we see that the ratio of successful received message in case 3 is over 75% similar to that of case 2 as shown to Figure 10. This means that the result shows the ratio of fully and adequate received message of sensor node in USN under low speed.

Increasing the ratio of successful received messages in USN with cooperative works to mostly 100% can be easily realized by applying the mechanism that makes full use of the duplicated sensor nodes. This works can be implemented using one of two test kernel cases (case 2 or case 3) because the ratio of successful received message in USN of two test cases are good equally.

Then it is inferred that case 3 must be adapted rather than case 2 in using the system state-monitor mechanism because the ratio of successful received packets in USN in case 2 is high and the ratio of received packets in case 2 is by internal kernel for maintaining links of USN. Therefore the result of test kernel case 3 is better than that of test kernel case 2 in comparing with the processing of the actual sensing data necessary to the user applications.

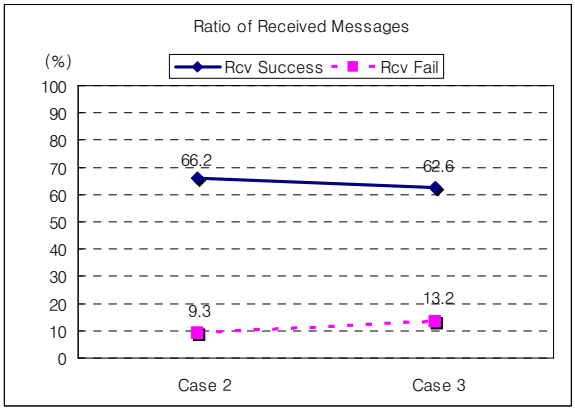


Figure 10. Ratio of Received Message in Case 2 and Case 3.

## 5. CONCLUSIONS

In this paper, not only the MCU stopping and the auto-reset problems but also the dead end transition problems in the Nano OS kernel of the sensor modules are analyzed. In order to avoid and control these problems, the stack-safe Nano OS kernel suitable for USN and the system state-monitor mechanism in the Nano OS are suggested, and the performance is evaluated by a number of experimentation.

The evaluation results show that the reliability and stability enhancements can be adapted and achieved by implementing thread-driven Nano OS kernel based on system state-monitor for USN.

## REFERENCES

[1] Levis P., Madden S., Gay D., Polastre J., Szewczyk R., Woo A., Brewer E., Culler D., "The emergence of networking abstractions and techniques in TinyOS," in First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)", 2004.

[2] Han C.C., Kumar R., Shea R., Kohler E., Srivastava M.B., "A dynamic operating system for sensor nodes," in MobiSys pp. 163-176, 2005.

[3] Bhatti S., Carlson J., Dai H., Deng J., Rose J., Sheth A., Shucker B., Gruenwald C., Torgerson A., Han R., "Mantis OS : An embedded multithreaded operating system for wireless micro sensor platforms," ACM Kluwer Mobile Networks and Applications (MONET) Journal, Special Issue on Wireless Sensor Networks, 2005.

[4] Kwangyong Lee, Youngsam Shin, Heeseok Choi, Seungmin Park, "A Design of Sensor Network system based on Scalable & Reconfigurable Nano-OS Platform," in Proceedings of the IT-SOC, Seoul, Korea, Oct. 2004.

[5] Dong Myung Lee and Kwangyong Lee,"A Development of the Nano OS Kernel based on System State-Monitor for Ubiquitous Sensor Network," Proceedings of the 10th International Conference on Advanced Communication Technology(ICACT 2008), pp.963~966, Feb. 2008.