INTERNATIONAL JOURNAL OF RESEARCH SCIENCE & MANAGEMENT

A LIGHTWEIGHT OPERATING SYSTEM FOR DYNAMIC WIRELESS APPLICATION DEVELOPMENT KIT TO INCREASE USABLE MEMORY

Rajesha N[1*], HL Viswanath[2]
[1*]Research Scholar, Department of ECE, Shri JJTU, Jhunjhunu-01
[2]Prof. and Head, Dept. of ECE, RRIT, Bangalore-90
Correspondence Author: rajeshmurthy44@gmail.com

Keywords: Flash File System, Wear Levelling, Scheduling, Remote reprogramming

Abstract

A replaceable scheduler is provided to support different scheduling policies that meet the requirements of different WSN applications. The efficiency of the scheduler is improved by fast algorithms instead of linear search for the next running thread; the overhead of context switch can also be reduced by storing the critical registers on the internal stack and avoiding using the slow external memory. The proposed approach for overcoming resource constraints is compiler-assisted run-time support for features such as virtual memory, remote reprogramming and memory allocation. Virtual memory can be achieved without hardware support via code insertion at compile time. Position-independent code (PIC) is a suitable approach to achieving dynamic loading; the compiler can assist with the generation of PIC on the host PC, thus avoiding the overhead of relocating addresses at run-time. With the host PC's help, the run-time overhead on the wireless sensor nodes can be reduced significantly. Flash File System has been developed to achieve efficient file operation and low power consumption. To minimize the active time of the MMC card by I/O scheduling and by limiting the access via the API is provided by Flash File System. To reduce code size, it is proposed to let the controller can handle wear levelling and erase-before-write.

Introduction

Real Time Operating System (RTOS) is a limited constrained platform for wireless sensor networks (WSN). RTOS constitutes a cooperative threading model, which is applicable to WSN applications. Virtual memory is supported with the assistance of the compiler, so that the sensor platforms can execute code larger than the physical code memory they have. To enable firmware update for deployed sensor nodes, RTOS supports remote reprogramming. The commonly used library and the main logical structure are separated; each sensor device has a copy of the dynamic loading library in the Micro-SD (Secure Digital) card, and therefore only the main function and user-defined subroutine scan is updated through RF. A lightweight, efficient Flash file system is also included in RTOS. The designed RTOS can run on many RF-enabled systems that cannot run most other WSN OSs.

Sensor Network nodes are least expensive, low power embedded systems. Random access memory (RAM) is tightly-constrained in many embedded systems. The sensor network nodes have only 1-10KB of RAM and do not contain memory management units (MMUs). It is very difficult to implement complex applications under such limited memory constraints. It is proposed to use of compile-time and run-time techniques to increase the amount of usable memory in MMU-less embedded systems and to increase the size of the flash memory. The proposed techniques do not increase hardware cost, and are designed to require small changes to existing applications. A fast compression algorithm has been developed, that is well suited to this application, as well as runtime library routines and compiler transformations to control and optimize the automatic migration of application data between compressed and uncompressed memory regions. These techniques were experimentally evaluated on sensor network nodes running a number of data collection and signal processing applications. The results indicate that available memory can be increased by up to 26-30%.

Embedded systems are primary constraints in applications of wireless sensor networks. In these systems, processing and flash memory are limited due to constraints on cost, size, and power consumption. For example, eight-bit microcontrollers generally have no memory management units (MMUs). Sensor network nodes are the embedded systems whose resource and power are limited, and hence constrained [2]. Even though the proposed techniques may be used in any memory constrained embedded system without an MMU, this article will focus on using them to increase flash memory in sensor network nodes with limited changes to hardware and with no or minimal changes to applications. With advances in technology, sensor networks composed of small and cost effective sensing devices equipped with wireless radio transceiver for environment monitoring are deployed. The key advantage of using these small devices to monitor the environment is that it does not require infrastructure such as electric mains for power supply, wired lines for Internet connections to collect data, no need for human interaction while deploying. These sensor nodes can monitor the environment by collecting information from their surroundings and work cooperatively to send the data to a base station, or sink, for analysis. Many ideas utilized for improving the communication, security, and in-network processing capabilities of sensor networks rely on sophisticated routing encryption, query processing and signal processing algorithms are implemented on sensor network nodes. However, sensor network nodes have limited memory constraints. Unfortunately, it is not economical to fabricate the deep trench capacitors used for high-density RAM with the same process as processor logic. As a result, SRAM is used in sensor network nodes. Unlike DRAM, SRAM generally requires six transistors per bit and it has more power consumption. Some researchers have proposed addressing memory constraints using hardware techniques such as compression units inserted between memory and processor. However, such hardware implementations typically have difficulty adapting to the characteristics of different application data.

Software techniques that use data compression to increase usable memory have some advantages over hardware techniques. They do not require processor or printed circuit board for design and they allow the selection and/or modification of compression algorithms, permitting good performance and compression ratio for the specific applications. However, software techniques that require the design of applications are to be used by embedded systems programming experts. Most sensor network application experts are not embedded system programming experts. If memory expansion technologies are to be widely deployed, they couldn't require changes to hardware and could require minimum or no changes to applications. In this article, it is proposed to use a memory expansion technique, named MEMMU, for use in wireless sensor networks. This technique uses compile time transformation and run-time library support to automatically manage on-line migration of data between compressed and uncompressed memory regions in sensor network nodes. It provides application developers with access to more usable RAM and requires minor changes to application code and limited changes to hardware. The proposed technique requires no MMU and has enabling its use in sensor network nodes with extremely limited memory and performance constraints. It has been optimized to minimize impact on performance and power consumption; experimental results indicate that in many applications, such as data sampling and audio signal correlation computation, its operating cost is small [4].

The rest of this paper is organized as follows. Section 2 discusses Architecture of Flash File System. Section 3 presents Data Types in Flash File System concepts. Section 4 describes the Design Overview of Remote System Call. Section 5 describes the Summary. Section 6 presents the Experimental Setup. Finally, Section 7 concludes this paper.

The flash-file system

A dynamic and efficient storage system is a basic constraintto build ultra-compact sensor nodes for asynchronous transmission. Nonvolatile storage system is act as a significant role for recording node status and timestamps of events, especially since power depletion may occur unpredictably on a deployed sensor node. However, it is possible to write codes for drivers to directly control the nonvolatile storage. Amore structured glide path is to build a simple file system generalization on top of the raw storage device. It is proposed to use a file system named Flash File as a component in RTOS for Underwater Environment [11].

Flash file architecture overview

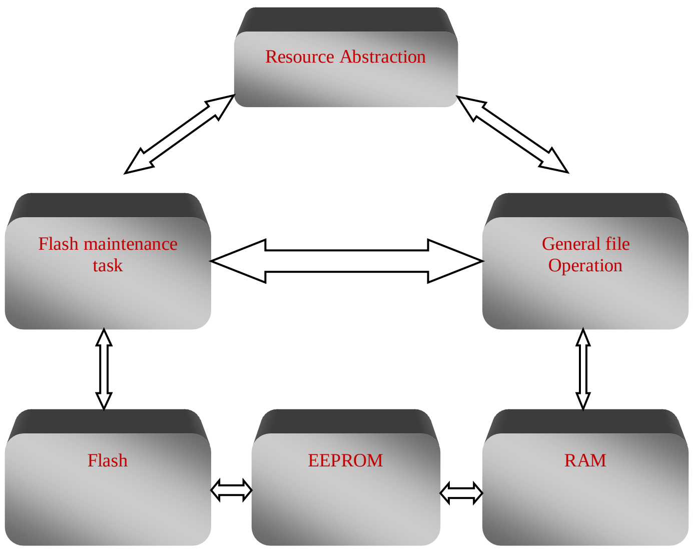


Figure 1: Flash File System Architecture

Figure 1 illustrates the architecture of the FLASH FILE system. At the top is the "Resource Abstraction" maintained in run-time memory. It consists of the in-memory representations of open files, cleaning policy, and other FLASH FILE configuration data. The "General File Operations" is the logical abstraction of all file/directory operations. The "FLASH FILE Maintenance Tasks" is the abstraction of system maintenance tasks, *e.g.* maintaining a snapshot of directory structure and file metadata in EEPROM. It also includes the garbage collection task. When predetermined conditions are met, the garbage collection task will be posted, though these functions can also be explicitly called by the users. The cleaning policy is selected at compile time.

However, the command graph can be used to analyze sensed data and plot charts. To demonstrate the convenient shell environment of Flash File host side implementation, figure 6 shows the snap shots of Flash File Shell. The users can easily manipulate the Flash files by plugging the MMC card into a card reader via the assistance of Flash File Shell on the host PC. The collected data by the sensors can be analyzed easily as shown in figure 6. In addition, all Flash File System-formatted data, including binary code segments, can be accessed via Flash File Shell [12].

Storage Medium in WSN

In recent years, flash memory has been widely used in embeddedsystems and handhelddevices. The features of flash memory include non-volatility, smaller size, low power consumption, and shock resistance. In fact, flash memory can be established on many wireless sensor platforms either on-board or as an expansion module. Experimental result shows that platform can have an external MMC card module connected via SPI. A MMC card is functionally identical to a regular SD (Secure Digital) card. Figure 2 shows the User program for the developed module.

An MMC card contains a controller that handles wear leveling, auto-erasing, and error correcting codes (ECC) recovering.MMC/MMC cards can be controlled through a serial peripheral interface (SPI). Although MMC cards consume more energy than flash components because of their simple interface, small physical size, and high capacity characteristics make them suitable for wireless embedded devices

```
public Packet Transmit() { if ((sSender.iResidualEnergy <= 0) || ((sReceiver != null) &&
(sReceiver.iResidualEnergy <= 0)))
{        if (sSender.iResidualEnergy <= 0)
{     if (sSender.Pherovalue > 0)
sSender.Pherovalue -= 0.1m;
}
else if (sReceiver.iResidualEnergy <= 0)
{ if (sReceiver.Pherovalue > 0)
sReceiver.Pherovalue -= 0.1m;
}
iTransmitting = 0;
}   else if (iTransmitting > 0)
{  iTransmitting--;
if (iTransmitting == 0)
{ if (sReceiver != null)
{      if (sReceiver.Pherovalue < 100)
sReceiver.Pherovalue += .1m;
} if(sSender.Pherovalue<100)
sSender.Pherovalue += .1m;
if (sReceiver != null)
sReceiver.aPackets.Add(packet);
else
{  Packet returnPacket = packet;
packet = null;
return returnPacket;
} }}
return null;
}}
```

Figure 2: User program

Host side vs. host side flash file

The implementation of Flash File consists of two main parts: node side and host side. The part on the host side is controlledusing the Flash File Shell. Due to the resource limitations, the node side focuses on how to efficiently accessFlash File data. On the other hand, the host side provides completefunctionalities of Flash File, including list, read, write, modify, and binary to HEX translation. There is no severerestriction on the host PC.

Node side

For the wireless sensor node implementation shown in Fig. 7, an MMC/MMC driver is built according to the Secure Digital Card specification that uses SPI protocol to accessthe memory card for basic I/O operations. A Flash File libraryis too provided in order to distinguish specialized data types, namely code data, preferences, sensed data, andnetwork data. The Flash File Library is configurable: only demandedones are configured and installed on the wirelesssensor nodes. Some reference applications commonly usedin WSN such as logging sensed data, booting from an MMCcard, periodically refreshing the node status, and accessingthe routing table in multi-hop wireless networks by using Flash File System.

Host side

After the data has been stored in the MMC Card, it is relativelyeasy to access the MMC card via a USB card reader on ahost PC as shown in Fig. 7. To extract Flash File-formatteddata, a Flash Fileprogram is required. The data read fromthe card are processed by the program that converts it intothe appropriate file format to be accessed by the user.The top layer of Flash File host-side implementation isFlash File Shell. The interactive shell environment providesgeneral Unix-like commands such as ls, cd, cp, and cat sothat users can list all files for each type and see the content in eachfiles. There are also special commands, for example, mkfs to format an MMC card, clean to eliminate all files, and installto modify and add files to Flash File System.

Data types in flash file system

Each data item in Flash File System is eitherfixed length or wrapped by special tags as used in a regularTCP/IP packet format and has a particular length field tosuggest the size of the item. Therefore, the analyzing processcan be done without consuming large data memory by usingload-partial-then-analyze scheme. In the following subsections, we describe the detailed format of Flash File data types [18].

Code data

The intention of code data is to affirm virtual memory for code in RTOS. By using the Flash File Shell, the dynamic loading library ELIB can be pre-installed on the MMC card. Each code segment has a unique virtual address, which is the location of the code segment onthe MMC card. The segment format starts from a special BEG (0xAE) tag followed by a two-byte field indicating the binary segment size; after the size field, the binary databytes with a pre-defined size will follow. When a node wants toretrieve the segment by its virtual address from the MMC card, it first checks for the BEG tag; then it reads two bytesto get the data size and allocate suitable code memory space; finally, the code segment is loaded from the MMC card into the code memory. The current design of the code datablock does not allow modification of the code segment bythe sensor node itself, though a later version of Flash File willenable replacement of binary segments at run-time.

Preferences

To provide a fast query data structure of Flash File, it is possible to add the preference data type. Each preference item is represented by 22 bytes, including 1 byte BEG(0xEA) tag, 1byte TYPE tag that indicates the data type of the value field such as character, integer, or string, along with 10-bytekey string and the 10-byte value with a type tag. To prevent high data memory consumption and to support modification of data, each preference item is distributed into a 512-byte MMC card sector. The characteristic of preference is to support fast searching and additional modifying ability. As a result, a hashing scheme is applied. When the specific value of a key is required, first the key is passed to a simple hash function to generate an integer. Second, the integer is added to an offset to get the sector number, the location of preference item. Finally, the value is retrieved. For the keys with same hash value, it is possible to allocate five slots to store the collision preferences; the next linked preference is located next to the current located sector. There is also a super block for data of preference type, which is the bitmap used to check whether the target hash value exists or not. Thus, the string-comparison time is reduced for non-existent preference keys.

Sensed data

The sensed data is supplemented only when changing the “already sensed data” is unnecessary and the modification of flash memory results in high overhead. Some of the WSN sensing tasks collect sensed data when specific events have happened. For this reason, begin and end tags are added to enclose sensed data to keep events separated. There is also a timestamp field in sensed data format, so that for the later analysis can be calculate the event-trigger time using the data retrieved from this specific field. To fill the timestamp field, either the system timer or user application granted time can be used. Besides, if the MMC card is full, then the decision to overwrite or stop depends on the preference setting.

Network data

Flash File network data are used to store network-related information such as packet routes, neighbor sensor node states, and the WSN topology. The fields can be customized by user applications. Each network data item is represented by fixed 32-byte data, including 2 bytes of network ID and 30bytes of a user-defined structure such as the type, state, and remaining power of sensor node, depending on the requirements of the WSN applications. A bitmap is maintained in the super block for the purpose of quickly enumerating existing items in the Flash File network data area. The current version of Flash File supports around 50,000 IDs, which consumes 8192 bytes. Even though a sensor node reads 8192 bytes of result in about 12 ms, this mechanism consumes only 32 bytes of data memory, which is more beneficial especially for constrained wireless sensor platforms.

Implementation of remote system call

Figure 3 shows the structure and the behavior of the DRTOS for a remote system call. A remote system call is issued using an OSEK OS API specifying the global task ID of the target task. When an application task issues a system call, the task location search module determines the node on which the target task resides referring to the task location data. If the target task is a local task, the original system call of OSEK OS is executed. If the target task is a remote task, the remote system call module executes the processing for the remote system call as follows.

The request transmission submodule generates a request message, which consists of the global task ID of the caller task, the global task ID of the target task, the system call identifier and the input parameters. Then the submodule calls the Zigbee driver to write the request message in the message RAM of the Zigbee controller. The communication of request messages of system calls is executed by Zigbee controllers. Received request messages are stored in the message RAM of the Zigbee controller of the callee node. The cycle start processing is executed by an ISR (Interrupt Service Routine), which is activated at the beginning of each Zigbee communication cycle.

The ISR belongs to the category 2 of OSEK OS. The cycle start processing executes global time maintenance at first, and then executes the request processing submodule. The submodule calls the Zigbee driver to read messages received in the previous communication cycle, interprets the request message, and calls the original system calls of OSEK OS. The ISR executes the return value transmission submodule after the system call execution. The submodule generates a return message, which consists of the global task ID of the caller task, the return value and output parameters, and then calls the Zigbee driver to write the return message in the message RAM.

The ISR of the caller node executes the return processing submodule after the cycle start processing. The submodule calls the Zigbee driver to read messages and stores the return value and output parameters in a buffer. Then the submodule releases the caller task from the waiting state. The caller task gets the return value and output parameters from the buffer and resumes the application program [16].

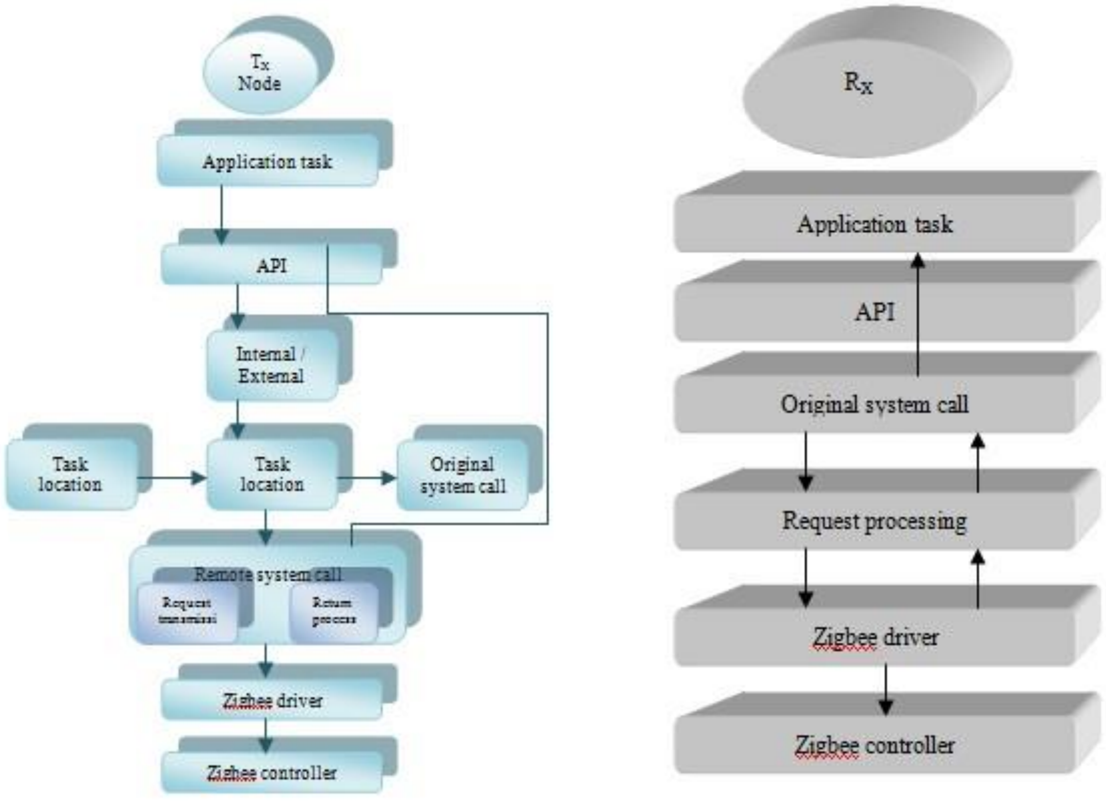


Figure 3: Structure and Behavior of DRTOS (Distributed Real Time Operating System) for Remote System Call.

Summary

Figure 4 illustrates the procedure for automatically generating an executable file from mid/high-level language source code such as C# with .NET platform.

Performance requirements

1. **Increased admin security:** The PC should be highly secured and accessible only by the administrator to avoid the misuse of the application.
2. **Portability:** The GUIs of this application is user-friendly so it is very easy for the user to understand and respond to the same.
3. **Reliability:** This system has high probability to deliver us the required queries and the functionalities available in the application.
4. **Response time**: The time taken by the system to complete a task given by the user is found to be very less.
5. Scalability: The system can be extended to integrate the modifications done in the present application to improve the quality of the product. This is meant for the future works that is to be done on the application.

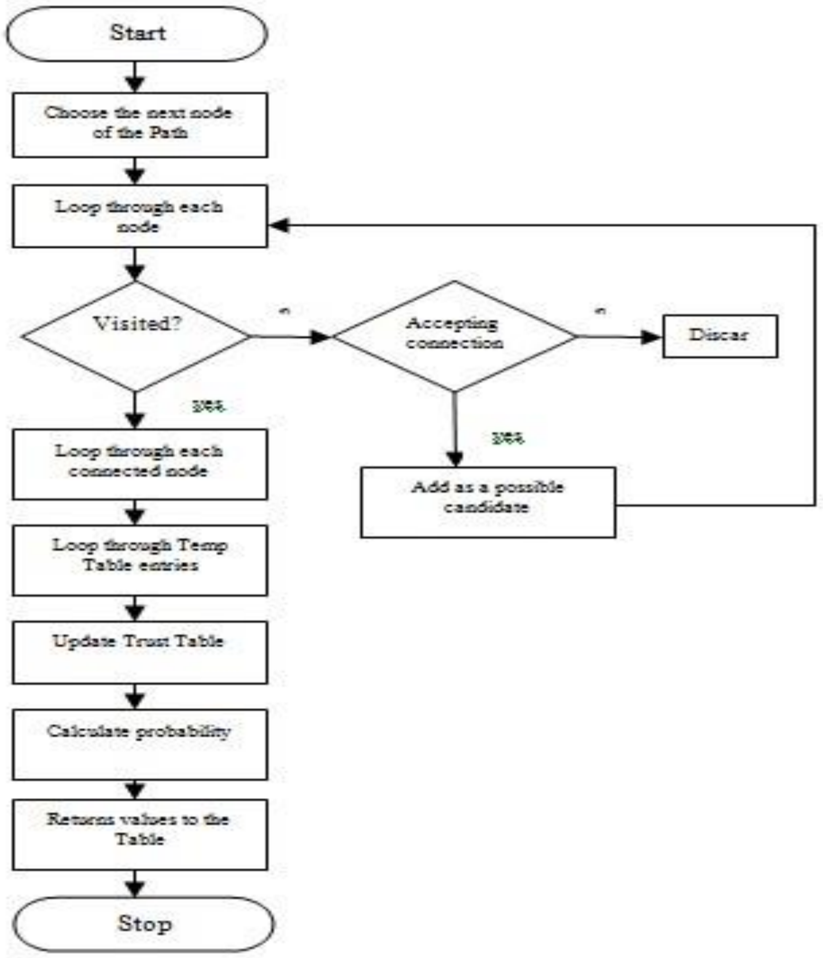

Start
Choose the next node of the Path
Loop through each node
Visited?
Accepting connection
Discar
yes
yes
Add as a possible candidate
Loop through each connected node
Loop through Temp Table entries
Update Trust Table
Calculate probability
Returns values to the Table
Stop

Figure 4: procedure for automatically generating an executable file from mid/high-level language source code

Evalution & results

This section shows the efficiency of the proposed module as a lightweightdynamic multithreaded OS for sensor network platform.
Issue:
While application debugging some time Visual Studio2008 have some sort of configuration problem. Deploy started: Project: Hello1, Configuration: Debug Intel_SDK. The configuration data for this product is corrupt. Contact your support personnel. Deploy: 0 succeeded, 1 failed, 0 skipped Solution:
This problem occurs, when Visual Studio 2008 Beta and all the beta files were not removed prior to installing the MSDN library version of VS 2008. The solution to this problem is as follows:

1. Shutdown Visual Studio 2008
2. Delete all the files in C:\Documents and Settings\<user name>\local settings\application data\Microsoft\CoreCon\1.0". (May want to create a backup just in case)
3. Restart Visual Studio 2008

If that does not solve the problem, you may need to completely uninstall Visual Studio 2008 and clean up the stuffs left by pre-RTM version probably with the help of the following tool: Visual Studio 2008 previous beta removal tool.

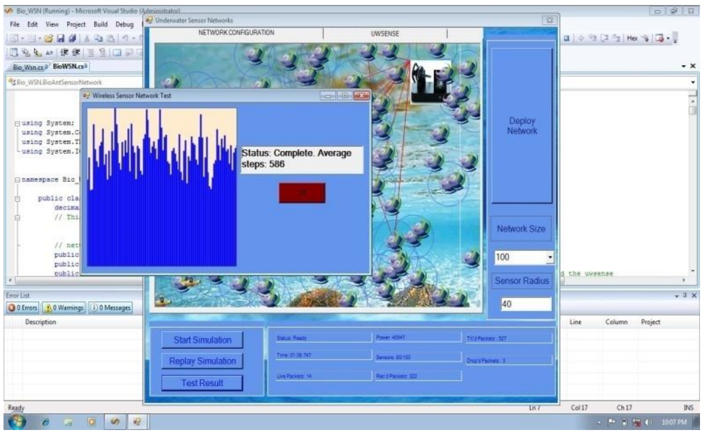

Bio_WSN (Running) - Microsoft Visual Studio (Administrator)
Underwater Sensor Networks
NETWORK CONFIGURATION
UWSENSE
Wireless Sensor Network Test
Status: Complete. Average steps: 586
Deploy Network
Network Size
100
Sensor Radius
40
Start Simulation
Replay Simulation
Test Result
0 Errors
0 Warnings
0 Messages

Figure 5: Utility tool for uploading and RF debugging

Initially, describing the experimental setup, including the hardware platform and software tools. Second, comparisonof the context switch overhead and the code anddata memory footprints of different multi-threaded schedulerimplementations have been demonstrated. It is also compare the powerconsumption of loading code segments over RF andMMC cards and evaluating the code updating scheme againstother methods to shown that, it is possible to reduce runtime overhead significantlyin terms of the size of the uploaded code [17].

Experimental Setup

This section describes the hardware platform and softwaretools for the proposed module.

Software tools

The software for the proposed module includes an IDE and graphical user interface tools (GUI) on the host computer, system software on the sensor node and the base station, and utility tools for uploading and RF debugging. This is as shown in figure 5. It is possible to build an IDE with Eclipse by creating a plugin for module development. With this plugin, a fully GUI-based programming environment has been provided. This lowers the burden for users to memorize commands and enables them to focus on the software development process, from editing, compiling, and linking to firmware updating.

Context switch overhead of different schedulers

To evaluate the context-switch overhead of cooperative threads in this proposed module, implementation of both round-robin (RR)and priority-based schedulers has been developed, along with different algorithms have been proposed, that may affect the context-switching time. It is going to measure the execution time by taking the average of 50,000 context switches. It is clear to observe that fast algorithms significantly improve both RR and priority-based schedulers by cutting the execution times down to a quarter of the original linear search implementation. Preemptive multi-threading has the highest context-switch overhead due to the unpredictable preemption time, and therefore all of the registers must be saved and restored during context switch. C-coroutines using C-switch statements have been implemented and adding the priority based and RR schedulers to it. C-coroutines and cooperative threads have the feature that context switching occurs only when the running thread calls yield or sleep functions, and thus they have lower context-switch overhead. The implementation of C-coroutines uses C-switch statements, this means that every context switch results in several comparisons of variables and an absolute jump. Consequently, a context switch of cooperative threads is simply a replacement of the program counter, stack pointer, and some global variables, and thus it has the lowest overhead.
To compare Proposed Module with a real-world OS, we ported µC/OSII to Proposed Module. It is widely used in industry, whereas no other WSN OSs are known to run on the 8051. Table 1 compares the context-switch overhead of µC/OS-II and our work. The reason why µC/OS-II has high context-switch overhead is that it uses external memory to store both the per-thread stack and registers, thus incurring great overhead from many external memory movements. Table 2 shows a comparison of the code and data memory usage between different scheduler implementations. The preemptive ones consume the most memory for the same reason mentioned before. Other scheduler implementations require about 1KB of code memory, which is frugal compared to other regular RTOSs such as µC/OS-II and FreeRTOS.

Table 1. Context switch overhead comparison between the proposed module and μC/OS-II for 50,000 context switches

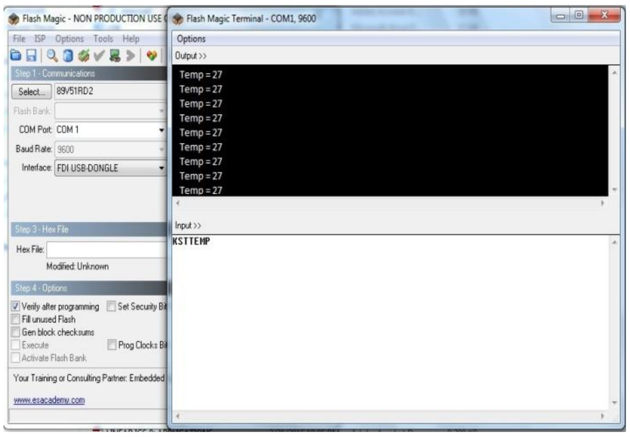
	Proposed Module	**μC/OS-II**
Overhead(μs)	23.3	159

Table 2. Code and Data Size Comparison. Unit: bytes

	Proposed Module	FreeRTOS	μc/OS II
Code Size	8234	7560	10294
Data Size	646	719	488

Sensor platform

Experimental platform is as shown in Fig. 7. The module refers to the larger version for prototype and application development, and a compact version called the "simple node" is more suitable for field deployment. This platform consists of an MCU with a Zigbee Module, power circuitry, and an expansion interface.


Flash Magic - NON PRODUCTION USE
Flash Magic Terminal - COM1, 9600

Step 1 - Communications
Select... 89V51RD2
Flash Bank:
COM Port: COM 1
Baud Rate: 9600
Interface: FDI USB-DONGLE

Step 3 - Hex File
Hex File:
Modified: Unknown

Step 4 - Options
[✓] Verify after programming
[ ] Set Security Bi
[ ] Fill unused Flash
[ ] Gen block checksums
[ ] Execute
[ ] Prog Clocks Bi
[ ] Activate Flash Bank

Your Training or Consulting Partner: Embedded
www.esacademy.com

Output >>

```
Temp = 27
Temp = 27
Temp = 27
Temp = 27
Temp = 27
Temp = 27
Temp = 27
Temp = 27
Temp = 27
```

Input >>

```
KSTTEMP
```

Figure 6.Flash File host PC shell environment with built-in data analysis and data viewers

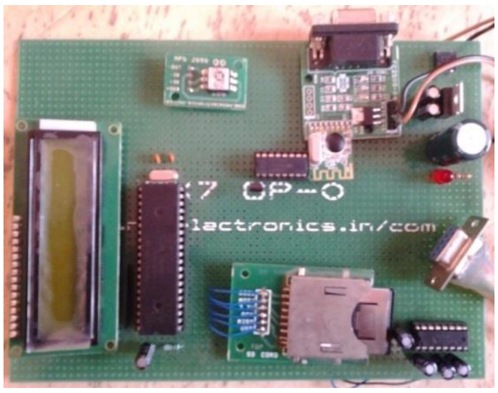


Figure 7: The proposed module.

The MCU (P89V51RD2) core runs at 16 MHz by default and comes with 16KB program flash and 1KB data RAM. A 32-Mbit on-board flash memory and MMC expansion capability are included for nonvolatile data storage. Most WSN OS cannot fit intothis limited platform and are thus not easily comparable. To measure the energy consumption in this module, it is possible to develop a power measurement module as shown in figure 7.

Conclusion

RTOS leads maximum contributions in the WSN OS area. The cooperative threads programming model enhances the performance by decreasing context-switching, which makes it two times faster than the preemptive multithreaded programming model. This cooperative threading model is easy to learn compared to the event-driven model. RTOS provides code virtual memory is overcome to reduce the on-chip code memory. To achieve virtual memory, the Flash File system is built on the host PC composed of PIC segments and is loaded to code memory on-demand by the run-time loader of RTOS. Remote reprogramming is

also available in RTOS. Due to the pre-stored Flash File on the Micro-SD card, the size of the data of the user program is to be wirelessly updated and it is reduced significantly during the remote reprogramming stage.
It is going to be described that the MEMMU, is an efficient software-based technique to increase Flash memory in MMU-less embedded systems. A number of compile-time and run-time optimizations are used to increase the performance and power consumption of the sensor nodes. An efficient delta-based Deflate compression algorithm was designed for sensor data compression. MEMMU was evaluated using a number of representative wireless sensor network applications. Experimental results show that the optimization technique effectively improves MCU's performance.
Finally, Besides, a shell for the host PC is also provided to control Flash File-formatted SD cards, including listing, reading, writing, and modification, thus reducing the difficulty and complexity to access Flash File data. Only ten percent of code is machine-dependent, while the rest is written in C language, and thus it is easy to port to other wireless sensor platforms. The fact that it runs on a modest sized P89V51RD2 MCU, means that RTOS is expected to be easily portable to many other integrated RF-MCUs, most of which contain an 8051-compatible core. Thus, RTOS is opening up a whole class of cost-effective, compelling RF-MCUs previously unsupported by WSN OSs.

REFERENCES

1. D. Aldous and J. Fill. Reversible Markov Chains and Random Walks on Graphs. Monograph in preparation, 2006.
2. I. Aydin and C. Shen. Facilitating Match-Making Service in Ad hoc and Sensor Networks Using Pseudo Quorum. In ICCCN, Oct. 2002.
3. O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. ACM Trans. Auton. Adapt. Syst., 1(1):26–66, 2006.
4. L. Badia, M. Mastrogiovanni, C. Petrioli, S. Stefanakos, and M. Zorzi. An Optimization Framework for Joint Sensor Deployment, Link Scheduling and Routing in Underwater Sensor Networks. In WUWNet'06, Los Angeles, CA, Sep. 2006.
5. S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In MOBICOM' 98, Dallas, TX, 1998.
6. G. D. Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. Artificial Life, 5(2):137–172, 1999.
7. G. D. Caro, F. Ducatelle, and L. M. Gambardella. AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks. Telecommunications (ETT), Special Issue on Self Organization in Mobile Networking, 16(2):137–172, 2005.
8. A. Caruso, F. Paparella, L. F. M. Vieira, M. Erol, and M. Gerla. The Meandering Current Model and its Application to Underwater Sensor Networks. In INFOCOM'08, Phoenix, AZ, Apr. 2008.
9. S. M. Das, H. Pucha, and Y. C. Hu. On the Scalability of Rendezvousbased Location Services for Geographic Wireless Ad Hoc Routing. Elsevier Computer Networks, 51(13):3693–3714, 2007.
10. F. Dressler. Benefits of Bio-inspired Technologies for Networked Embedded Systems: An Overview. In Dagstuhl Seminar 06031 on Organic Computing - Controlled Emergence, Dagstuhl, Germany, Jan.
11. BAI, L. S., YANG, L., AND DICK, R. P. Automated compile-time and run-time techniques to increase usable memory in MMU-less embedded systems. In CASES '06: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (New York, NY, USA, 2006), ACM, pp. 125–135.
12. BAI, L. S., YANG, L., AND DICK, R. P. MEMMU: Memory expansion for MMU-less embedded systems. ACM Trans. Embed. Comput. Syst. 8, 3 (2009), 1–33.
13. BHATTI, S., CARLSON, J., DAI, H., DENG, J., ROSE, J., SHETH, A., SHUCKER, B., GRUENWALD, C., TORGERSON, A., AND HAN, R. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. Mob. Netw. Appl. 10, 4 (2005), 563–579.
14. CAO, Q., ABDELZAHER, T., STANKOVIC, J., AND HE, T. The LiteOS operating system: Towards Unix-Like abstractions for wireless sensor networks. In IPSN '08 (Washington, DC, USA, 2008), IEEE Computer Society, pp. 233–244.
15. CHA, H., CHOI, S., JUNG, I., KIM, H., SHIN, H., YOO, J., AND YOON, C. RETOS: resilient, expandable, and threaded operating system for wireless sensor networks. In IPSN '07 (New York, NY, USA, 2007), ACM, pp. 148–157.
16. CHEN, C., CHEN, Y., TU, Y., YANG, S., AND CHOU, P. EcoSpire: an application development kit for an Ultra-Compact wireless sensing system. Embedded Systems Letters, IEEE 1, 3 (2009), 65–68.
17. CHOUDHURI, S., AND GIVARGIS, T. Software virtual memory management for MMU-less embedded systems. Tech. rep., Center for Embedded Computer Systems, University of California, Irvine, Nov 2005.
18. DAI, H., NEUFELD, M., AND HAN, R. ELF: an efficient log structured flash file system for micro sensor nodes. In SenSys '04 (New York, NY, USA, 2004), ACM, pp. 176–187.
19. DUFFY, C., ROEDIG, U., HERBERT, J., AND SREENAN, C. J. Adding preemption to TinyOS. In EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors (New York, NY, USA,

2007), ACM, pp. 88–92DUNKELS, A., FINNE, N., ERIKSSON, J., AND VOIGT, T. Runtime dynamic linking for reprogramming wireless sensor networks. In SenSys ’06 (New York, NY, USA, 2006), ACM, pp. 15–28.

20. DUNKELS, A., GRONVALL, B., AND VOIGT, T. Contiki – a lightweight and flexible operating system for tiny networked sensors. In LCN ’04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (Washington, DC, USA, 2004), IEEE Computer Society, pp. 455–462.
21. DUNKELS, A., SCHMIDT, O., VOIGT, T., AND ALI, M. Protothreads: simplifying event-driven programming of memory constrained embedded systems. In SenSys ’06 (New York, NY, USA, 2006), ACM, pp. 29–42.
22. EMITT SOLUTIONS. Microcontroller market and technology analysis report – 2008, 2008.
23. GAY, D., LEVIS, P., VON BEHREN, R., WELSH, M., BREWER, E., AND CULLER, D. The nesC language: A holistic approach to networked embedded systems. SIGPLAN Not. 38, 5 (2003), 1–11.
24. GU, L., AND STANKOVIC, J. A. t-kernel: providing reliable os support to wireless sensor networks. In SenSys ’06 (New York, NY, USA, 2006), ACM, pp. 1–14.