International Journal of Research (IJR) Vol-1, Issue-7, August 2014 **ISSN 2348-6848**

Operating Systems for Multitasking Sensor Networks: A Survey

Pavitha N[1]*

[1] *Department of Computer Engineering, Sinhgad Academy of Engineering, Pune,Maharashtra, India*
Email: pavithanrai@gmail.com

Abstract:

Development of multitasking operating systems for sensor networks and other low-power electronic devices is motivated by the networked application environment. These operating system's multitasking capability is severely limited because traditional stack management techniques perform poorly on small-memory systems without virtual memory support. This paper formalizes different multitasking operating systems like Mat'e, Contiki, MANTIS OS, t-kernel, RETOS, LiteOS, TOSThreads, Sensmart. This paper analyses all these operating systems for their performance in multitasking parameters. Under multitasking parameters out of these operating systems Sensmart performs best.

Keywords:

Multitasking, Sensor Network, Operating System, Stack Management

INTRODUCTION

The wireless sensor nodes are spatially distributed over a region of interest and observe physical changes such as those in sound, temperature, pressure, or seismic vibrations. If a specific event occurs in a region of distributed sensors, each sensor makes local observations of the physical phenomenon as the result of this event taking place. An example of sensor network applications is area monitoring to detect forest fires. A network of sensor nodes can be installed in a forest to detect when a fire breaks out. The nodes can be equipped with sensors to measure temperature, humidity, and the gases produced by fires in trees or vegetation. Other examples include military and security applications. Military applications vary from monitoring soldiers in the field, to tracking vehicles or enemy movement. Sensors attached to soldiers, vehicles and equipment can gather information about their condition and location to help planning activities on the battlefield. Seismic, acoustic and video sensors can be deployed to monitor critical terrain and approach routes; reconnaissance of enemy terrain and forces can be carried out.

After sensors observe an event taking place in a distributed region, they convert the sensed information into a digital signal and transmit the digitized signal to the Fusion Centre. Finally, the Fusion Centre assembles the data transmitted by all the sensors and decodes the original information. The decoded information at the FC provides a global picture of events occurring in the region of interest. Therefore, we assume that the objective of the sensor network is to determine accurately and rapidly reconstruct transmitted information and reconstruct the original signal.

Background and Basics:

The growing popularity of low-power and pervasive wireless computing devices naturally leads to an emphasis on networked operations and a seamless interaction with the ambient context. This trend is seen on PDAs, active RFIDs, various intelligent consumer electronic devices, and wireless sensor networks. Such networked operations and contextual interaction make the application software much more complex than that running on traditional embedded devices. Particularly, the sensor network is a representative technology where the relevant design factors such as resource constraints and application complexity are manifested to a great extent. A typical sensor node may only have a simple CPU and a few kilobytes of data memory [1], [2], [3], but the software running on it can take tens of thousands lines of code to implement, performing a wide range of tasks related to sensing, topology control, wireless routing, power management, signal processing, and system administration [4], [5], [6].

The complexity of application software and the fact that the software runs on numerous unreliable devices call for strong system software support [7], [8]. One critical need is a pre-emptive multitasking operating system. Without that, handling important interrupts could be delayed by long computational tasks, communication operations could disrupt the timing of the sensor channel sampling, and unpredictable latencies would make network level activity unreliable and energy costly.

Consequently, a number of recent operating systems for sensor networks have included multi-tasking and pre-emptive scheduling features. However, a careful examination of those systems shows severe limitations in both functionality and usability. One of the key problems, as mentioned by a classic research work on the topic of multitasking, is stack management [9] that is how can an operating system automatically and efficiently manage multiple stacks. Especially, the problem is even harder on a small-memory platform.

In a multitasking system, the stacks of concurrent tasks routinely grow and shrink during their execution. The dynamics of the stacks is of great variation for event-driven systems, which is the de facto standard programming model for sensornet systems [10], [11]. The ability to hold multiple stacks in memory and efficiently handle the stack dynamics is a fundamental determinant of a multitasking OS.

Designing Adaptive Stack Management:

Designing adaptive stack management on resource constrained platforms is a new challenge; one important question is whether we could avoid this problem by upgrading hardware to qualify traditional solutions. Though the low-power computing technology develops steadily, virtual memory is still very unlikely to be available to the sensor nodes using very low power processors. Some recent embedded processors claim to enable a 32-bit architecture with the cost and power consumption of 8-bit systems. The claim is, however, only partially true because downscaling power is often accompanied by removing architectural features. Most low-power microcontrollers (MCUs) do not support hardware memory translation or memory protection, and many low-power systems do not support instruction privilege, which is pre-requisite for traditional multitasking designs. It is also unlikely that very low power systems can afford to scale up physical memory size as quickly as the cost of RAM drops. In the past two decades, the typical memory capacity of computer systems has grown dramatically, but many MCUs today still use kilobytes of SRAM for energy efficiency.

OPERATING SYSTEMS FOR MULTITASKING SENSOR NETWORKS

Researchers have developed a number of operating systems for sensor networks and low-
power devices, such as TinyOS [3], SOS [14], Contiki [15], MANTIS OS [16], LiteOS [20],
SESAME/SESAME-P [18] [19] and the t-kernel [21] in order to support more reliable, efficient, and sophisticated applications. Multitasking has become an important feature in such systems.

The table below lists the implemented features of typical related systems as a comparison. Although these systems have respective advantages, SenSmart performs better in multitasking-related functionalities as listed. SenSmart also uses binary rewriting as an important technique to implement preemptive scheduling and memory isolation. Different from the t-kernel, SenSmart conducts complete binary translation on the base station. This approach brings unique advantages in reducing system complexity and code inflation ratio.

Table 1: Comparison of typical Systems

	Mat'e	**Contiki**	**MANTIS**	**t-kernel**	**RETOS**	**LiteOS**	**TOS Threads**	**SenSmart**
TinyOS Compatible	**No**	**No**	**No**	**Yes**	**No**	**No**	**Yes**	**Yes**
Preemptive Multi-tasking	**No**	**Yes**	**Yes**	**Partial**	**Yes**	**Yes**	**Yes**	**Yes**
Concurrent Applica-	**NA**	**No**	**No**	**No**	**No**	**No**	**No**	**Yes**

tions								
Interrupt-free Preemption	**NA**	**No**	**No**	**Yes**	**No**	**No**	**No**	**Yes**
Memory Protection	**Yes**	**No**	**No**	**Partial**	**Yes**	**No**	**No**	**Yes**
Logical Memory Address	**NA**	**No**	**No**	**No**	**No**	**No**	**No**	**Yes**
Memory Arrange-ment	**Automatic**	**Automatic**	**Automatic**	**Automatic**	**Automatic**	**Manual**	**Automatic**	**Automatic**
Stack Relocation	**No**	**No**	**No**	**No**	**No**	**No**	**No**	**Yes**

TinyOS [3]

In TinyOS [3], tasks are executed in serial. Hence, there is no concurrency among them, and the stack management is rather simple. Moreover, the memory isolation is absent so that
the program code can write to any physical memory areas.

TOSThread [23]

TOSThread [23] introduces user threads along with existing TinyOS tasks. Each thread is allocated an independent but _xed-size stack for local variables and execution context. Substantially, such multitasking models are tailored from the traditional design techniques, while they often work ine_ciently in resource constrained systems.

SOS [14], MANTIS OS [16], LiteOS [20]

Attempt to adopt traditional OS mechanisms as TOSThread does. Those traditional solutions usually lead to harsh restrictions on application tasks. For example, it is very difficult to efficiently allocate stack memory to tasks without introducing extra burden (and dependence) on application programmers. For correctness and simplicity, such systems usually allocate stack memory based on the worst case situation. Without virtual memory paging, this pessimism, combined with the aforementioned inflexible allocation, aggravates the waste and drains a fair portion of previous memory resources. The fundamental reason is the weak stack adaptivity, and consequently, limited application flexibilities.

SESAME/SESAME-P [18] [19]

Researchers have noticed that the traditional monolithic stack allocation can reduce the overall efficiency. To improve the flexibility, SESAME/SESAME-P [18], [19] propose novel solutions to convert the call stack into the heap area, and perform bookkeeping to manage the discrete stack blocks. The runtime overhead is mitigated by a flexible dynamic stack allocation mechanism.

Contiki [15]

Lightweight thread models can avoid the stack management problem by dramatically simplifying the semantics of concurrent tasks. For instance, the stackless protothreads in Contiki minimizes memory usage [13], [15], but they also incur severe functional limitations, e.g., no retention of state between context switches. Such limitations are likely to make programming harder.

t-kernel [21]

The t-kernel [21] implements preemptive scheduling, OS protection and virtual memory with binary rewriting on sensor nodes. The tasks in the t-kernel share a common stack space, and the memory protection is asymmetric that is only the kernel memory is protected. SenSmart also uses binary rewriting as an important technique to implement pre-emptive scheduling and memory isolation. Different from the t-kernel, SenSmart conducts complete binary translation on the base station.

CONCLUSION

In this paper a brief study of multitasking operating systems like Mat'e, Contiki, MANTIS OS, t-kernel, RETOS, LiteOS, TOSThreads, Sensmart is carried out. This paper analyses all these operating systems for their performance related to multitasking parameters. Under multitasking parameters out of these operating systems Sensmart performs best.

REFERENCES

[1] S. Lin et al., E_cient Indexing Data Structures for Flash-Based Sensor Devices,ACM Trans. Storage, vol. 2, no. 4, pp. 468-503, 2006.

[2] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events, Proc. Fourth Intl Conf. Information Processing in Sensor Networks, 2005.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System Architecture Directions for Network Sensors, Proc. Ninth Intl Conf. Architectural Support for Pro-gramming Languages and Operating Systems, 2000.

[4] M. Eltoweissy, D. Gracanin, S. Olariu, and M. Younis, Agile Sensor Network Systems, Ad Hoc and Sensor Wireless Networks, vol. 4, no. 1, pp. 97-124, 2007.

[5] T. He et al., VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveil- lance,ACM Trans. Sensor Networks, vol. 2, no. 1, pp. 1-38, 2006.

[6] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler, An Analysis of a Large Scale Habitat Monitoring Application,Proc. Second Intl Conf. Embedded Networked Sensor Systems, 2004.

[7] K. Romer and J. Ma, PDA: Passive Distributed Assertions for Sensor Networks,Proc. Eight Intl Conf. Information Processing in Sensor Networks, pp. 337-348, 2009.

[8] M. Khan et al., Diagnostic Powertracing for Sensor Node Failure Analysis,Proc. Ninth Intl Conf. Information Processing in Sensor Networks, pp. 117-128, 2010.

*[9] A. Adya, J. Howell, M. Theimer, B. Bolosky, and J. Douceur, Cooperative Task Manage-
ment without Manual Stack Management,Proc. USENIX Ann. Technical Conf., 2002.*

[10] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler, The nesC Language: A Holistic Approach to Networked Embedded Systems,Proc. ACM SIGPLAN Conf. Pro- gramming Language Design and Implementation, 2003.

[11] O. Kasten and K. Romer, Beyond Event Handlers: Programming Wireless Sensors with Attributed State Machines,Proc. Fourth Intl Conf. Information Processing in Sensor Networks, 2005.

[12] W. McCartney and N. Sridhar, Abstractions for Safe Concurrent Programming in Networked Embedded Systems,Proc. Fourth Intl Conf. Embedded Networked Sensor Systems, pp. 167-180, 2006.

[13] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems,Proc. Fourth Intl Conf. Embedded Networked Sensor Systems, pp. 29-42, 2006.
[14] C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, A Dynamic Operating System for Sensor Nodes,Proc. Third Intl Conf. Mobile Systems, Applications, and Services, pp. 163-176, 2005.

[15] A. Dunkels, B. Gronvall, and T. Voigt, ContikiA Lightweight and Flexible Operating System for Tiny Networked Sensors,Proc. 29th Ann. IEEE Intl Conf. Local Computer Networks, pp. 455-462, 2004.

[16] S. Bhatti et al., MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms, ACM/Kluwer Mobile Networks and Applications, vol. 10, no. 4, pp. 563-579, 2005.

*[17] A. Eswaran, A. Rowe, and R. Rajkumar, Nano-RK: An Energy- Aware Resource-Centric
rtos for Sensor Networks,Proc. 26th IEEE Intl Real-Time Systems Symp., pp. 256-265, 2005.*

[18] S. Yi, H. Min, S. Lee, Y. Kim, and I. Jeong, SESAME: Space E_cient Stack Allocation Mechanism for Multithreaded Sensor Operating Systems,Proc. 22nd Symp. Applied Computing, pp. 1201-1202, 2007.

[19] S. Yi, S. Lee, Y. Cho, and J. Hong, SESAME-P: Memory Pool- Based Dynamic Stack Management for Sensor Operating Systems,Proc. Third Intl Conf. Distributed Computing in Sensor Systems, pp. 544-549, 2008.

[20] Q. Cao et al., The LiteOS Operating System: Towards Unix-like Abstractions for Wireless Sensor Networks,Proc. Intl Conf. Information Processing in Sensor Networks, pp.233-244, 2008.

[21] L. Gu and J. Stankovic, t-kernel: Providing Reliable os Support to Wireless Sensor Networks,Proc. Fourth Intl Conf. Embedded Networked Sensor Systems, 2006.

[22] Rui Chu, Lin Gu,Yunhao Liu,Mo Li,Xicheng Lu "SenSmart: Adaptive Stack Management for Multitasking Sensor Networks" IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 1, JANUARY 2013.

[23] K. Klues et al., TOSThreads: Thread-Safe and Non-Invasive Preemption in Tinyos,Proc. *Seventh Intl Conf. Embedded Networked Sensor Systems, pp. 127-140, 2009.*

[24] Guragain, Bijay. "Power Efficient Routing Protocol for Mobile Ad Hoc Networks." *International Journal of Research* 1.4 (2014): 353-360.