An Approach to Memory management in Wireless Sensor Networks

Prof. Manjiri Pathak
Associate Prof., Computer Dept.,
Padm. Vasantdada Patil Pratishthan's College of Engg, Sion,
Mumbai, India,
manjiri_pathak@yahoo.com

Abstract— In recent years, wireless sensor network has become an important research domain. A typical WSN is a multi-hop wireless network consisting of hundreds or thousands of small sensor devices that are capable of sensing, processing (computing), and communicating. Nowadays WSNs represent a new generation of distributed embedded systems with a broad range of real-time applications. Some of the applications include process control, fire monitoring, border surveillance, medical care, asset tracking, agriculture, highway traffic coordination etc. Such systems need heavy computations & must meet new kinds of timing constraints under severe resource limitations & limited communication capabilities in highly dynamic environments.

In WSN, sensor devices are severely constrained by the resources. They usually consist of a processing unit with limited computational power & limited memory, sensors (including specific conditioning circuitry), a communication device (usually a radio transceiver or alternatively optical) and a power source in the form of a battery. As many of the new applications supporting real time traffic require more memory, it is challenging to design the efficient memory management techniques to support these applications. Although great amount of work is done in this area, still many problems have to be resolved. Especially there are many research gaps in case of memory management for WSN in supporting concurrent applications. In this paper we will discuss about the challenges to be considered while designing the efficient memory management system for these kind of applications & how these issues are handled in various OSs designed for this purpose. Next, we will also consider the further research opportunities in this area.

Keywords- Wireless Sensor Network (WSN), WSN operating systems, Memory management, Wireless Multimedia Sensor Networks (WMSN), WSN-cloud computing, challenges, research opportunities.

I. INTRODUCTION

Infrastructural support for WSN applications in the form of operating systems is becoming increasingly important. It bridges the gap between hardware simplicity and application complexity, and it plays a central role in building scalable distributed applications that are efficient and reliable. One of the important OS design issues which requires lot more attention is memory management in WSN.

Memory in current sensor nodes consists of: RAM (for fast data storage), internal flash (for code storage), EEPROM (for data storage), and external flash which is required for data persistence.

In a traditional operating system, memory management refers to various techniques used for allocation and de-allocation of memory blocks to different processes and threads. Commonly used memory management techniques are static memory allocation & dynamic memory allocation. Earlier, very little or no support used to be provided for managing the memory assuming that only single application runs on the sensor node. But with the emergence of new application domains for WSNs which support real time traffic, multithreaded, multi core designs, multimedia streams of data for transfer, these WSNs provide the mechanism for concurrent execution of multiple threads[4]. Since the memory is one of the constrained resource in case of WSN, it becomes a challenging task for applying these memory management techniques efficiently to various processes & threads with real time traffic. In the next section, we will see various issues & challenges relevant to memory management in WSN. Section III consists of the features & an approach of various WSN operating systems to the memory management. Section IV elaborates more about the research gaps & opportunities to contribute the work further in this area. The paper concludes with summary & conclusion.

II. MEMORY MANAGEMENT ISSUES & CHALLENGES

There are many issues & challenges that should be considered while designing efficient memory management system. In this section, we will see some of the major concerns in this area.

i. **Virtual memory**

Many of the sensor nodes lack or have very limited support for the address translation(MMU). As it is power intensive operation & sensor nodes have very limited power & storage capabilities, it is really challenging to provide more memory to the applications than assigned by the physical memory. Especially the work done in virtual memory management for WSN is very limited.[2]

ii. **Secondary storage management**

As many emerging WSN applications require more memory, and these applications require management of large databases & real time traffic, the need for secondary storage increases. In these types of applications, data must be stored in the network, and thus storage becomes a primary resource which, in addition to energy, determines the useful lifetime and coverage of the network. There are only few OSs that provide a file system to manage secondary storage. So the scalable(distributed) file system for WSNs to manage secondary storage has to be designed for such applications. The collaborative storage provides more suitability to meet the goals of storage management.

iii. **Small Footprint**

The storage capacity available on a sensor node is in terms of few kilobytes due to which the OS has to be designed with a very small footprint[15]. It is a fundamental characteristic of a sensor Operating System.

iv. **Task Scheduling & resource sharing**

It provides the task environment for executing long running application. The task scheduling can be either event-driven or multi-threaded. Here also memory management should be done in efficient way for memory allocation between scheduled tasks. Another issue to be considered is the resource sharing between execution of multiple applications. For this, the efficient concurrency control & memory protection mechanism should be provided between these applications.

v. **Dynamic memory allocation**

Data memory has been a very scarce resource in sensor networks[2]. Thus, its efficient utilization is necessary. Allocation of a memory to the dynamic data structures becomes a challenging task on the sensor node as the memory requirement varies depending on the size of the data structure. WSN applications with increasing application domains require efficient dynamic memory allocation techniques to be designed.

vi. **Reprogramming & memory management**

Reprogramming & up-gradation of soft wares on already deployed nodes is challenging because of the fact that sensor networks may be deployed in physically unattended environment and often consist of few thousands of nodes. Therefore, an already deployed sensor network must be wirelessly reprogrammable irrespective of above problems[2].

Reprogramming requires dynamic loading & unloading of the software modules or individual services & proper memory management policies like contiguous memory allocation, de-allocating memory, and paging[3]. For these policies to be enforced, proper APIs should be provided by OS to support reprogramming.

In the next section, we will see how the memory management approach is handled in various WSN operating systems.

III. MEMORY MANAGEMENT APPROACH IN WSN OPERATING SYSTEMS

In this section, we will see how these issues are handled in various operating systems designed for WSN.

a. TinyOS

- In early sensor network operating systems like TinyOS there was no memory management available(or little support in case of new version) assuming the execution of single application on a sensor node. Because of this, there was no support for real time applications.
- This OS uses static memory management.
- It provides single level file system so hardly it can support for secondary storage management. It provides database management in the form of TinyDB.
- It doesn't support virtual memory management.
- As far as the memory protection is considered, it exploits the concept of Deputy which is a resource to resource compiler that ensures type and memory safety for C code. Code compiled by Deputy relies on a mix of compile and run-time checks to ensure memory safety. In Tiny OS version 2.1, the memory safety is incorporated[4].
- As far as dynamic memory allocation is concerned, TinyAlloc compaction based memory manager is used in this O.S.[2] It allocates memory bytes from a fixed-size frame and returns double pointers to the allocated bytes. Because of which, TinyAlloc can move the memory around in the frame without having to change the external references used by the application program. This allows it to carry out compaction and avoid external fragmentation. In its TinyOS implementation, TinyAlloc is a split-phase operation. The

application makes a call to allocation routine and its completion is asynchronously signaled by allocCompleteSignal. Similar to allocation, compaction is also asynchronous and signaled via compactCompleteSignal.

- In case of dynamic reprogramming, the entire OS is propagated onto the target node, & then image is read into the program memory. The node is rebooted with the new image[2].

b. Nano-RK

- It is energy aware, resource-centric OS that supports real-time requirements and high level networking primitives as well as task management and synchronization mechanisms[14]. Due to this, it is also suitable for Wireless Multimedia Sensor Networks supporting multimedia traffic.
- It provides rich functionality and timing support using less amount of memory. It has a support for multihop networking & multitasking. It has extended WSN lifetime.
- Nano-RK provides support for static memory management. Here, both the OS and applications reside in a single address space.[5]
- It uses static configuration approach for energy usage control.
- It provides priority-based preemptive scheduling for multiple tasks, thus ensuring to meet the deadlines for complicated tasks.
- For shared resources such as memory, Nano-RK provides mutexes and semaphores for serialized access & concurrence control mechanism. In addition, it provides APIs to reserve the system resources. The tasks can specify their resource needs & OS provides guaranteed access to these resources.[15]

c. LIMOS

- Its kernel adopts a component-based three-level system architecture: action (system operation), thread (component) and event (container)[17]. So, it consists of a predictable and deterministic two-level scheduling mechanism: 'non pre-emption priority based' high level scheduling for events and 'preemptive priority-based' low level scheduling for threads.
- This natively hybrid operating system works in multi-threading & event driven modes to minimize resource requirements & improve system efficiency depending on the application diversity. It supports real time applications. It's distributive real time micro-kernel supports multi-level system architecture that adopts predictable and deterministic two-level scheduling mechanism[1] a) non pre-emptive priority based high level scheduling for events and b) preemptive priority-based low level scheduling for threads[6].
- It is dedicated to strict resource constrained embedded applications. It has less memory requirement(<5KB) comparing with other RTOSs[18].

d. SOS

- Here, linked lists of memory blocks having three different sizes are provided. Application can request for the fixed size block from one of the lists as per the requirement.[2]
- It supports runtime reconfiguration & reprogramming of the program code. In case of dynamic reprogramming, the modules are loaded & unloaded at run time. Since the modules are position independent binary, they can be dynamically linked. Here relative address is used rather than the absolute address thus providing relocatability. Due to module level reprogramming, the memory requirement is reduced. The updates are installed directly into the program memory thus reducing the energy & time requirements.
- The current implementation of SOS doesn't provide the memory protection.

e. MANTIS

- It is multi-threaded, lightweight & energy efficient OS & is portable across multiple platforms.
- It provides preemptive, priority based scheduling.
- It's small footprint of 500 bytes, includes kernel, scheduler, and network stack.
- The memory is dynamically allocated to the stack & process registers for each thread[11].
- Even though here the support for real-time multimedia applications is not provided in its communication protocol stack, one can implement real-time transport and routing protocols for multimedia sensor networks in MANTIS by using custom routing and transport layer protocols on top of the MAC layer.
- In case of dynamic reprogramming, the entire OS is propagated on the node & then it is rebooted with new image. The reprogramming is possible here at the thread level also where the values of variables can be changed.
- It maintains two logically distinct sections of RAM: the space for global variables that is allocated at compile time & remaining part of the RAM is managed as a heap[3]. Due to the multi-threaded model of operation, every Mantis program must have a stack space allocated from the system heap, and the space is retuned to the heap once the thread completes it execution. Locking mechanisms must be used to achieve mutual exclusion of shared variables[11].

- Multithreading support here comes at the cost of context switching and stack memory allocation. Since the kernel occupies 500bytes of memory, remaining portion of the memory is available to support multithreading.[3]
- MANTIS manages different threads' memory using the thread table & does not provide any mechanism for memory protection.
- In MANTIS, mutexes & counting semaphores are used to avoid race conditions. It performs resource sharing with the help of semaphores.

f. Contiki

- It is a multitasking, light weight, multithreading, open source operating system developed for portable and memory-constrained embedded systems that contains optional pre-emptive scheduling.
- The Contiki kernel comprises of a lightweight event scheduler that dispatches events to running processes.
- It supports dynamic memory management & dynamic linking of the programs. For this purpose, it provides powerful memory block management functions for blocks of fixed length. It keeps the allocated memory free from fragmentation by compacting the memory when blocks are freed. Therefore, a level of indirection is used here. A memory block is statically declared using the MEMB() macro. Memory blocks are allocated from the declared memory by the memb_alloc() function, and are de-allocated using the memb_free() function[3][7].
- A full system with the code & provision of GUI requires about 30 KB of RAM. Thus it contains a small footprint with less requirement for memory consisting of the base system, web server, virtual network computing server (VNC), and a small virtual desktop. It has two low power TCP/IP communication stacks supporting IPv4 & IPv6, consisting of µIP(to communicate over the internet) and Rime(designed for low power radios), where, it allows limited TCP/IP connections to the systems featuring TCP, IP, unicast UDP, ARP, ICMP, SLIP protocols[10]. Here micro references to the small memory requirements of a program code, needed to support a limited number of connections.
- Contiki provides a flash-based file system for storing data inside the sensor network called as 'Coffee'. It allows multiple files to coexist on the same physical on-board flash memory.
- Contiki does not provide any memory protection mechanism between different applications.
- In case of dynamic & remote reprogramming, instead of propagating the entire OS binary image, the modules are loaded & unloaded at run time like in SOS, thus reducing the memory, energy & time requirements while transmitting through the network[7]. Here the core code is kept separate from the program code in ROM & the program code is loaded in memory at runtime. Due to communication capability with the network using TCP/IP, it can also load and unload programs over the network connection into RAM or ROM.

g. LiteOS

- Since it is a Real Time OS used for WSN, it supports real time applications.
- In this OS, entire network is modeled as a distributed file system. It consists of hierarchical file organization & remote scriptable wireless shell interface is provided using UNIX like commands[12].
- It uses a plug-and-play routing stack and is a lightweight event logging.
- LiteOS provides a familiar programming environment based on UNIX, threads, and C.
- In this OS, both thread driven & event driven programming(through callback functions) is supported[8][9].
- Dynamic memory management is supported at the kernel level through C like malloc & free functions. It also supports online debugging and file system assisted communication stacks.
- Here memory address space assigned to the kernel is separate from that of an application & also between multiple applications thus providing the security.
- It supports object oriented programming using LiteC++ & Unix like shell.
- It provides round robin & priority based scheduling.
- Kernel supports dynamic loading, unloading of multithreaded applications and concurrent execution of multiple threads without overlapping the memory sections accessed by them.
- Wireless reprogramming is done at application level.
- LiteOS maps WSN to UNIX like file directories, allowing user friendly operations. It uses modified HEX files that are smaller in size to relocate information.
- It provides little support for real time applications.
- It contains smaller footprint which is suitable for resource constrained nature of WSN[8][9].

h. Enix

- Enix is a lightweight dynamic operating system for tightly constrained platforms of WSN.
- Enix supports remote reprogramming.

- A lightweight, efficient file system named EcoFS is included in Enix that is configurable. Here only required resources are used to conserve the code memory & to increase the life time of WSN. Here 4 types of data are supported: binary code data, preferences of sensor devices, network data such as routing tables, and sensed data. Active time of the Micro-SD card is reduced by I/O scheduling and by limiting the access via the API provided by EcoFS. To reduce code size, we let the controller inside the Micro-SD card handle wear leveling (i.e., evenly distributing writes among pages) and erase-before-write[13].
- The replaceable scheduler is used to support different scheduling policies. It's efficiency can be improved by fast algorithms & overhead of context switching(which is required while scheduling another thread by preempting the running one) can be reduced by storing the critical registers on the internal stack instead of using external memory.
- Virtual memory is supported with the assistance of a compiler. It is achieved using demand segmentation & without MMU. Here each function is compiled into a code segment that is then stored on the Micro-SD card at a specific virtual address. The segment is loaded on demand by using the approach of PIC(Position Independent code) while the user program calls this virtual address at run-time. Memory compaction and garbage collection are implemented to solve the problem of external fragmentation by checking the memory usage periodically and to recycle unused memory for future allocation. The cascaded call problem is handled in Enix by restricting the garbage collector to only non swappable code.

i. µC/OS-II

- µC/OS-II is a real-time preemptive multitasking embedded OS kernel, which is portable & scalable that can support time-sensitive tasks[19].
- It provides various functionalities for networked applications such as multitasking, synchronization, timer management, memory management.
- It also enables micro sensor nodes to natively interleave complex tasks with time-sensitive tasks, thereby mitigating the bounded buffer producer-consumer problem.
- It is easily applicable to Wireless Sensor network applications because of its advanced functionalities.
- µC/OS-II can effectively support concurrency model.
- In case of small footprint, its RAM size has to be still reduced to run normally on h/w platforms of sensor nodes. So to reach memory efficiency, this OS provides a function OSTaskStkChk() to reduce the amount of RAM needed by the application code by not over allocating stack space. The RAM space can also be saved by modifying the system function called OSTaskDel().
- As far as remote reprogramming is concerned, it can work on following possibilities: re-flashing of the entire OS; reprogramming of a single thread; and changing of variables within a thread. To overcome the difficulty of reprogramming in the network, a shell is designed for µC/OS-II that runs in PC to connect with the sink node. The programmer simply connects the sink node to a PC through LAN and opens the µC/OS-II shell.

Even though the issues related to memory management are handled in these operating systems up to some extent & improvements are made in the newer versions of earlier operating systems or new operating systems, still there are some lacunas in each & every OS. So there are many research gaps & opportunities to work further in this area. In the next section, we will see how these research gaps can be considered as a future research scope in this field.

IV. RESEARCH GAPS & OPPORTUNITIES IN OS DESIGN W.R.T. MEMORY MANAGEMENT FOR WSN

In this section, we will discuss, some of the challenges where further research work is required in this area:

- As new application areas with real time traffic need more memory, the sensor nodes with large secondary storage are required[14]. As huge amount of data is collected & processed at the nodes, this data has to be stored & maintained in large databases. So the requirement of secondary memory management here is also increased. To achieve this, much more improvement in file system to manage the secondary storage & large databases is required.

  Even though, some of the OS like Enix provides lightweight & efficient file system, still there is a scope in implementation of efficient secondary memory management.
- In the existing operating systems for WSN, there is very limited work done in case of memory management for multiple concurrent applications & support for virtual memory management. Even though some amount of work is done w.r.t. this issue in some of the operating systems as mentioned above, still significant amount of work is required here to support heavy real time data storage & computations. It should also be energy & memory efficient. Especially, in WMSN there is much more work to be carried out in memory management where the network consists of the heterogeneous sensors that are capable of retrieving multimedia data like audio, video, images, scalar sensor data & do collaborative sensing[14].

- Even though, schedulers in some WSN OSs have been designed to support soft as well as hard real-time operations, still there is a scope of work to be done in fair scheduling of real time data & to achieve concurrency. This issue is taken into consideration here, as the memory allocation between multiple tasks or concurrent execution of multiple applications should be done in efficient way. Along with the same, efficient memory protection is also required during allocation.
- As mentioned earlier, remote reprogramming of densely deployed nodes is a challenging task due to dynamic loading & unloading of the modules or components which should be handled efficiently by designing proper APIs.
- Another emerging field of research w.r.t. WSN is integrating it with cloud computing[16]. This next generation of WSN will benefit when sensor data is added to blogs, virtual communities, and social network applications. While the idea of having cloud computing just as a transparent layer on top of real world WSNs is appealing, its realization is really challenging. The devices in the cloud are normally equipped with powerful processor, large memories and constant power source. In WSNs the nodes are equipped with constrained resources. Thus, combining both worlds requires fully new concepts that help to do proper load balancing from the very beginning and by design & it can be achieved with concept of distributed shared memory.

SUMMARY & CONCLUSION

As the range of possible WSN application domains with heterogeneous sensors is growing, lots of improvements & further investigations are required to give stronger real time support & efficient memory management techniques for WSNs. Much more attention is required for resolving the problems in areas like WMSNs, integration of WSN with cloud computing etc.

In this paper, the issues & research challenges related to memory management that should be considered while designing the operating system for WSNs are discussed. In the next section, the features & an approach to memory management are considered for some of the WSN operating systems. Further, the research gaps & opportunities for future work in this area are mentioned. Even though the significant amount of work is done here, there are still the research gaps & so a wide scope for additional work in this field.

ACKNOWLEDGMENT

I would like to thank the authors of references whose contribution in this field helped me to prepare my paper.

REFERENCES

[1] https://moodle.polymtl.ca/pluginfile.php/84612/mod_resource/content/0/wireless.pdf
[2] http://www.comp.nus.edu.sg/~doddaven/cata.pdf
[3] http://www.mdpi.com/1424-8220/11/6/5900
[4] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3231431/
[5] https://www.msu.edu/~liyang5/docs/nanork_tutorial.pdf
[6] http://motive.cemagref.fr/_publication/PUB00030956.pdf
[7] http://contiki.sourceforge.net/docs/2.6/a01685.html#_details
[8] http://en.wikipedia.org/wiki/LiteOS
[9] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4505477&url=http://ieeexplore.ieee.org/iel5/4505448/4505449/04505477.pdf%3Farnumber%3D4505477
[10] http://www.sics.se/~adam/contiki/
[11] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. 'MANTIS: system support for MultimodAl NeTworks of In-Situ sensors'. In *Proc.WSNA'03*, 2003.
[12] http://dl.acm.org/citation.cfm?id=1372737
[13] Yu-Ting Chen, Ting-Chou Chien , Pai H. Chou 'Enix: A Lightweight Dynamic Operating System for Tightly Constrained Wireless Sensor Platforms', SenSys'10, November 3–5, 2010, Zurich, Switzerland.
[14] Manjiri Pathak, 'OS design challenges & research opportunities in real-time WSNs & approach for real time support in Nano-RK', COMPUSOFT, An international journal of advanced computer technology, 2 (7), July-2013 (Volume-II, Issue-VII)
[15] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5462978&url=http%3A%2F%2Fieeexplore.ieee.org
[16] http://www.lcc.uma.es/~tolo/publications/WSN-ADT12.pdf
[17] Hai-ying Zhou, Feng Wu, Kun-mean Hou, 'An Event-driven Multi-threading Real-time Operating System Dedicated to Wireless Sensor Networks', 2008 International Conference on Embedded Software and Systems.
[18] http://www.bvicam.ac.in/news/INDIACom%202011/210.pdf
[19] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4351116&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4351116