Memory Management Strategies for Different Real Time Operating Systems

Thesis · February 2016

DOI: 10.13140/RG.2.2.25470.18243

1 author:

Zafar Ali Khan
Software Technology Innovators
8 PUBLICATIONS **1** CITATION

SEE PROFILE

Memory Management Strategies for Different Real Time Operating Systems

Zafar Ali
Virtual University of Pakistan,
Federal Government University
zafaralikhan14@gmail.com

***Abstract*—Choice of an RTOS for an embedded system is quite challenging. Detailed understanding of architectures and processes is necessary. Most critical part is the selection of memory management unit. Dynamic memory management in hardware is essential for hard real time systems. Hardware assures the process is deterministic.**

Keywords—RTOS; MMU; embedded system; SOC; memory management.

I. Introduction

A Real-Time Operating System (RTOS) is an Operating System (OS) intended to serve real time application requests. It must be able to process data as it comes in, typically without buffering delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter. Real-time operating systems have evolved over the years from being simple executives using cyclic scheduling to the current feature-rich operating environments.

Selection of the best RTOS for the embedded system in use is quite a challenging task as there are too many products available in the market [1, 2]. Possibilities of CPU software combination are given by the following figure.

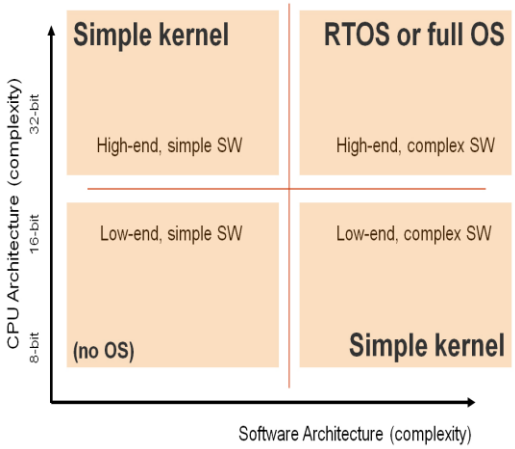

Figure 1: The four quadrants are based on CPU complexity and software architecture complexity [2].

An RTOS is selected based on the application requirement and embedded architecture. For a real time application the designer needs to decide how critical the time is. For hard real time system if the output is not delivered at the required instant there could be a catastrophic disaster. Interrupt latency plays a critical role to guarantee a hard real time output.

One of the most important aspects is memory management. Rest of the paper discusses various aspects of memory management.

II. Related Work

Most of the research work is focused on comparison of existing products or algorithms for memory management. As there are too many

products and algorithms and many new products are coming up almost every day the results are of little help while actual selection at work place.

Most important research results are based on innovative ideas to cater to the need of emerging embedded architectures and applications.

III. Problem Statement

Embedded system usually has much less memory as compared to mainframe or desktop. Large memory cannot be allowed as it adds to weight and cost of the system. This memory needs to be distributed among various tasks. When a task is complete the associated memory can be released. When a lower priority task is suspended to provide entry of the higher priority task a part of memory associated to lower priority task can be released. Multiple allocation and release creates unused holes in the memory if it is used as a single chunk.

From the compiler perspective garbage collection or collection of 'free'-d memory is very important. Unfortunately most widely used compiler for embedded systems C / C++ does not have the garbage collection feature. An RTOS needs to take care of this feature. RTOS might have a provision to accommodate a memory management unit.

Memory allocation / release can be static or dynamic. Static allocation is done before execution. Heap memory should be allocated in a static manner. The C language command 'malloc' should not be used as it is not deterministic and does not guarantee delivery in real time.

This global memory can be divided into pages and de / allocated dynamically. The process should be fast. It should not suffer from memory leakage. Write before Read should not happen. For multi users memory must be protected from malicious and inadvertent attacks.

Research is concentrating on innovative strategies to endow the memory management unit with these features.

IV. Proposed Solution

Xiuqing [3] analyzed two algorithms TLSF and ERMM for memory allocation and release while development of a new type of RTOS. Following bar-chart provides the comparison of the two algorithms.

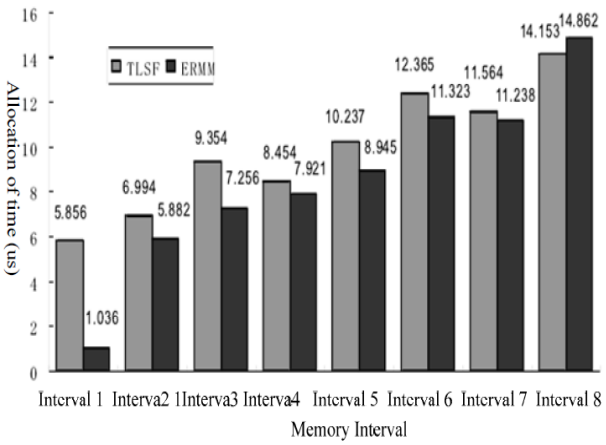

Figure 2: Comparison of memory allocation time [3].

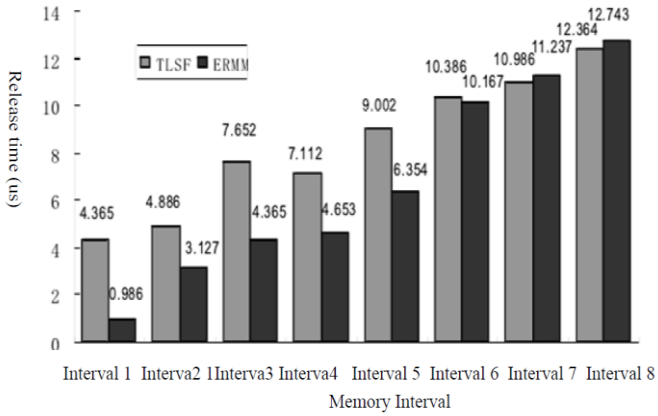

Figure 3: Comparison of memory release time [3].

For both allocation and release ERMM is more efficient for smaller intervals but TLSF is slightly better for longer intervals.

Shalan and Mooney proposed a System-on-a-Chip Dynamic Memory Management Unit (SOCDMMU) [4, 5]. The unit is configured in hardware so memory management is very fast and

deterministic. Also there is no memory overhead. In an SOC there can be different types of Processor Elements (PE) as shown in fig. 4. In this example SOC there are 2 types of PE-s Digital Signal Processor (DSP) and Reduced Instruction Set Computer (RISC). Four PE-s are shown in the figure.

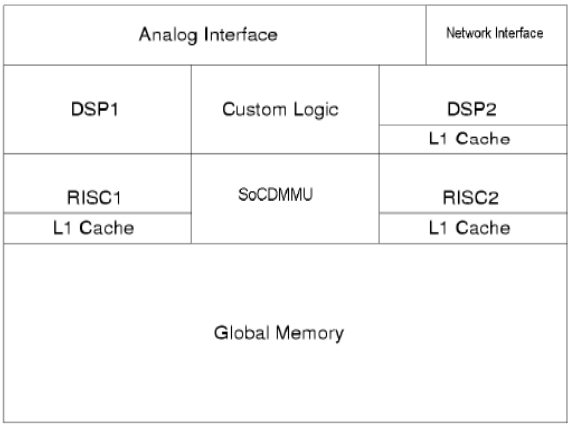

Figure 4: Example of an SOC with 4 PE-s [4].

The global memory is allocated to each PE called as local or virtual memory. The global memory allocation and de-allocation to PE-s is done by SOCDMMU. The memory can be SRAM as well as DRAM.

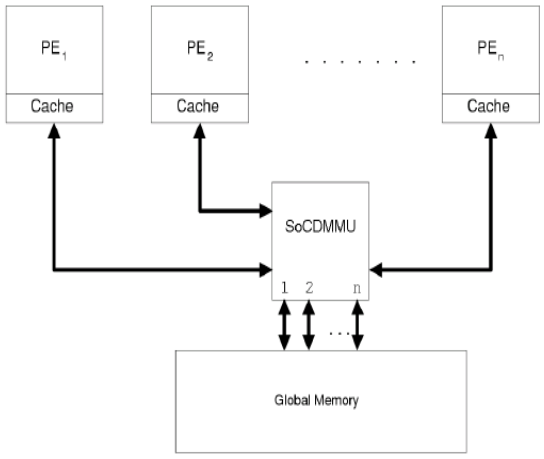

Figure 5: Memory allocation and de-allocation to PE-s by SOCDMMU [4].

Each memory page consisting of a multiple of memory blocks can have 3 types of accesses for each PE. The memory can be accessed as:

I. Exclusive memory: Only the owner PE can access the memory.

II. Read Only: PE has only read permission but no write access.

III. Read / Write: PE has read and write access. All other PE-s may have read access but no write access. In other words, for all other PE-s the page is Read Only. This restriction significantly reduces cache coherency problems.

SOCDMMU is implemented in hardware. The verilog code has open access. The speed of execution is compared when implemented on PIC micro-controller. The result is given in the table below. Results show the proposed scheme has a better execution time.

Table 1 : Comparison of speed of execution [4].

SoCDMMU Worst-Case Execution Time	20 Cycles
Micro-controller Best-Case Execution Time	221 Cycles

Since the allocation process is deterministic it works effectively for RTOS.

Ingstrom and Daleby[6] implemented a Dynamic Memory Allocator (DMA). This is different from the popularly known Direct Memory Access controller. The allocator is implemented in hardware using VHDL coding and ported in an FPGA chip, XILINX Spartan XCS40XL. Results show the hardware allocator is much more efficient than the software counterpart.

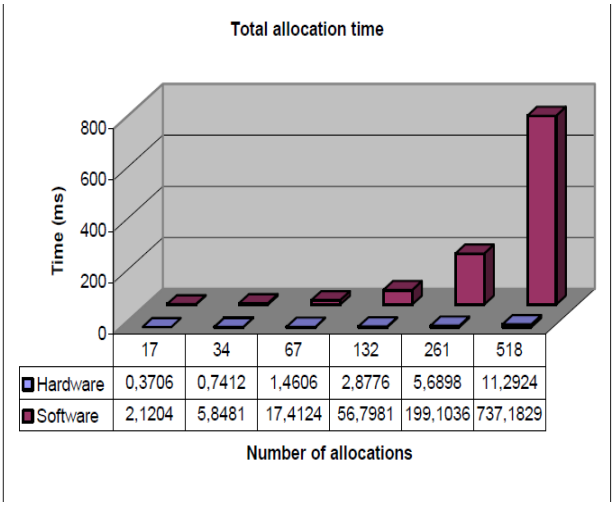

Figure 6: Allocation time for number of allocations [6].

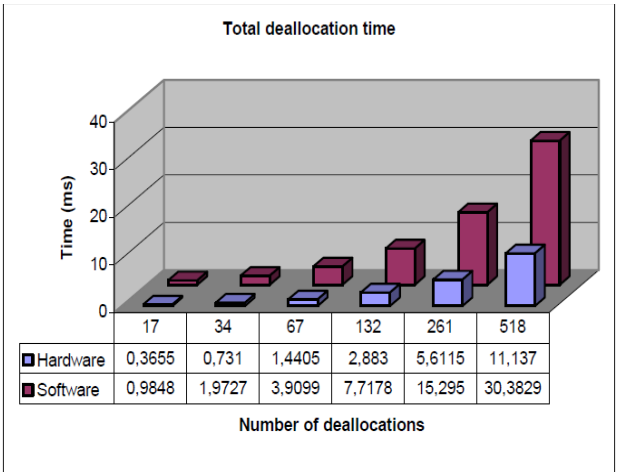

Figure 7: Deallocation time for number of deallocations [6]

Plots show hardware allocation and deallocation is much faster than the software counterpart. The speed difference increases with number of allocations and deallocations.

Wang et al. [7] developed the formal architecture and memory management for RTOS. Authors explained the processes with state transition diagrams as shown in the figure below.

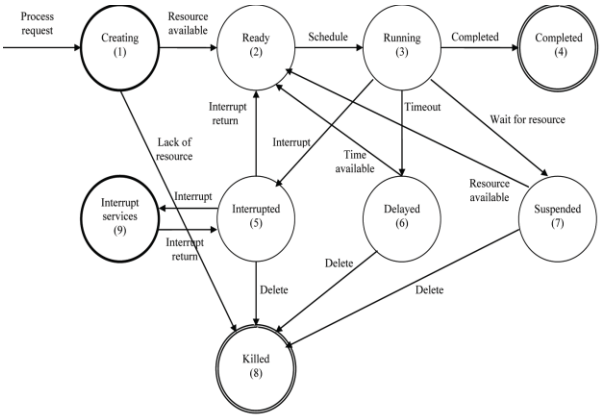

Figure 8: State transition diagram [7].

Memory allocation algorithm in RTOS is drawing lots of research attention off-late. Proposed algorithms [8, 9] are briefly explained in the following:

I. Sequential Fit:

The algorithm uses single linear list of all free memory blocks. The blocks are allocated from the list. The algorithm is easy to develop but suffers from slow execution.

II. Buddy System:

It uses an array of free lists, one for each allowable block size. The buddy allocator rounds up the requested size to an allowable size and allocates from the corresponding free list. If the allocated free list is empty then another free list is selected. The main advantage of buddy system is that freed blocks can be found quickly by simple address computations.

III. Indexed Fit:

The algorithm uses an indexing data structure like tree to find the free blocks.

IV. Bit Mapped Fit:

The algorithm uses a bitmap to represent the usage of the heap.

V. Two Level Segregated Fit (TLSF):

TLSF is a combination of segregated and bitmapped fit algorithm. It solves the worst case bound problem and produces highly efficient allocation and de-allocation.

VI. Smart Memory Allocator:

It is a new allocation style which allows designing a custom dynamic memory allocation mechanism with reduced memory fragmentation and superb response time.

A comparison of different memory management system is shown in the following table.

Table 1: Comparison among different memory allocation systems [8].

		Parameters		
		Fragment-ation	*Memory Footprint Usage*	*Response Time*
Memory Management Algorithms	*Sequential fit*	Large	Maximum	Slow
	Buddy System	Large	Maximum	Fast
	Indexed fit	Large	Minimum	Fast
	Bitmapped fit	Large	Maximum	Fast
	TLSF	Small	Minimum	Faster
	Smart Memory Allocator	Reduced	Minimum	Excellen t

Karthikeyan *et al.* [9] proposed a new scheme for memory management in RTOS. Authors also briefed the existing algorithms. Algorithm steps are given in the following flow-chart.

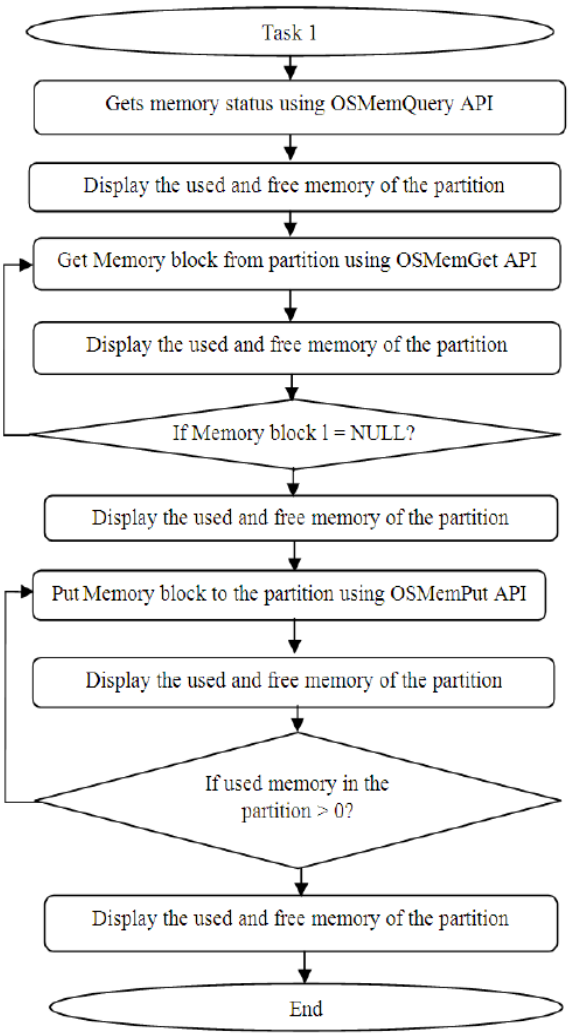

Figure 9: Steps in RTOS memory management [9].

µC/OS II RTOS is installed in LPC 1768 micro-controller. The experimental setup is shown in the following figure.

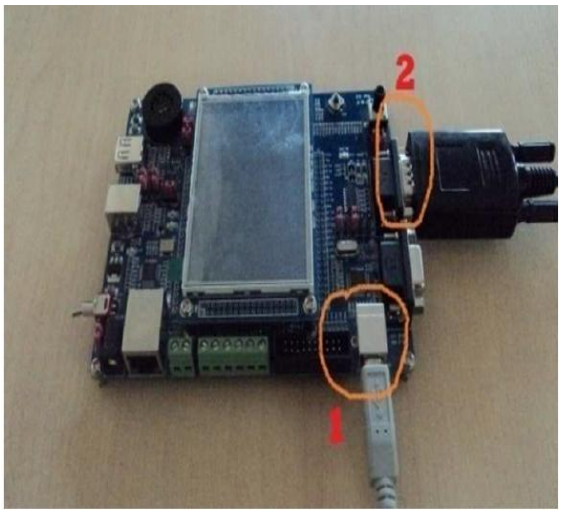

Figure 10: Experimental setup. 1. USB cable of LPC 1768 is connected to USB port of PC. 2. Serial port of LPC 1768 is connected to PC serial port [9].

The paper lacks comparison of the proposed algorithm with the existing ones.

Coleman and Zalewski [10] compared performance of some of the popular existing RTOS products. Performance or execution time is usually estimated as a number of clock cycles. For a multi-tasking system such unit does not give the clear picture. Authors defined two types of timing:

$T_{lifetime} = T_{current} - T_{start}$ (1)

$T_{wait} = T_{lifetime} - (T_{user} + T_{kernel})$ (2)

where:

T_{start}: Time a thread began.

T_{user}: Amount of time spent in user mode.

T_{kernel}: Amount of time spent in kernel mode.

In the following two figures comparison is done for MAC and Linux. T_{kernel} / T_{user} is plotted against execution time. Lower the value better is the peformance. From the plots it can be noted that Linux is giving better performance.

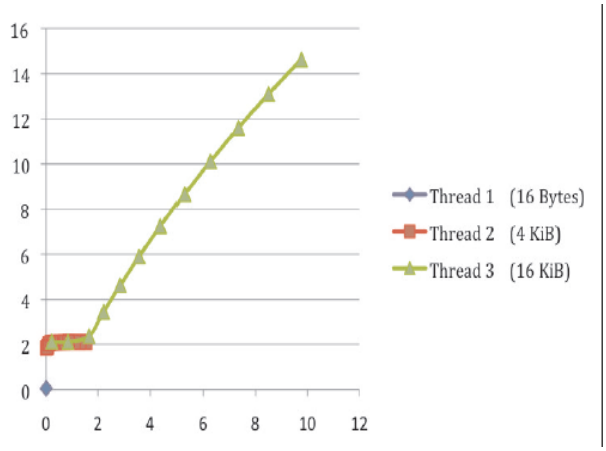

Figure 11: Ratio of time in kernel mode to time in user mode vs. execution time for Hoard on MAC OS 10.6.8 [10].

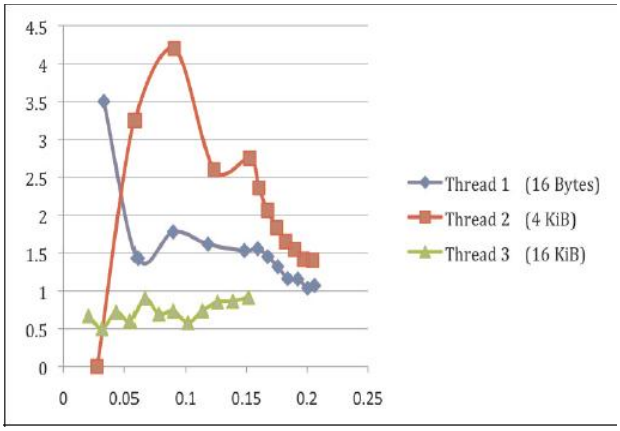

Figure 12: Ratio of time in kernel mode to time in user mode vs. execution time for Hoard on Linux 3.2.16 [10].

This observation should not be generalized as Linux is always better than MAC OS. For another RTOS 'jemalloc' MAC OS performs better than Linux.

Shah did comparisons for another set of RTOS products [11]. Instead of performance analysis other parameters e.g. synchronization, interrupt and event handling were compared.

There are still challenges for distributed systems e.g. wireless sensor network [12]. Sangorrin et al. did a detailed study of dual operating system architecture [13]. Memory management is quite

complex when security issues are considered. ARM TrustZone security hardware is utilized to overcome these issues.

V. Discussion and Analysis

- Selection of memory management strategy is quite challenging.
- Selection requires detailed understanding of the architecture and processes,
- Initial allocation should be static and not by the 'malloc' command in C language. As the execution result of this command is not deterministic.
- The process should be dynamic.
- The unit should be implemented in hardware.
- Hardware unit assures deterministic delivery and more efficiency.
- Future research must focus on more hardware implementations. The RTOS may be delivered in hardware.

VI. Conclusion

Memory management is a big challenge for embedded system users. Efficient management demands detailed understanding of the architectures and processes. Memory from heap should be transferred to global memory by static allocation and not by a command e.g. 'malloc' as the command execution is not deterministic. Among different tasks memory can be allocated and released dynamically. The management unit should be in hardware. In future whole RTOS is expected to be in hardware.

References

[1] [2005B1] S. Baskiyar and N. Meghanathan, "A survey of contemporary real-time operating systems", Informatica, vol. 29, 2005, pp. 233 – 240.

[2] [2014W1] C. Walls, "Selecting an operating system for embedded applications", White Paper, Mentor Graphics, 2014, pp. 1 – 8.

[3] [2015X1] H. Xiuqing, "Construction on embedded real time operating system of computer", 2nd International Conference on Electrical, Computer Engineering and Electronics, ICECEE 2015, pp. 973 – 979.

[4] [2000S1] M.A. Shalan, V.J. Mooney, "A dynamic memory management unit for embedded real-time system-on-a-chip", CASES'00, November 2000, San-Jose, CA, pp. 142 – 148.

[5] [2003S1] M.A. Shalan, "Dynamic memory management for embedded real-time multiprocessor system on a chip", PhD Thesis, 2003, Georgia Institute of Technology, USA.

[6] [2002I1] K. Ingstrom and A. Daleby, "Dynamic memory management in hardware", Master thesis, Malardalens University, Vasteras, Sweden, 2002.

[7] [2010W1] Y. Wang, G. Zeng, C. Philip, Y. Sheu, C.C. Philip, and y. Tian, "The formal design model of a real-time operating system (RTOS+): Conceptual and architectural frameworks", International Journal of Software Science and Computational Intelligence, vol. 2, no. 2, IGI Global, April-June 2010, pp. 105 – 122.

[8] [2012D1] D. Diwase, S. Shah, T. Diwase, P. Rathod, "Survey report on memory allocation strategies for real time operating system in context with embedded devices", International Journal on Engineering Research and Applications, vol. 2, no. 3, May – June 2012, pp. 1151 – 1156.

[9] [2014K1] V. Karthikeyan, S. Ravi, and M. Anand, "Robust memory management using real time concepts", Journal of Computer Science, vol. 10, no. 8, 2014, pp. 1480 – 1487.

[10] [2013C1] A. Coleman and J. Zalewski, "A study of real-time memory management: Evaluating operating system's performance", Automatyka / Automatics, vol. 17, no. 1, 2013, pp. 29 – 42.

[11] [2013S1] S. Shah, "Real-time operating systems: The next stage in embedded systems", Advance in Electronic and Electric Engineering, vol. 3, no. 5, 2013, pp. 543 – 552.

[12] [2004S1] J.A. Stankovic and R. Rajkumar, "Real-time operating systems", Real-Time Systems, Kluwer Academic Publishers, vol. 28, 2004, pp. 237 – 253.

[13] [2010S1] D. Sangorrin, S. Honda, and H. Takada, "Dual operating system architecture for real-time embedded systems", S.M. Petters and P. Zijlstra Eds., 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2010, pp. 6 - 15.