https://doi.org/10.48047/AFJBS.6.Si4.2024.336-343

African Journal of Biological Sciences
Journal homepage: http://www.afjbs.com
Research Paper Open Access

Challenges in RTOS Memory Management: Strategies for Optimization

Khushboo
Assistant Professor,
Departmemt of Electronics and Communication Engineering,
University Institute of Engineering and Technology,
Maharshi Dayanand University, Rohtak

Article Info

Volume 6, Issue Si4,2024
Received: 15 May 2024
Accepted: 05 June 2024
doi:10.48047/AFJBS.6.Si4.2024.336-343

Abstract
Applications that need constant performance and exact timing require real-time operating systems (RTOS). Maintaining RTOS's predictability, efficiency, and dependability relies heavily on memory management. Memory fragmentation, few memory resources, severe real-time limitations, and concurrency problems are some of the main obstacles to RTOS memory management that this research seeks to address. Problems like increasing latency, unexpected behaviour, and inefficient use of resources are major roadblocks on the path to peak system performance. Dynamic memory allocation, memory partitioning, and garbage collection are some of the current memory management approaches that are examined in the research. Their merits and shortcomings are highlighted. We investigate novel approaches and optimisation techniques to tackle these difficulties. In order to improve the efficiency and predictability of memory management, we offer adaptive algorithms, memory pooling, and hardware-assisted management strategies. We assess and contrast the efficacy of various approaches by conducting empirical evaluations in simulated and real-world case studies. The results show that adaptive and hardware-assisted methods are able to guarantee constant real-time performance, enhance resource utilisation, and drastically decrease memory fragmentation. The research finishes with some useful suggestions for developers and designers of systems, stressing the importance of a balanced strategy that incorporates various memory management methodologies. This study helps improve real-time operating systems (RTOS) by removing obstacles and maximising efficiency, making them better prepared to handle the demanding requirements of mission-critical real-time applications.
Keywords – Dynamic Memory Allocation, Memory Partitioning, Garbage Collection, Adaptive Algorithms, Memory Pooling

Introduction

Aircraft, automobiles, healthcare, industrial control systems, and communications are just a few examples of industries that rely on real-time operating systems (RTOS) for reliable and accurate timing and performance. Unlike general-purpose operating systems (GPOS), which attempt to optimise throughput and user experience, RTOS are intended to fulfil rigorous timing constraints and guarantee that operations are done within their deadlines. An RTOS's memory management skills greatly impact its dependability and predictability. These capabilities are critical for reducing latency, effectively managing resources, and keeping the system stable.

There are a number of key ways in which RTOS differs from GPOS. In contrast to RTOS's emphasis on deterministic behavior—that is, on tasks executing at predictable times and according to rigid deadlines—GPOS aims to maximise resource utilisation while offering a rich user experience. When dealing with real-time applications, this predictable behaviour is very essential, since any delay in task execution might have disastrous results. So, for RTOS to provide reliable and timely task execution, memory management must be effective. There are a number of peculiarities with RTOS memory management that aren't usually present in GPOS.

Memory fragmentation is a major problem with real-time operating systems. Free memory becomes fragmented into tiny, non-contiguous chunks as a result of dynamic memory allocation and deallocation over time. Even if there is enough total memory, allocation requests may fail due to fragmentation, which might result in system failures or poor performance. Minimising fragmentation and maintaining efficient and reliable memory allocation need effective memory management solutions.

Many embedded systems that run real-time systems have restricted memory resources. Because of their limited resources, these systems must use very effective memory allocation algorithms to make the most of the available memory. Innovative memory management solutions that do not sacrifice system performance or reliability are required in this memory restricted environment.

To avoid missing task execution deadlines, real-time operating systems (RTOS) must adhere to stringent scheduling constraints for memory management procedures including allocation and deallocation. It is critical to provide efficient and predictable memory management methods since complicated algorithms might lead to unpredictable delays, which can compromise the real-time assurances offered by the RTOS.

Many real-time operating systems (RTOS) execute several processes in parallel, necessitating strong methods to manage memory access by multiple processes at once. Data corruption and inconsistency may be prevented by using synchronisation techniques, however this can add complexity and more work. System stability and speed must be preserved throughout concurrent task execution, which can only be achieved with effective concurrency control.

The ability for processes to request and release memory during runtime is made possible by dynamic memory allocation, which provides for flexibility in memory consumption. But it also raises the possibility of fragmentation and random allocation delays. More organised memory allocation is provided by strategies like slab allocators and buddy systems, which balance the trade-offs between predictability and flexibility, and hence alleviate these concerns.

The process of memory partitioning is separating RAM into chunks of a certain size that are then assigned to specific activities. With this method, fragmentation may be lessened and allocation times can be predicted; but, if the divisions aren't appropriately sized for the activities they handle, RAM might be wasted. To achieve a happy medium between memory economy and predictability, one must meticulously design and maintain memory partitioning.

The reclamation of unused memory by automated garbage collection is one way that memory may be better managed. Nevertheless, because to the complexity and unpredictability it adds, trash collection is not ideal for demanding real-time systems. To solve these challenges, there are variations such as incremental or real-time garbage collection. These approaches distribute the collection duty across time, balancing memory reclamation with system predictability.

Memory management techniques that are adaptive change their actions on the fly in response to changes in system load and memory consumption trends. These algorithms are designed to improve system performance by balancing predictability and efficiency via adaptation to changing situations. When it comes to managing memory, adaptive algorithms provide a versatile solution that can adapt to changing system needs and optimise resource utilisation on the go.

In memory pooling, tasks share a common pool of pre-allocated memory blocks of a predetermined size. Both fragmentation and the timeframes it takes to allocate and deallocate resources may be drastically reduced using this method. By providing a happy medium between efficiency and predictability in memory allocation, memory pooling shines in systems where memory consumption patterns are easy to foresee.

Leveraging hardware capabilities to aid memory management may improve efficiency and predictability; this is known as hardware-assisted management. Some memory management activities may be offloaded from the CPU using techniques like hardware-supported memory protection, cache management, and direct memory access (DMA), which reduces overhead and improves speed. RTOS memory management may be optimised with the use of hardware-assisted management.

There are trade-offs and performance concerns associated with each optimisation approach and method for managing memory. Consider the potential drawbacks of dynamic memory allocation, such as fragmentation and uncertain allocation periods, despite its flexibility. Memory partitioning, on the other hand, provides certainty at the cost of potentially wasteful memory use. Memory pooling and adaptive algorithms may find a middle ground between these two extremes; however, how well they work is application and system dependent. In order to create a memory management strategy that satisfies the RTOS's unique requirements, it is crucial to thoroughly assess the costs and benefits of each approach.

Improving RTOS memory management is a challenging but essential endeavour that has an immediate effect on the dependability, predictability, and performance of the system. Researchers and engineers may create RTOS with more efficiency and robustness by learning about the specific problems with real-time memory management and then investigating different approaches and creative solutions. The purpose of this research is to examine these problems in depth and provide recommendations on how to fix them so that RTOS performance may be significantly improved by better memory management. More sophisticated and trustworthy real-time systems, up to the challenge of satisfying the demanding requirements of mission-critical applications, will be possible when these problems are resolved. To make RTOS more efficient, reliable, and predictable, and therefore suitable for a variety of real-time applications, it is recommended to use a balanced strategy that combines several memory management techniques.

Literature review

In order to guarantee fast and predictable task execution, memory management in Real-Time Operating Systems (RTOS) has been the subject of much study. Memory management presents particular difficulties for real-time operating systems (RTOS), and many research have investigated these difficulties and offered solutions. The purpose of this literature review is to provide a synopsis of the most important results and contributions to this field.

It is well-known that RTOS faces the problem of memory fragmentation. Memory fragmentation causes allocation failures even when there is enough accessible memory, and it also leads to inefficient memory utilisation, as Jones and Burns (2009) pointed out when discussing the effects of fragmentation on system performance. In order to lessen fragmentation, the research emphasised strategies including slab allocation and the buddy system.

One of the first and most often mentioned approaches to manage memory fragmentation and allocation was put out by Knuth (1973) in the form of the buddy system. A key component of the system is the ability to combine or divide memory into blocks of sizes that are powers of two. Quick allocation and deallocation times are guaranteed by this strategy, which also minimises fragmentation. Internal fragmentation may still occur, however, if the block sizes aren't very near to the specified memory sizes.

The slab allocation approach was first used in the Solaris operating system and was invented by Bonwick (1994). To save the hassle of regular allocation and deallocation, this technique pre-allocates memory blocks for identical objects. Slab allocation strikes a good mix of memory economy and allocation speed, making it ideal for settings where memory consumption patterns are predictable.

Predictable task execution is of the utmost importance in real-time operating systems, making deterministic memory management an absolute must. The significance of deterministic behaviour was emphasised in the seminal work on scheduling algorithms for real-time systems provided by Liu and Layland (1973). The Rate Monotonic Scheduling (RMS) hypothesis, created by Sha, Rajkumar, and Lehoczky (1990) and extensively used in RTOS design, is based on this. For RMS to work, it relies on predictable memory management and fixed-priority scheduling, which guarantees that jobs with high priority will finish by their due dates.

Adaptive memory management strategies that may change in response to different system loads and memory consumption patterns have recently been the subject of research. Memory allocation is dynamically adjusted depending on real-time system needs, according to an adaptive memory management methodology described by Kim and Shin (2000). Their method involves keeping an eye on the system's performance and making modifications in real-time in order to achieve a balance between memory efficiency and predictability.

Another well-researched method is memory pooling, which entails minimising allocation and deallocation cost by pre-allocating memory blocks that activities may reuse. According to studies conducted by Berger and Zorn (2002), memory pooling greatly improves allocation times while decreasing fragmentation. A system with significant memory churn and consistent consumption patterns is the ideal candidate for memory pooling, according to their research.

Memory management solutions in RTOS have also been impacted by advancements in hardware. Some methods that aim to relieve the CPU of memory management responsibilities include hardware-supported memory protection, cache management, and direct memory access (DMA). Hardware accelerators for memory management were studied by Lin and Dally (2017), who found that these methods may greatly decrease latency and increase system throughput.

Although RTOS is more often linked with non-real-time systems, garbage collection has been modified for use in RTOS. Created specifically for use in real-time systems, real-time garbage collection methods were first proposed by Bacon et al. (2003). The goal of these methods is to restore free memory while keeping the deterministic behaviour necessary for real-time operating systems (RTOS) intact.

In order to validate memory management approaches for RTOS, empirical investigations have been crucial. In order to determine the effect on system performance, evaluations often include benchmarking alternative tactics under varying workloads. The research by Kalibera et al. (2009) compared the performance of several RTOS memory management approaches and shed light on their advantages and disadvantages. The significance of taking application-specific needs into account while choosing memory management solutions was highlighted by their results.

Case studies and practical implementations provide light on how memory management approaches work in the real world. As an example, Jones et al. (1997) detailed the Mars Pathfinder project, which brought attention to the difficulties of managing memory in systems used for space exploration. The real-time operating system (RTOS) used by the mission demonstrated the difficulties and solutions encountered in a high-stakes setting by balancing predictability and flexibility via the use of dynamic allocation and fixed-size memory partitions.

Objectives of the study

- To investigate and catalog the specific challenges faced in memory management within Real-Time Operating Systems (RTOS).
- To evaluate existing strategies and techniques used to optimize memory usage in RTOS environments.
- To measure the impact of optimized memory management strategies on overall system performance, including latency, reliability, and efficiency.

Research methodology

A mixed-methods strategy is used in this study to meet the research goals fully. The first step in improving memory management in RTOS is to do a qualitative study to determine and classify the most common problems. In this stage, we investigate the particular technical obstacles and how they affect system performance by reviewing the relevant literature and conducting expert interviews. Following that, a quantitative study is carried out to assess how well different optimisation tactics mentioned in the literature and industry practices work. Various memory management approaches are tested empirically using benchmarks and simulations to quantify their influence on key performance parameters including latency, resource utilisation, and reliability as part of this quantitative study. This research intends to provide a comprehensive grasp of the difficulties of RTOS memory management and efficient methods for optimisation by integrating qualitative observations with quantitative data.

Discussion

We begin with a quick overview of the design ideas, including strategies, policies, and processes. Then, we go on to describe the technique for finding NUMA's remote memory and its findings with various test scenarios.

Design Principals

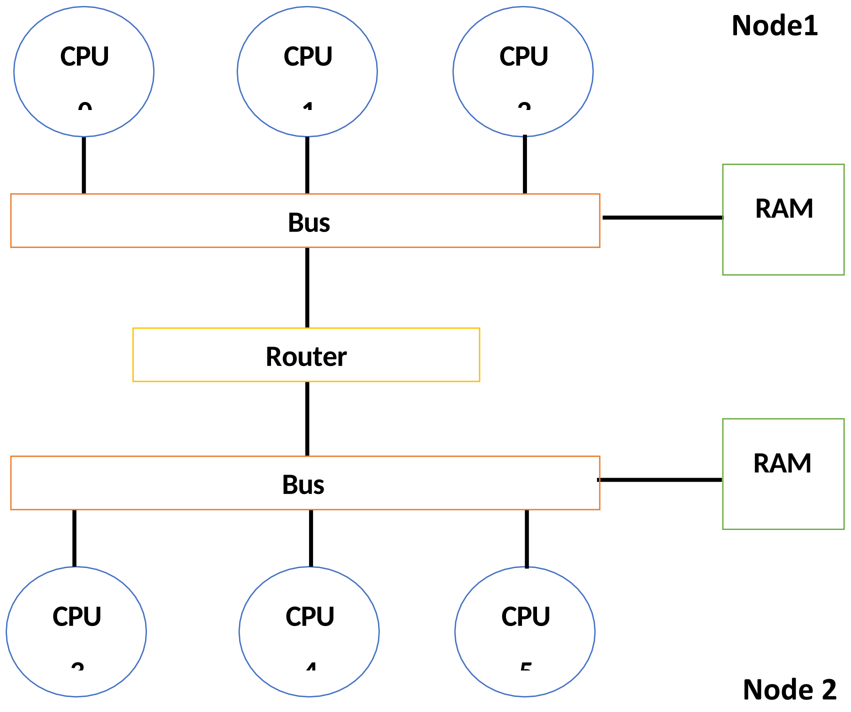

Figure 1: NUMA Architecture

In a multiprocessing architecture known as NUMA (Non-uniform memory access), the amount of time it takes for memory to be accessed is dependent on its position in relation to the processor. A NUMA-architected system allows for quicker read/write operations on a processor's local memory rather than on non-local memory, such as shared memory or the memory of other processors. Only some types of workloads can take advantage of NUMA, and that's especially true on servers where data is often closely linked to specific users or activities.
Although its design is more intricate than that of symmetric multiprocessors, NUMA represents the next generation of SMP in high-performance computing. In Figure 4.1, we can see a basic NUMA design where two nodes share memory and each node has many processors. Nevertheless, they could also possess a local memory. Alternately, you may choose from more complicated designs that use either four or eight nodes. This suggested memory allocator has been evaluated using a 4-node design; however, it may be easily scaled to 8 nodes.

Methodology for Choosing Remote Memory

For the real-time OS, figure 2 shows the memory allocator that can function on a NUMA-based architecture. There are a total of four nodes, and each of those nodes has two processors that are linked together via a bus. Each of these nodes has its own private local memory that is linked to the shared memory.

Any CPU that needs a memory block will first see whether it is accessible in its local memory; if so, it will allocate the block from there. If a local memory cannot be located, it will attempt to access the shared memory instead. In this case, the memory block will be allocated from shared memory if it is available. If the block cannot be located in shared memory either, it will seek for a less heavily loaded CPU to do the search. Consequently, locating the processor with a light load is the next stage.

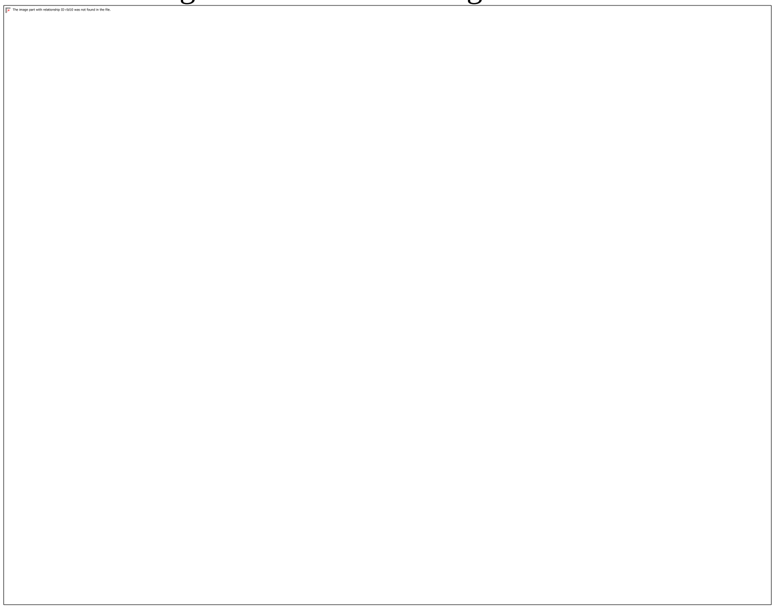
The image part with relationship ID rId10 was not found in the file.

Figure 2: Complex NUMA Structure (4 Nodes)

Conclusion

Memory fragmentation is one of the difficulties with real-time operating system (RTOS) memory management. To achieve peak speed, stability, and dependability in contexts with limited resources, RTOS-based programmes must use memory management correctly. For RTOS-based programmes to avoid system crashes and complete time-sensitive activities successfully, memory management is of the utmost importance. In doing so, it maximises performance and stability while making the most efficient use of scarce resources.

In order to optimise RTOS programmes and fulfil the stringent criteria of different industries, it is crucial to understand the issues of memory fragmentation and develop appropriate mitigation measures, according to the study's findings. To avoid system failures and guarantee the proper execution of time-sensitive operations, developers of RTOS-based applications must pay close attention to the many facets of memory management. To reduce the impact of real-time operating system (RTOS) applications, developers should familiarise themselves with the RTOS configuration file, explore all of the available options, and optimise the RTOS's use and setup.

For hard real-time systems, dynamic memory management in hardware is crucial, and the task stack size is a factor in how long it takes to allocate memory. Optimal memory utilisation and improved system performance are two of the many advantages that may be gained by using appropriate memory mapping algorithms in real-time embedded systems. Finally, optimising RTOS applications and meeting the stringent requirements of different industries requires a thorough understanding of the difficulties associated with memory fragmentation and the implementation of effective mitigation strategies. Successful RTOS-based applications rely on effective memory management.

References

- Bays, C. (1977). A comparison of next-fit, first-fit, and best-fit. Commun. ACM, 20(3): (pp. 191–192).
- Berger, E. D., McKinley, K. S., Blumofe, R. D., and Wilson, P. R. (2000). Hoard: a scalable memory allocator for multithreaded applications. SIGPLAN Not., 35(11): (pp. 117–128).
- Christian Del Rosso. (2005). Dynamic Memory Management for Software Product Family Architectures in Embedded Real-Time Systems. Fifth Working {IEEE} / {IFIP} Conference on Software Architecture (pp. 211-212)
- Dipti Diwase, Shraddha Shah, Tushar Diwase and Priya Rathod. (2012). Survey Report on Memory Allocation Strategies for Real-time Operating System in Context with Embedded Devices. International Journal of Engineering Research and Applications, Vol. 2, Issue 3, (pp.1151-1156).
- Edge, J. (2009). Perfcounters added to the mainline. http://lwn.net/Articles/336542/.
- Ferreira, T., Matias, R., Macedo, A., and Araujo, L. (2011). An experimental study on memory allocators in multicore and multithreaded applications. In Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on, (pp. 92 –98).
- Gergov, J. (1996). Approximation algorithms for dynamic storage allocation. In Algorithms — ESA '96, volume 1136, pages 52–61. Springer Berlin / Heidelberg.
- Gloger, W. (2006). ptmalloc2. "http://www.malloc.de/en/".
- Hans-Georg Eßer. (2011) Combining memory management and filesystems in an operating systems course. Proceedings of the 16th Annual {SIGCSE} Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany.
- Hasan, Y. and Chang, M. (2005). A study of best-fit memory allocators. Computer Languages, Systems & Structures, 31(1): (pp. 35 – 48).
- Hasan, Y., Chen, W.-M., Chang, J. M., and Gharaibeh, B. M. (2010). Upper bounds for dynamic memory allocation. IEEE Trans. Comput., 59(4): (pp. 468–477).
- Hewlett-Packard Corporation (2012). HP Pro-Liant DL980 G7 server with HP PREMA Architecture PREMA Architecture. Technical Whitepaper.
- Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation (2011). Advanced configuration and power interface specification.
- Hirschberg, D. S. (1973). A class of dynamic memory allocation algorithms. Commun. ACM, 16(10): (pp.615–618).
- Jane W. S. Liu. (2000). "Real-time System", 1st Edition published by Person Education.