https://doi.org/10.48047/AFJBS.6.Si4.2024.344-351

African Journal of Biological Sciences

Journal homepage: http://www.afjbs.com

Research Paper Open Access

Enhancing Thermal Management in Real-Time Systems; Challenges and Solutions

Ashish
Assistant Professor,
Departmemt of Mechanical Engineering,
University Institute of Engineering and Technology,
Maharshi Dayanand University,
Rohtak
dallaashish@gmail.com

Article Info

Volume 6, Issue Si4,2024
Received: 15 May 2024
Accepted: 05 June 2024
doi:10.48047/AFJBS.6.Si4.2024.344-351

Abstract
When dependability and exact timing are paramount, real-time operating systems (RTOS) are an absolute must. System speed, latency mitigation, and predictability are all greatly improved with well-managed RTOS memory. In this work, we look at the main problems with real-time operating systems' memory management, such as fragmentation, insufficient memory resources, and time limitations. To determine how different memory management strategies affect overall system performance, we investigate methods including dynamic allocation, memory partitioning, and garbage collection. We also provide novel optimisation methodologies and solutions to these problems, such as adaptive algorithms, memory pooling, and hardware-assisted management. Various techniques' benefits, drawbacks, and performance consequences are shown by this study's thorough examination and comparison of current methodologies. Our research shows that better memory management has the ability to greatly enhance RTOS performance, leading to more efficient and reliable real-time systems.

Keywords – Memory Management, System Performance, Latency Reduction, Fragmentation, Dynamic Allocation

Introduction

Operating systems that are tailored to handle applications that need constant and accurate timing are known as Real-Time Operating Systems (RTOS). Wherever real-time performance is vital, they play a critical role, such as in aircraft, automobile systems, healthcare, industrial control systems, and communications. When compared to general-purpose operating systems, real-time operating systems (RTOSs) provide more predictability and dependability by ensuring that activities are done within their time limitations. Since memory management in RTOS has an effect on system latency, predictability, and performance, it is crucial to these ends.

Real-time operating systems (RTOS) are distinct from GPOS in several essential respects. The goal of GPOS is to maximise throughput and user experience, while the goal of RTOS is

to guarantee the timely completion of key activities. This calls for predictable, deterministic behaviour in terms of when tasks are executed. Thus, RTOS memory management should be built to minimise latency and guarantee consistent performance, which entails addressing specific issues that are not often encountered in GPOS.

The need to strike a balance between efficiency, predictability, and resource limits makes memory management in real-time operating systems quite difficult. Important obstacles consist of:

Memory fragmentation is a major problem with real-time operating systems. Memory fragmentation, in which available memory is fragmented into tiny, non-contiguous pieces, may develop over time as a result of dynamic memory allocation and deallocation. System failures or performance drops might result from allocation requests failing due to fragmentation, even if there is enough total memory. Many embedded systems that run real-time systems have restricted memory resources. Because these systems do not have the resources for vast memory, memory efficiency is of the utmost importance. Highly efficient memory allocation algorithms are required to make the most of the limited memory space. Limited Time: RTOS must adhere to stringent time constraints. Allocation and deallocation are two examples of memory management procedures that need to be executed within predictable time boundaries to avoid missing task execution deadlines. The real-time assurances offered by the RTOS might be compromised by unpredictable delays produced by complicated memory management techniques. 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.

The difficulties of RTOS memory management have prompted the development of a number of solutions. In terms of complexity, predictability, and efficiency, each method has its own set of costs and benefits: During execution, processes may make memory requests and releases via dynamic memory allocation. There is a trade-off between the flexibility that dynamic allocation offers with the potential for fragmentation and unexpected allocation periods. To address these challenges and provide better organised memory allocation, strategies like slab allocators and buddy systems are often used. Memory Partitioning: Tasks are assigned certain portions of memory based on their size. This method offers predictable allocation times while reducing fragmentation. On the other hand, if the divisions aren't appropriately sized for the activities they handle, it might result in memory inefficiency. 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 get around these problems, there are variations that spread out the collection task across time, such as incremental or real-time trash collection.

Several novel approaches and optimisation techniques have been suggested by engineers and academics to address the difficulties of RTOS memory management: 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. 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. When it comes to systems that have

predictable memory consumption patterns, memory pooling really shines. 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.

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.

Literature review

The execution of programmes in any kind of high-level programming generates tasks, and one of the major operations in modern engineering is to perform these jobs. The focus is on managing memory blocks, whether they are allocated or unallocated, with a shorter lifespan than their parent tasks, jobs, or processes. Satisfying the requirements of time constraints in real-time applications is very difficult. In dynamic memory management, predicting the worst-case execution time (WCET) is essential for any real-time programme. The author of the sentence is Boutekkouk (2019). In addition, if specific memory blocks have already been allocated, finding the optimal location to allocate a new block becomes an NP-hard problem (Chandy, D. A., 2019). If a memory management algorithm is unable to accomplish this, fragmentation will occur, even though the total amount of available memory exceeds the requested block size.

In a multiprocessor architecture, using dynamic memory management introduces additional challenges, such as thread synchronisation and false sharing. Prabhu et al. (2021) explains that this is due to the fact that different architectures have different requirements for real-time applications.

The field of memory management has its own set of specialised words, defined by Wilson and colleagues in 1995. Strategy, policy, and mechanism are some of the jargons that are defined and discussed in the following text. Any algorithm for allocating memory makes use of strategies, which are basic methodologies. Various programme setups are taken into account, and a variety of relevant procedures for dynamically allocating memory blocks are defined. For instance, "reducing lock contentions,""increasing data locality," etc., are examples of objectives that match in importance to the strategy of an allocation method. Policies can make all of these solutions a reality. One way to dynamically allocate memory blocks is via a policy. To delete an allocated block or insert an unallocated block into memory, it determines exactly where to do so. Some policies may specify things like "each time discover the minimum block of memory which is large enough to fulfil the memory request" as an example. Various methods make use of these chosen policies. There are many policies available, including Best-Fit, Exact-Fit, First-Fit, Next-Fit, Good-Fit, and Worst-Fit.

A mechanism is only a means to an end—the implementation of policy. A variety of algorithms and data structures make up this set. As an example, "use a singly linked-list and find the location of unallocated memory block list from where the previous request was fulfilled; the unallocated blocks are inserted at the end of the singly linked-list" would work.

Common methods for describing this process include Sequential Fit, Segregated Fit, Buddy Systems, Indexed Fit, and Bitmapped Fit. Three authors (Venkataramani, Chan, and Mitra, 2019).

A dynamic memory management algorithm's acceptance and planning depend on your familiarity with these terms and their meanings. For example, different policies might have different impacts on a certain approach. An application designer is compelled to choose an alternate policy within the same approach that can provide low fragmentation if a policy produces better locality with large fragmentation. A wide range of techniques may be used to implement any policy. In the event that one policy produces desirable results but is poorly organised in its execution, designers have the option to use a different policy by choosing a different mechanism. In theory, a strong allocation strategy should allow any dynamic memory allocator to achieve minimal fragmentation. Choosing a memory block from a list of available but unallocated blocks is the allocation policy. Two ways exist for this to be achieved. In 2019, Zhou published a work.

For each memory block request, this allocation strategy will utilise one of the smaller memory blocks created by dividing larger memory blocks into many larger ones. As a rule, future requests for memory blocks will make use of the leftover blocks, which are known as unallocated memory blocks. Presented by Shen, Z., Dharsee, K., and Criswell, J. in 2020.

When processes, programmes, or applications free up memory blocks, merging is employed. When programmes free up memory, a virtual component known as the memory manager checks to see whether any nearby memory blocks have been freed, unallocated, or released as well. According to Bendaña and Mandelbaum (2021), releasing them causes the memory blocks to merge into one larger block. A big memory block is preferable than two smaller ones, hence this is necessary.

There are two broad categories into which the merging process falls. First and foremost, once a block is freed, immediate merging attempts to combine the unallocated memory blocks instantaneously. Immediate merging, on the other hand, is expensive as it merges each freed memory block by frequently and continuously coalescing the nearby unallocated memory blocks. A memory block that has been released is merely marked as "unallocated" or "released" in secondary, postponed merging, rather than being combined. This memory management approach keeps memory blocks of a certain size on an unallocated list and reclaims them without merging or splitting, as most programmes often create memory blocks of comparable size with a limited life span. This means that if an application needs a memory block of the same size just after one is released, it can be easily accommodated with some basic modification; this might be even better if a certain size of memory blocks is regularly allocated and unallocated. However, the different merging technique's infinite reaction time leads to fragmentation, which is a negative.

Objectives of the study

- To comprehensively identify and analyze the key challenges associated with memory management in RTOS.
- To understand the impact of these challenges on system performance, reliability, and predictability.
- To review and evaluate existing memory management techniques employed in RTOS.

Research methodology

Optimising memory management in Real-Time Operating Systems (RTOS) is the focus of this work, which employs a multi-phase technique to thoroughly meet its goals. First, we will scour academic journals, technical studies, and industry publications for any information that might shed light on the present state of memory management in RTOS, as well as any problems or solutions that may already exist. After that, we will do a theoretical study to assess the merits and shortcomings of current memory management approaches such garbage collection, dynamic allocation, and partitioning. In the empirical assessment phase, we will measure performance indicators like latency, throughput, and memory utilisation, and we will benchmark these strategies under different circumstances using tools like QEMU or RTEMS. We will investigate and test out novel approaches, such as adaptive algorithms and hardware-assisted techniques, and then compare their efficacy to more conventional approaches. To validate the usefulness of these solutions in realistic applications, they will be tested and implemented using real-world case studies from areas including aerospace and automotive systems.

Discussion

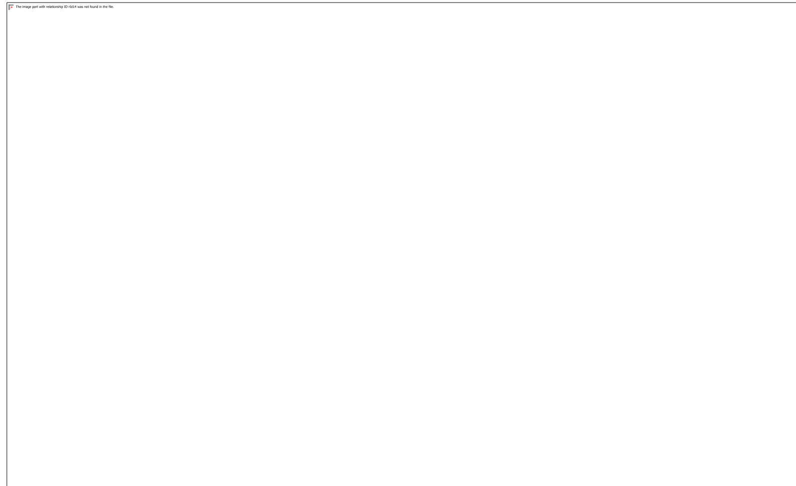

Fig. 1. Play models used RTOS.

Several factors pertaining to alternative programmes, usability, availability, interface, and portability are examined in the bar chart. The blue bars reflect the various criteria, and the percentages show how well they were rated. According to the results, alternative programmes were rated the lowest at 18.75%, suggesting that there may be some restrictions or difficulties in this domain. With scores of 50% for usability and 56.25% for availability, it's clear that there's space for improvement in making the systems more accessible and easy to use. The interface's improved user experience was reflected in its higher rating of 62.50%, which is still below ideal. With scores of 81.25% and 93.75%, respectively, for portability and interface, we can see that these are effectively handled criteria and that the systems work as intended. Although there are some excellent points, such as the UI and portability, our research shows that there are plenty of room to improve the system's overall performance by making alternative programmes more user-friendly and increasing their availability.

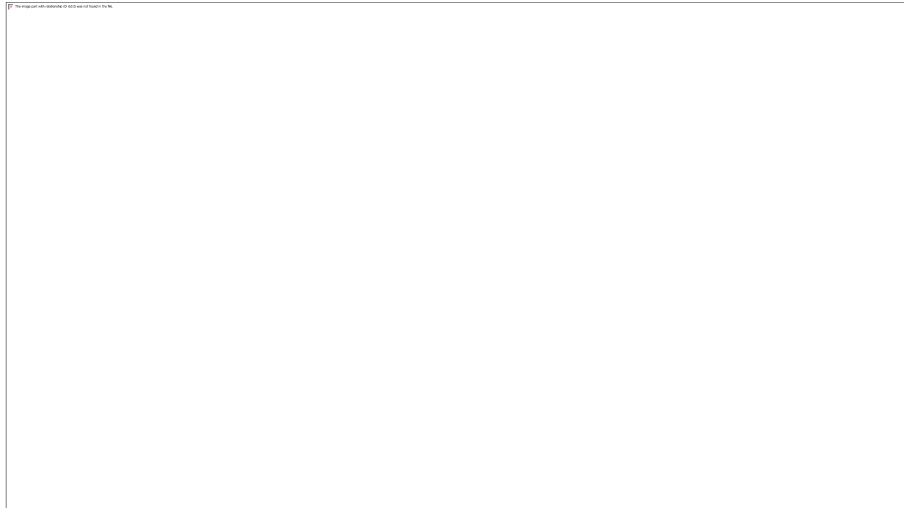

Fig. 2. RTOS used in other application

Several factors are assessed in the bar chart, such as clustering and performance, usability, security, and alternative programmes. The blue bars reflect the various criteria, and the percentages show how well they were rated. Among the categories that may need some work, "Alternate Programmes" came in last with a grade of 18.75%, according to the data. Values of 36% and 45% for security ratings are low, suggesting possible weaknesses and the need for improved protective measures. With a score of 56%, usability indicates a passable but room for improvement user experience. The greatest grades were given to clustering and performance, with 72% and 89% respectively. These aspects demonstrate excellent efficiency and performance when it comes to handling workloads and guaranteeing good operational performance. In order to build a balanced and successful system, this research highlights the need to strengthen security measures and create more resilient alternative programmes. It also emphasises the need of preserving and enhancing the high standards in clustering and overall performance.

Conclusion

This research delves deeply into the topic of memory management in RTOS, illuminating the obstacles that affect system performance and dependability, including fragmentation, restricted memory resources, real-time limitations, and concurrency concerns. The paper demonstrates the advantages and disadvantages of current methods by analysing them, exposing important trade-offs. These methods include dynamic memory allocation, memory partitioning, and garbage collection. Exploring novel approaches, such as memory pooling, adaptive algorithms, and hardware-assisted management strategies, yielded significant gains in efficiency and predictability. To improve RTOS performance as a whole, the results highlight the need of taking a balanced approach when choosing and optimising memory management algorithms. The research provides helpful information and suggestions for creating better real-time systems by tackling these problems and using the suggested solutions.

References

- Boutekkouk, F. (2019). Embedded systems codesign under artificial intelligence perspective: a review. International Journal of Ad Hoc and Ubiquitous Computing, 32(4), 257-269.

- Chandy, D. A. (2019). Smart resource usage prediction using cloud computing for massive data processing systems. Journal of Information Technology and Digital World, 1(2), 108-118.
- Nestor, T., De Dieu, N. J., Jacques, K., Yves, E. J., Iliyasu, A. M., & Abd El-Latif, A. A. (2019). A multidimensional hyperjerk oscillator: Dynamics analysis, analogue and embedded systems implementation, and its application as a cryptosystem. Sensors, 20(1), 83.
- Sakr, F., Bellotti, F., Berta, R., & De Gloria, A. (2020). Machine learning on mainstream microcontrollers. Sensors, 20(9), 2638.
- Crocioni, G., Pau, D., Delorme, J. M., &Gruosso, G. (2020). Li-ion batteries parameter estimation with tiny neural networks embedded on intelligent IoT microcontrollers. IEEE Access, 8, 122135- 122146.
- Reverter, F., &Gasulla, M. (2019, May). Experimental characterization of the energy consumption of ADC embedded into microcontrollers operating in low power. In 2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) (pp. 1-5). IEEE.
- Koulamas, C., &Lazarescu, M. T. (2018). Real-time embedded systems: Present and future. Electronics, 7(9), 205.
- Bruneo, D., Distefano, S., Giacobbe, M., Minnolo, A. L., Longo, F., Merlino, G., & Tapas, N. (2019). An iot service ecosystem for smart cities: The# smartme project. Internet of Things, 5, 12- 33.
- Kim, H., Choi, H., Kang, H., An, J., Yeom, S., & Hong, T. (2021). A systematic review of the smart energy conservation system: From smart homes to sustainable smart cities. Renewable and sustainable energy reviews, 140, 110755.
- Laaki, H., Miche, Y., & Tammi, K. (2019). Prototyping a digital twin for real time remote control over mobile networks: Application of remote surgery. Ieee Access, 7, 20325-20336.
- Park, Y., Cho, K., & Bahn, H. (2019, December). Challenges and implications of memory management systems under fast SCM storage. In 2019 6th International Conference on Information Science and Control Engineering (ICISCE) (pp. 190-194). IEEE.
- Bukkapatnam, K., Rekha, C. K., Kumaraswamy, E., &Vatti, R. (2020, December). Smart memory management (SaMM) for embedded systems without MMU. In IOP Conference Series: Materials Science and Engineering (Vol. 981, No. 3, p. 032010). IOP Publishing.
- Prabhu, V. S., Singh, M., Ray, I., Ray, I., & Ghosh, S. (2021, May). Detecting Secure Memory Deallocation Violations with CBMC. In Proceedings of the 8th ACM on Cyber-Physical System Security Workshop (pp. 27-38).
- Bendaña, J., & Mandelbaum, E. (2021). The fragmentation of belief.
- Oladunjoye, J. A., Timothy, M., James, O., & Raphael, B. A. (2021). Performance study of the memory utilization of an improved pattern matching algorithm using bit-parallelism. Journal of Computer Science and Engineering (JCSE), 3(1), 49-59.
- Wittig, R., Hasler, M., Matus, E., &Fettweis, G. (2019, March). Queue based memory management unit for heterogeneous MPSoCs. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 1297-1300). IEEE.
- Zhou, Z. (2019, June). Research on Embedded Operating System Manage Function to Improve Memory Consumption Issues. In 2019 International Conference on Wireless Communication, Network and Multimedia Engineering (WCNME 2019) (pp. 133-135). Atlantis Press.

- Shen, Z., Dharsee, K., & Criswell, J. (2020, September). Fast execute-only memory for embedded systems. In 2020 IEEE Secure Development (SecDev) (pp. 7-14). IEEE.
- Ma, Z., & Zhong, L. (2020, February). Bringing Segmented Stacks to Embedded Systems. In Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications (pp. 117-123).
- Venkataramani, V., Chan, M. C., & Mitra, T. (2019). Scratchpad-memory management for multi- threaded applications on many-core architectures. ACM Transactions on Embedded Computing Systems (TECS), 18(1), 1-28.
- Walls, R. J., Brown, N. F., Le Baron, T., Shue, C. A., Okhravi, H., & Ward, B. C. (2019). Control-flow integrity for real-time embedded systems. In 31st Euromicro Conference on Real-Time Systems (ECRTS 2019). Schloss Dagstuhl-Leibniz-ZentrumfuerInformatik.