IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

ADVANCED MEMORY MANAGEMENT FOR EMBEDDED SYSTEMS WITH LIMITED RESOURCES

Mohamed Ashraf. K[1], Jamal Kutty. K[2]

[1]Lecturer [2] Lecturer

[1]Department of Electronics Engineering, IPT And Government Polytechnic College Shoranur, Palakkad, Kerala, India

[2]Department of Electronics Engineering, AKNM Government Polytechnic College Tirurangadi, Malappuram, Kerala, India

Abstract: Dynamic memory management is vital in embedded systems as it allows efficient allocation and deallocation of memory blocks, optimizing resource utilization. It provides adaptability to changing runtime requirements, enabling embedded systems to handle varying data sizes and structures effectively. Furthermore, dynamic memory management plays a crucial role in memory optimization, minimizing fragmentation and maximizing memory usage for improved system performance. This paper investigates the potential of smart and programmable memory management methods to enhance performance and efficiency in MMU-less embedded systems, addressing the challenge of limited space and the increasing use of IoT. The current approach has achieved an improvement in allocation speed, around 3-4 times faster, with a slight trade-off in deallocation speed. Additionally, the outcome includes achieving 0 fragmented memory. Additionally, the paper provides detailed algorithms for implementing memory allocation, deallocation, and defragmentation processes directly within the application software that is already in use. These algorithms offer practical and feasible solutions for managing memory efficiently without requiring significant changes to the existing software infrastructure.

Index Terms **- Internet of Things, Memory Management Units, Dynamic Memory Allocation Schemes, Embedded systems.**

I. INTRODUCTION

Embedded systems are specialized computer systems designed to perform specific tasks or functions within larger systems. They are typically integrated into various devices and appliances, ranging from household appliances like washing machines and smart thermostats to complex industrial machinery and automotive systems [1]. What sets embedded systems apart is their dedicated functionality, real-time operation, and often limited resources such as processing power [2], memory, and energy. These systems are optimized for efficiency and reliability, often requiring real-time responses to external events or inputs. Embedded systems [3] are designed to operate seamlessly, often running autonomously without the need for user intervention, and are essential components in modern technology, contributing to the automation, control, and intelligence of countless devices and industries.

The development of embedded systems involves a combination of hardware and software design. The hardware components are typically microcontrollers [4] or specialized chips that provide the necessary processing power and interfaces to interact with external devices or sensors. The software running on these systems is specifically tailored to the embedded application, with a focus on efficiency, real-time performance, and reliability [5]. Embedded software is often written in low-level programming languages like C or assembly language to maximize control over hardware resources. Additionally,

embedded systems often have strict constraints in terms of power consumption, size, and cost, requiring designers to carefully optimize both the hardware and software to meet these constraints [6]. The field of embedded systems continues to evolve rapidly, driven by advancements in technology, miniaturization, connectivity, and the demand for smart and interconnected devices in various industries [7].

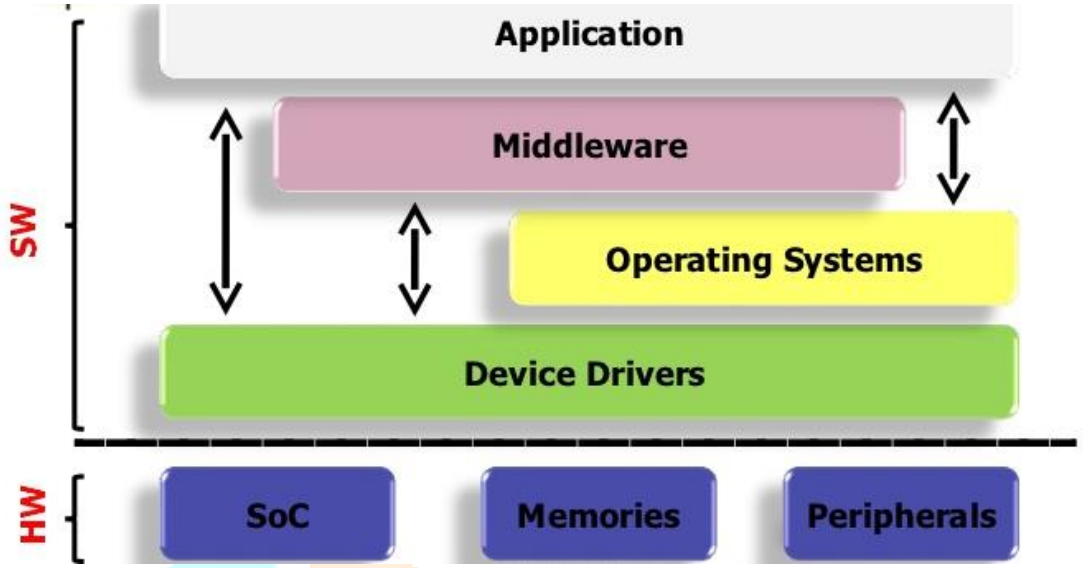

Figure 1: architecture of embedded systems

IoT extends the realm of the internet beyond traditional computing devices to include everyday objects, creating a vast ecosystem of smart devices [8] that communicate and interact with each other. By connecting these devices, IoT enables the seamless exchange of data and information, leading to improved efficiency, automation, and convenience in various domains, including smart homes [9], healthcare, agriculture, transportation, and industrial applications. IoT holds great potential to transform industries and enhance our lives by enabling real-time monitoring, remote control [10], predictive analytics, and intelligent decision-making based on the vast amounts of data generated by interconnected devices. However, it also presents challenges such as data privacy, security, and interoperability that need to be addressed to fully harness its benefits and ensure the responsible and secure deployment of IoT solutions.

1.1 Memory allocation Systems

Memory management systems are essential components of computer operating systems that are responsible for managing the allocation, deallocation, and utilization of a computer's memory resources [11]. These systems ensure efficient memory usage and play a crucial role in optimizing system performance.

In a memory management system, the available memory is divided into smaller units or blocks, which are allocated to different processes or applications as needed [12]. The system keeps track of which memory blocks are in use and which are available, ensuring that processes are provided with the necessary memory resources to execute their tasks. When a process no longer requires a particular memory block, the memory management system deallocates that block [13], making it available for future allocation. This process of allocation and deallocation is carefully managed to prevent memory leaks and ensure that memory resources are utilized effectively.

Additionally, memory management systems implement memory protection mechanisms to prevent unauthorized access or modification of memory regions. They enforce access controls to ensure that processes can only access memory areas that they are authorized to access, contributing to the security and integrity of the system. Memory management systems also handle memory fragmentation [14], where free memory becomes scattered in small fragments. Techniques such as compaction or memory block reorganization may be employed to consolidate free memory and reduce fragmentation, improving memory utilization [15]. Overall, memory management systems are vital for efficient memory allocation, protection, and optimization, enabling computer systems to efficiently run multiple processes and applications simultaneously.

II. LITERATURE REVIEW

For heterogeneous embedded systems, authors suggested the Q-MMU in this paper by Wittig et al [16]: a fully reconfigurable, shared, low-latency memory management unit. Comparing the introduced passive conflict detection to current work, the critical route is reduced, improving system performance overall. Two new dynamic memory management strategies for embedded systems are proposed by Zhou et al [17] to address the requirements of embedded real-time applications. The algorithm in this work distinguishes between the use of large memory and the use of tiny memory. This paper presents a novel idea and its implementation, focusing on how to swiftly realise memory allocation, release, and recovery as well as how to improve memory utilization.

Shen et al [7] introduced PicoXOM, a quick and innovative XOM system for ARMv7-M and ARMv8-M devices that makes use of the MPU and DWT unit of ARM. On the BEEBS and CoreMark-Pro benchmark suites as well as five real-world applications, PicoXOM experiences an average performance overhead of 0.33% and an average code size overhead of 5.89%.

Ma et al [8] contend that segmented stacks, a theory that was looked into and later rejected for systems with virtual memory, can address both issues for embedded software.

Venkataraman et al [9] provided Coordinated Data Management (CDM), a compile-time system that detects shared/private variables automatically and moves them to appropriate on-chip or off-chip memory with replication (if necessary), taking NoC contention into account. Walls et al [10] proposed RECFISH, a framework for delivering CFI guarantees on ARM Cortex-R devices running basic real-time operating systems. We offer strategies for safeguarding processes, runtime structures, and compiled ARM binaries with CFI protection. We empirically assess RECFISH's effects on real-time systems' performance.

III. MATERIALS AND METHODS

The Memory Management Scheme employed in this study utilizes two primary approaches to achieve improved memory handling. These approaches form the foundation of the scheme and are instrumental in enhancing the overall management and utilization of memory resources.

- Advanced Two-Level Segregated Fit Memory Allocation Scheme (E-TLSS)
- Dynamic Memory Block Movement for Efficient Free Memory Slot Creation

Considering that the smallest available memory block in this technique is 16 bytes, including metadata for block allocation, it is essential to note that embedded systems face severe space constraints. Therefore, the ability to add a significant amount of additional memory to the system is limited. Consequently, it becomes crucial to promptly release memory and avoid holding onto it for prolonged periods.

- Memory Allocation
- Deallocate memory

3.1 Smart Memory Management

Smart memory management is crucial in embedded systems where resources, including memory, are often limited. To optimize memory usage, several strategies can be employed. Firstly, developers should conduct memory profiling to understand the memory requirements of their software. By identifying memory-intensive functions and data structures, they can focus on optimizing those areas. Additionally, minimizing the memory footprint is essential. This involves using efficient algorithms and data structures that require less memory, avoiding unnecessary memory allocations and deallocations, and favouring static memory allocation over dynamic allocation whenever possible. Techniques like memory pooling, which pre-allocates a fixed block of memory and assigns and recycles memory blocks from that pool, can be employed to reduce fragmentation and overhead. Furthermore, optimizing stack usage, utilizing non-volatile memory for infrequently modified data, optimizing memory access

patterns, and implementing memory leak detection mechanisms contribute to efficient memory management in embedded systems.

The memory organization in this approach involves two types of blocks: Pointer Block and Data Block. Multiple pointers are used to create complex data structure. A buffer is kept between Pointer Block and Data Block to accommodate additional pointers. If the buffer becomes full, the subsequent Data Block is utilized. Each allocated block consists of two parts: metadata, which contains a header, and the payload, which holds the actual data. In the SaMM approach, the metadata section is enhanced with additional information, such as the end location and size of the last data package.

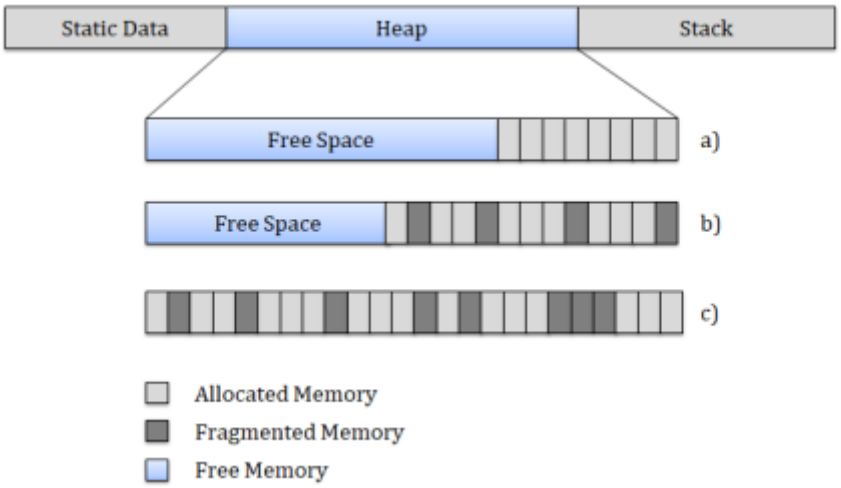

Figure 2: system memory and diverse fragmentation level

3.2 Allocation Process Algorithm

The process utilized in this approach is straightforward yet highly effective. It leverages the margin space to allocate payloads efficiently, either in the front or the back, depending on the availability of sufficient margin space. However, if there is insufficient free space in fragmented blocks, the margin, or the free memory, the memory allocation process concludes unsuccessfully and returns a null value.

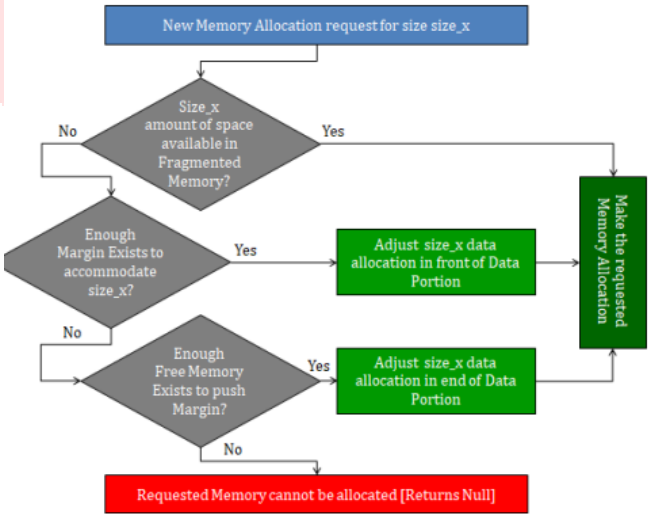

Figure 3: memory allocation as per SaMM

3.3 Algorithm for Deallocation

A separate defragmentation process, running concurrently with allocation cycles, can then handle the grouped fragmented cells. This distinct handling of deallocation and defragmentation contributes to multitasking capabilities in real-time operating systems, enabling efficient memory management in multitasking applications.

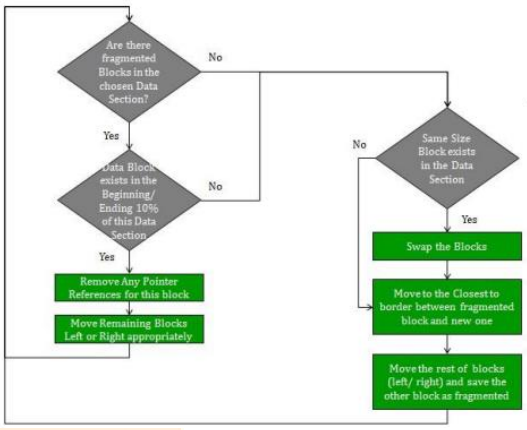

Figure 4: deallocation of memory through smart memory management

3.4 Defragmentation Process Algorithm

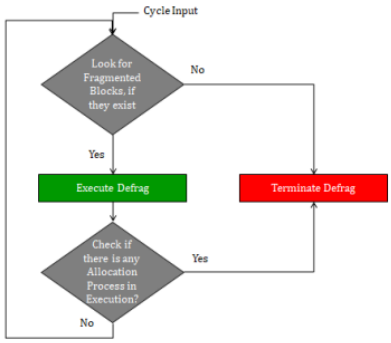

Figure 5: defragmentation of memory through smart memory management

IV. CONCLUSION

In recent years, the increasing usage of IoT systems has sparked numerous discussions about addressing the challenge in allocation of memory for embedded systems. The industry has been actively seeking stable execution of embedded system, as previous memory allocation schemes often led to fragmentation and unstable operations. This paper presents a solution that effectively eliminates fragmentation by consolidating fragmented blocks into a single location, enabling a streamlined defragmentation process. While currently most suitable for small-scale operations, there are opportunities to expand this solution to a wider range of devices, making it a cost-effective and universal smart memory allocation scheme for embedded systems in the future.

REFERENCE

1. Boutekkouk, F. (2019). Embedded systems codesign under artificial intelligence perspective: a review. International Journal of Ad Hoc and Ubiquitous Computing, 32(4), 257-269.
2. 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.
3. 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.
4. Sakr, F., Bellotti, F., Berta, R., & De Gloria, A. (2020). Machine learning on mainstream microcontrollers. Sensors, 20(9), 2638.
5. 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.
6. 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.
7. Koulamas, C., & Lazarescu, M. T. (2018). Real-time embedded systems: Present and future. Electronics, 7(9), 205.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. Prabhu, V. S., Singh, M., Ray, I., Ray, I., & Ghosh, S. (2022, May). Detecting Secure Memory Deallocation Violations with CBMC. In Proceedings of the 8th ACM on Cyber-Physical System Security Workshop (pp. 27-38).
14. Bendaña, J., & Mandelbaum, E. (2021). The fragmentation of belief.
15. Oladunjoye, J. A., Timothy, M., James, O., & Raphael, B. A. (2022). 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.
16. 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.
17. 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.
18. 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.
19. Ma, Z., & Zhong, L. (2023, February). Bringing Segmented Stacks to Embedded Systems. In Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications (pp. 117-123).

20. 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.
21. 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-Zentrum fuer Informatik.