Dipti Diwase, Shraddha Shah, Tushar Diwase, Priya Rathod / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp.1151-1156

Survey Report on Memory Allocation Strategies for Real Time Operating System in Context with Embedded Devices

Dipti Diwase*, Shraddha Shah, Tushar Diwase***, Priya Rathod******

*Department of Computer Science & Engineering, G. H.R.C.E., Nagpur, India
**Department of Electronics & Telecommunication, G. H.R.C.E., Nagpur, India
***Department of Electronics & Communication, D.B.A.C.E.R, Nagpur, India
****Department of Electronics & Telecommunication, G. H.R.C.E., Nagpur, India

ABSTRACT

An embedded system is a computer system designed for specific control functions or for any dedicated application, often with real-time computing constraints. Real time operating system (RTOS) is specially used to meet the real time constraint and to support the sophisticated facilities provided by embedded system. While designing any embedded system memory management is an important issue. Embedded system developers commonly implement custom memory management facilities on top of what the underlying RTOS provides. That's why understanding memory management is therefore an important aspect. In this paper we are dealing with different memory management strategies and algorithm that are used in real time operating system in order to improve the performance of the intended application. Firstly we have compared the static and dynamic memory management strategy and then we consider different algorithms that can be used to implement the dynamic memory management in real time operating system.

Keywords - **Real Time Operating System, Memory allocation, sequential fit, Buddy Allocator, Indexed fit, Two Level Segregated Fit algorithm, Smart Memory Allocator.**

I. INTRODUCTION

Embedded systems have become an integral part of daily life. Be it a cell phone, a smartcard, a music player, a router, or the electronics in an automobile - these systems have been touching and changing modern lives like never before. An embedded system is a combination of computer hardware, software, and additional mechanical or other technical components, designed to perform a dedicated function. Sophisticated facilities provided by an Embedded System are shown in figure 1. Most of the embedded systems need to meet real-time computing requirements. Thus real time operating system (RTOS) is one of the major building blocks of the Embedded Systems. Along with the real time requirement many times embedded system is having limited memory and processing power. Therefore, understanding Memory management an important aspect.

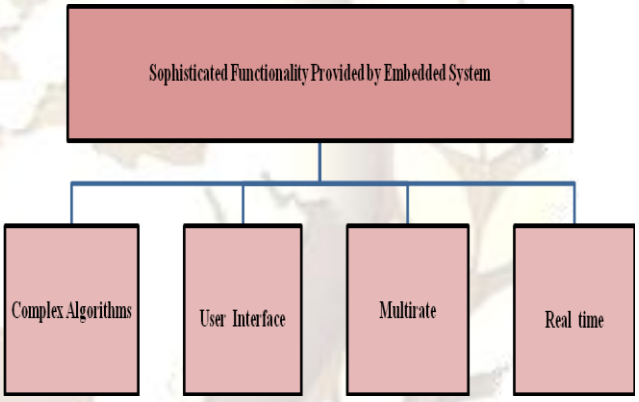

Figure. 1: Facilities provided by Embedded System

Before moving towards the memory management concept; let's have a look at what is RTOS and what kind of features it has. RTOS stands for "Real Time Operating System" which is nothing but an operating system that supports real-time applications by providing correct result within the deadline. Real Time Operating System can be categorize as hard real time system and soft real time system based on how strictly it follows the task completion deadline. The basic difference between general operating system and real time operating system is given in Table I. Differentiation is made on different aspects like determinism, preemptive kernel, priority inversion, task scheduling, latency is priorly defined or not etc. RTOS's are designed in such a way that the timing of events is highly deterministic.

Table I: Difference between General Purpose OS and RTOS

	RTOS	General Purpose OS
Determinism	Deterministic	Non-deterministic
Preemptive kernel	All kernel operations are preemptable.	Not Necessary
Priority Inversion	Have mechanisms to prevent priority inversion	No such mechanism is present
Task Scheduling	Scheduling is time based	Scheduling is process based
Latency	Have their worst case latency defined	Latency is not of a concern Purpose OS
Application	Typically used for embedded applications	General purpose OS is used for desktop PCs or other generally purpose PCs

RTOS is key to many embedded systems and provides a platform to build applications [14]. Figure 2 shows the overview of embedded system meant for real time applications.

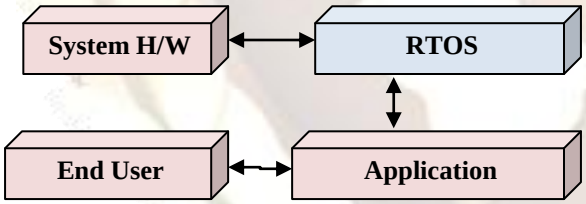

Figure 2: overview of embedded system meant for real time applications

RTOS provides following features [14]:

- Synchronization: Synchronization is necessary for real time tasks to share mutually exclusive resources. For multiple threads to communicate among themselves in a timely fashion, predictable inter-task communication and synchronization mechanisms are required.
- Interrupt Handling: Interrupt Service Routine (ISR) is used for interrupt handling. Interrupt latency is defined as the time delay between the occurrence of an interrupt and the running of the corresponding Interrupt Service Routine (ISR).
- Timer and clock: Clock and timer services with adequate resolution are vital part of every real-time operating system.
- Real-Time Priority Levels: A real-time operating system must support real-time priority levels so that when once the programmer assigns a priority value to a task, the operating system does not change it by itself.
- Fast Task Preemption: For successful operation of a real-time application, whenever a high priority critical task arrives, an executing low priority task should be made to instantly yield the CPU to it.
- Memory Management : Real-time operating system for large and medium sized application are expected to provide virtual memory support, not only to meet the memory demands of the heavyweight real-time tasks of an application, but to let the memory demanding non-real-time applications such as text editors, e-mail etc. An RTOS uses small memory size by including only the necessary functionality for an application while discarding the rest [14].

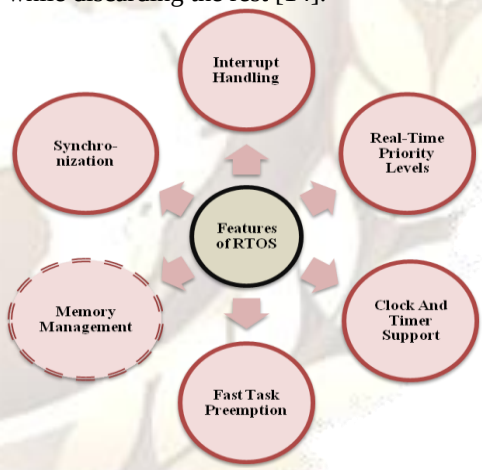

Figure 3: Features Provided By RTOS

In section II we will discuss the memory management in real time operating system, section III is dedicated to the dynamic memory management and different algorithms available for it.

II. MEMORY MANAGEMENT

Basically two types of memory managements are provided in RTOS stack and heap. The stack management is used during context switching for

Task Control Blocks while heap management deals with memory other than memory used for program code and it is used for dynamic allocation of data space for tasks. Management of this memory is called heap management [6][12].

Real time operating system also supports static and dynamic memory management [7]. Table II. describe the basic difference between static memory management and dynamic memory management.

Table II. Difference between static and dynamic memory management

	Static Memory management	**Dynamic Memory Management**
1	Memory is allocated at compile or design time.	Memory is allocated at run-time.
2	Here, memory allocation is a deterministic process that means required memory for particular process is already known and once memory is allocated no changes can be during at run time	Dynamic memory management requires memory manager to keep track of which parts of the memory in use and which parts are not. This help to allocate memory to processes when they need it and to deallocate it when they are done.
3	No memory allocation or deallocation actions are performed during execution.	Memory Bindings are established and destroyed during the Execution.
4	More memory Space required.	Less memory Space required.

The above figure 4, shows the classification of Memory Management Techniques. Dynamic memory management can be classified into two categories:

1. Manual memory management

In manual memory management, the programmer has direct control on process of memory is allocation and recycling. Usually this is either by explicit calls to heap management functions or by language constructs that affect the stack.

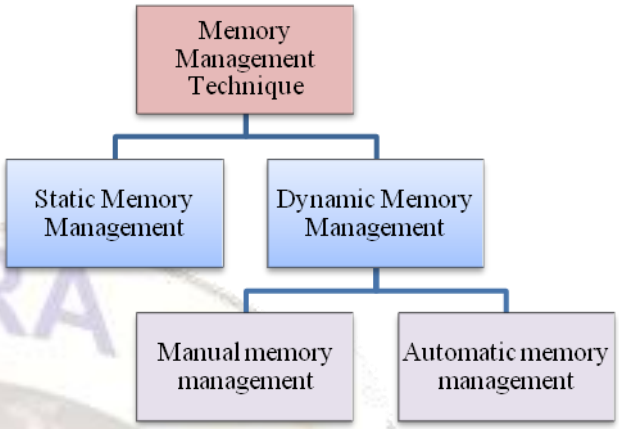

Figure 4: Classification of Memory Management Technique

Advantage:

- Manual memory management is easier for programmers to understand and use.

Disadvantage:

- Memory management bugs are common when manual memory management is used.

2. Automatic memory management

In case of Automatic memory management, automatic memory managers (often known as garbage collectors) usually do their job by recycling blocks that are unreachable from program variables.

Advantage:

- Automatic memory management eliminates most memory management bugs.

Disadvantage:

- Sometimes automatic memory managers fail to deallocate unused memory locations causing memory leaks.
- Automatic memory mangers usually consume a great amount of the CPU time and usually have nondeterministic behavior.

In this paper, we focus only on supporting manual memory management techniques suitable for real-time systems.

III. DYNAMIC MEMORY ALLOCATION ALGORITHMS

The main motive of DMA algorithms is to allow the application to access blocks of memory from a pool of free memory blocks [2][5][8]. Number of Dynamic Memory Allocation algorithms has been developed till now and most of them are based on the strategy in which link all free blocks are linked

together to form one or more list containing free memory blocks so that free memory can be made available for the another process in execution. Use of one free list to keep track of free memory blocks is the simplest method; however, algorithms utilizing multiple free lists often results in better performance specially for the system that requires good and reliable timing response at the cost of high memory footprint [2][5][8]. But still DM Allocators usage unreliable for embedded systems due to its unbounded long response time or its inefficient memory usage. The DM allocators for embedded systems should be implemented in their constrained operating systems by considering the limited available resources. Hence, the allocators should provide both the features such as optimum memory footprint [1].

The following are some DMA Algorithms that are used so far.

1. Sequential Fit

Sequential fit is the most basic algorithm which uses a single linear list of all free memory blocks called a free list. Free blocks are allocated from this list in one of four ways [13]:

- First fit: the list is searched from the beginning, returning the first block large enough to satisfy the request;
- Next fit: the list is searched from the place where the last search left off, returning the next block large enough to satisfy the request;
- Best fit: the list is searched exhaustively, returning the smallest block large enough to satisfy the request.
- Worst fit : the list is searched; returning largest available free block .[13]

Advantage:

- Simple to understand and easy to apply.

Disadvantage:

- As memory available for allocation becomes large, the search time becomes large and in the worst case can be proportional to the memory size.
- As the free list becomes large, the memory used to store the free list becomes large; hence the memory overhead becomes large
- Sequential Fit is not a good strategy for RTSs because it is based on a sequential search, whose cost depends on the number of existing free blocks.
- This search can be bounded but the bound is not normally acceptable.

2. Buddy allocator System

A buddy allocator uses an array of free list s, one for each allowable block size (e.g., allowable block size is a power of 2 in the case of binary buddy system) [10]. The buddy allocator rounds up the requested size to an allowable size and allocates from the corresponding free list. If the free list is empty, a larger block from another free list is selected and split. A block may only be split into a pair of buddies (of the same size in case of binary buddy system). A block may be coalesced only with its buddy, and this is possible only if the buddy has not been split into smaller blocks [10].

Advantage:

- The advantage of a buddy system is that the buddy of a block being freed can be quickly found by a simple address computation.

Disadvantage:

- The disadvantage of a buddy system is that the restricted set of block sizes leads to high internal fragmentation, as does the limited ability to coalesce.

3. Indexed Fit

Indexed fits are a class of allocation mechanisms that use an indexing data structure, such as a tree or hash table, to identify suitable free blocks, according to the allocation policy [10][13].

Advantage

- This strategy is based on the use of advanced structures to index the free memory blocks thus provide better performance.

4. Bitmapped Fit

Bitmapped Fits are a class of allocation mechanisms that use a bitmap to represent the usage of the heap. Each bit in the map corresponds to a part of the heap, typically a word, and is set if that part is in use [10][11].

Advantage:

- In Bitmapped Fit mechanism, the search time is proportional to the bitmap size which result in a small memory overhead.

5. Two Level Segregated Fit Algorithms

Two level segregated fit solves the problem of the worst case bound maintaining the efficiency of the allocation and deallocation operations allows the reasonable use of dynamic memory management in real-time applications. The two level segregated fit

algorithm provides explicit allocation and deallocation of memory. There exists an increasing number of emerging applications using high amounts of memory, such as multimedia systems, video streaming, video surveillance, virtual reality, scientific data gathering, or data acquisition in control systems.

In order to meet these constraints and requirements, two level segregated fit has been designed with guidelines like immediate merging, splitting threshold for optimizes the memory usage and Good-fit strategy which tends to produce the lowest fragmentation on real workloads, compared to other approaches such as first fit approach. To implement a good-fit strategy two level segregated fit algorithm uses a segregated fit mechanism. The basic segregated fit mechanism uses an array of free lists, with each array holding free blocks within a size class. In order to speed up the access to the free blocks and also to manage a large set of segregated lists (to reduce fragmentation) the array of lists has been organized as a two-level array. The first-level array divides free blocks in classes that are a power of two apart and the second-level sub-divides each first-level class linearly, where the number of divisions can be determined by the user. Each array of lists has an associated bitmap used to mark which lists are empty and which ones contain free blocks. This bitmap helps in identifying free blocks for satisfying memory requests. Inside the block itself, Information regarding each block is stored [3][4].

Advantage:

- In Two Level Segregated Fit, there is extremely low overhead per allocation.

6. Smart Memory Allocator Algorithm

Smart Dynamic Memory Allocator is a new allocation style which allows designing a custom dynamic memory allocation mechanism with reduced memory fragmentation and superb response time. Here, the long-lived memory blocks are handled independently form short-lived memory blocks by the allocator. The direction of growing of heap is different for both types of memory blocks. The short-lived blocks are allocated from heap in the direction of bottom to top and the long-lived blocks are allocated from top to bottom. The used space of the heap grows from both sides. Initially whole heap memory is free and there is only one free block each for short-lived and long-lived memory pool. The heap space is initially divided into two blocks (with no physical Boundary) in a predefined proportions, one for short-lived and the other for long-lived objects. As the heap grows from both sides, the boundary (virtual boundary) between short-lived memory pool and long-lived memory pool can be easily modified according to the run time memory requirements. For instance, let the heap size is 400 bytes, each short and long-lived memory pool has only one free block of size 200-bytes. In response to a memory block request of size 16-bytes (for instance, short-lived memory), the bottom 16-bytes of the free block corresponding to short-lived blocks is allocated for the request. In distinction, upon a memory request of block size 64-bytes (assume, long-lived object), the top 64-bytes of the free block corresponding to long-lived memory blocks is allocated for the request.

A new smart dynamic memory allocator particularly for embedded systems have limited memory and processing power. The smart allocator predict the short-lived objects and allocates those objects on one side of the heap memory and remaining objects on other side of the heap memory for effective use of memory footprint. The allocator reduces the memory fragmentation by using directional adaptive allocation based on the predicted object lifetimes the lifetime can be predicted using a mixture of block size and the number of allocation events [1].

The allocator is implemented with improved multilevel segregated mechanism by means of lookup tables and hierarchical bitmaps that guarantee very good response time and dependable timing performance.

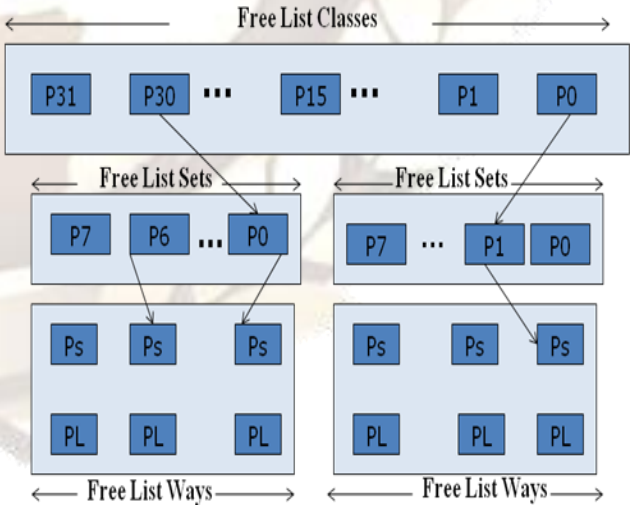

Figure 5: Organization of Free-block Lists [1]

A large number of free-lists are used by the smart memory allocator. Where, each Free-list keeps the

free blocks of size inside a predefined range and also the blocks belongs to some specific block-type. As shown in above figure 3, the free-lists are organized into 3 level arrays. Managing the short-lived memory blocks independently from long-lived memory blocks effectively without causing extra search overhead, the free-lists are controlled into three-level arrays. The first-level divides the free-lists into free-list-classes (for example, the block sizes between (215-1) and 214 comes beneath 14th free-list-class), each class of free-lists are further subdivided into diverse free-list-sets with each set keeping the free blocks of size within a predefined range (the dynamic rage of class is linearly subdivided among all the free-list-sets). Finally, each set of free-list-sets is divided into two free-list-ways, one equivalent to short-lived and other equivalent to Long-lived [1].

7. Comparison between Different Memory Allocation Algorithms

In the above table III, we have done comparison between various memory management algorithms on the basis of resulting fragmentation, response time and memory footprint usage

Table III: Comparison between Different Memory Allocation Algorithms

		Parameters		
		Fragmentation	*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	Excellent

IV. CONCLUSION

Memory Allocation is one of the most crucial components RTOS based Embedded System. There are different memory allocation algorithms are available. The major challenges of memory allocator are minimizing fragmentation, providing a good response time, and maintaining a good locality among the memory blocks. This paper has presented various memory allocators available which aimed at addressing the major challenges of memory allocation, along with their merits and demerits.

REFERENCES

[1] Ramakrishna M, Jisung Kim, Woohyong Lee and Youngki Chung, “Smart Dynamic Memory Allocator for embedded systems”, in proceedings of 23rd International Symposium on Computer and Information Sciences, ISCIS '08, 27-29 Oct. 2008

[2] I. Puaut, "Real-Time Performance of Dynamic Memory Allocation Algorithms," in Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02), June 2002

[3] M. Masmano, I. Ripoll and A. Crespo, “TLSF: A new dynamic memory allocator for real-time systems,” in proceedings of 16th Euromicro Conference on Real-Time Systems, Catania, Italy, July 2004, pages 79-88.

[4] XiaoHui Sun, JinLin Wang, xiao chan, “An Improvement of TLSF Algorithm”.

[5] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, “Dynamic Memory Allocation: A Survey and Critical Review,” In Proceedings of the Memory Management International Workshop IWMM95, Sep 1995

[6] MS Johnstone ,”Fragmentation Problem: Solved ?”In Proc. of the Int. Symposium on Memory Management (ISMM’98), Vancouver, Canada. ACM Press

[7] Robart L. Budzinski, Edward S. Davidson, “A Comparison of Dynamic and Static Virtual Memory Allocation Algorithms” IEEE Transactions on software Engineering, Vol. SE-7, NO. 1, January 1981.

[8] Takeshi Ogasawara,”An Algorithm with Constant Execution Time for Dynamic Memory Allocation”, in proceedings of Second International Workshop on Real-Time Computing Systems and Applications, 25-27 Oct 1995,Pg No. 21 – 25.

[9] David R. Hanson, “Fast allocation and deallocation of memory based on object life times,” Software-Practice and Experience, Vol. 20(1), Jan 1990 pp. 5-12.

[10] Mohamed A. Shalan, “Dynamic Memory Management for Embedded Real-Time Multiprocessor System On a Chip”, A Thesis in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy from School of Electrical and Computer Engineering ,Georgia Institute of Technology, November 2003.

[11] Kenny Bridgeson, “Information System Management”, (published by Lotus Press, Page no. 36-38)

[12] S. Baskiyar, Ph.D. and N. Meghanathan, “A Survey of Contemporary Real-time Operating Systems” Informatica 29 (2005),Pg No.233–240

[13] The Memory Management Glossary,” http://www.memorymanagement.org/glossary/b.html”

[14] Jane W. S. Liu, “Real-time System”, (published by Person Education, Page no. 20-40).