ISTANBUL UNIVERSITY –
JOURNAL OF ELECTRICAL & ELECTRONICS ENGINEERING

YEAR : 2005 (1435-1441)
VOLUME : 5
NUMBER : 2

DYNAMIC MEMORY ALLOCATOR ALGORITHMS SIMULATION AND PERFORMANCE ANALYSIS

[1]Fethullah Karabiber **[2]Ahmet SERTBAŞ** **[3]A.Halim ZAIM**

[1,2,3] Istanbul University, Engineering Faculty, Computer Engineering Department
34320, Avcilar, Istanbul, Turkey

[1]E-mail: fetullah@istanbul.edu.tr [2]E-mail: aserbas@istanbul.edu.tr

[3]E-mail: ahzaim@istanbul.edu.tr

ABSTRACT

In this work, Dynamic memory allocation that is critical issue in the design of computer systems is examined. During first part of this project, improved software techniques separated into three different categories such as Bit map, Linked list techniques, Buddy systems are analyzed. For this purpose, algorithms are simulated by using C++ programming language. In the simulations, internal, external and total fragmentation, and process times are used as performance criteria.

Keywords: *Memory allocation, fragmentation, simulaton, programming*

1. INTRODUCTION

Speed and performance are two of the most performance measure in computer systems at today's world. The trend of using fast processors in computer systems triggers improving high performance units. Dynamic Memory Allocation is used in arranging effectively the memory that is indispensable module of computer systems.

Memory manager performs memory allocation process. It is reported that dynamic memory management consumes 23%-38% of the time in six allocation-intensive C programs running on 17-SPECmarks SPARC architecture with 80 MB of memory [1]. Object-oriented programs have a very high object creation rate and, therefore, the speed of memory allocation is crucial for improving the system performance.

Until now, improved software techniques by using various data structures are separated into three different categories: Bitmap, Linked List techniques, Buddy Systems.

Bitmap implementation use a bitmap, where each bit represents some portions of memory and indicates whether it is free or occupied.

The most popular method in dynamic memory allocation techniques is Linked Lists. First Fit, Best Fit, Worst Fit and Next Fit Linked List techniques are well known and frequently used. Although Memory usage of these algorithms is good, list structure leads to decrease running speed of these algorithms. Each memory request (malloc function in C programming language) causes to search all list sequentially. The memory manager scans among the list until it finds blocks that are big enough. The hole is then

Received Date : 10.12.2004
Accepted Date: 27.03.2005

broken into two pieces; one for the process and one for the unused memory.

Knowlton [2] and Knuth [3] described the original buddy system. This memory management scheme allocates blocks whose sizes are powers of 2. Generally, this system is called as the *binary buddy system* to distinguish it from the other buddy systems. Hirschberg [4], taking Knuth's suggestion, has designed a *Fibonacci buddy system* with block sizes which are Fibonacci numbers. Shen and Peterson [5] have described an algorithm for a *weighted buddy system* which provides blocks whose sizes are 2^k and $3*2^k$. Details of these systems are explained at section 3. The other buddy systems: *Lazy buddy system* based on that coalescing process is postponed due to probability of the same size request, Double buddy system which allocates power of 2 (2, 4, 6.…) and power of 2 starting different size such as 3 (3, 6, 12…) by using two trees, F-2 buddy which is based on the recurrence relation $L_{i+1}=L_i + L_{i-2}$ has similar structure with Fibonacci. But the performance of these systems is worse than binary, Fibonacci and weighted.

In this paper, existing basic techniques are examined. After giving the general information about these techniques, performance comparisons are performed. Simulator written in C++ programming language is designed to compare software techniques. Comparisons are performed with respect to time and memory usage (fragmentation) parameters.

2. BASIC CONCEPT OF DYNAMIC MEMORY ALLOCATION

The goal of allocator design is usually to minimize wasted space without undue time cost. An allocator must keep track of which parts of memory are in use and which parts are free. A conventional allocator can not control the number or size of live blocks. A conventional allocator also can not compact memory, moving blocks to make them contiguous. It must respond immediately to a request for space deciding which block of memory to allocate. It can only deal with memory whether it is free and only choose where in free memory to allocate the requested block[6].

Allocators record the locations and sizes of free blocks of memory in some kind of data structure. These structures may be a linear list, a total or partial tree or a Bitmap.

2.1. FRAGMENTATION

Fragmentation is one of the most important problem that an allocator encounters. The fragmentation problem prevents memory to be used effectively. Fragmentation is classified as external or internal [6].

2.1.1. Internal Fragmentation

Unless the set of requested block sizes is a subset of the set of provided block sizes, it will be necessary to allocate more memory space than requested block size. The memory wasted due to this overallocation is internal fragmentation. Measure of internal fragmentation is the ratio of number of overallocated blocks to number of allocated memory.

$$Internal\_fragmentation = \frac{overallocated\_memory}{allocated\_memory} \quad (1)$$

2.1.2. External fragmentation

External fragmentation happens when available free blocks at memory are too numerous and small and can not be used to allocate next requests for larger blocks. Measure of external fragmentation is the proportion of total memory which is available when overflow occurs [7].

$$External\_fragmentaton = \frac{requested\_memory}{available\_memory} \quad (2)$$

2.1.3. Total fragmentation

Internal and external fragmentation are an expected result of different properties of the methods used in memory allocation. But, both decrease the effective size of available memory due to creation of memory portions that can not be used. We define total fragmentation to be total amount of memory which is unusable due to either internal or external fragmentation. Since our definition of internal fragmentation is the propotion of allocated memory which is unusable, while external fragmentation is a proportion of total memory, total fragmentation is not simple sum of internal and external fragmentation, but rather calculated as,

$$total = (1 - External) * Internal + External \quad (3)$$
$$total = Internal + External - (Internal * External)$$

2.2. SPLITTING AND COALESCING

The allocator may split large blocks into smaller blocks to satisfy the request. The remainders from this splitting can be used to satisfy future requests. The allocator also coalesces adjacent

free blocks to yield larger free blocks. After a block is freed, the allocator may check to see whether neighboring blocks are free and merge them into a single, larger block. Splitting and coalescing processes must be done to decrease fragmentation problem.

2.3. EXECUTING TIME

Another performance criteria in memory allocation is execution time (allocation and deallocatin duration). Splitting and coalescing performed for decreasing fragmentation increases the running time. In general, running time is inversely proportional to fragmentation.

3. MEMORY ALLOCATION TECHNIQUES

Very different techniques are developed to combat fragmentation. Although, some of them decrease external fragmentation, the others solve internal fragmentation. But, there is not yet any mechanism to solve this problem perfectly. The basic allocation mechanisms are separated into 3 categories [5]:

- Bitmap
- Linked List techniques: First Fit, Best Fit, Next Fit, Worst Fit
- Buddy Systems: Binary, Fibonacci, Weighted

3.1. BIT MAP TECHNIQUE

The fundamental of Bitmap technique is that status of each blocks in memory is represented with a bit in Bit map which is 0 if the block is free and 1 if it is allocated. Figure 1 shows part of the memory and the corresponding bit map. The smaller the allocation unit, the larger the bitmap.

In spite of simplicity, the main problem is that when it has been decided to bring a k unit process into memory, the memory manager must search the bit map to find k consecutive 0 bits in the bit map. Searching a bit map for a request is a slow operation, so in practice, bit maps are not often used.

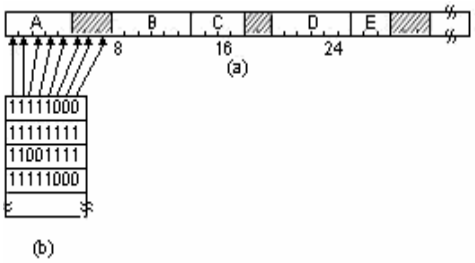

Figure 1: (a) A part of memory with five processes and 3 holes. The tick marks show the memory allocation units. The shaded regions (0 in the bit map) are free. (b) The corresponding bit map

3.2. LINKED LIST TECHNIQUES

Memory management techniques by using Linked lists is the most used technique. These techniques allocate by using linked list structure. Linked list techniques used more frequently are;

- First Fit: Allocate the first hole that is big enough.
- Next Fit: Allocate the first hole that is big enough starting where it left off.
- Best fit: Allocate the smallest hole that is big enough.
- Worst Fit: Allocate the largest hole

3.2.1. First Fit

The memory manager starts to scan from beginning of the lists until it finds a hole that is big enough. The hole is then broken up into two pieces, one for the process and one for the unused memory. A problem with first fit is that the larger blocks near the beginning of the list tend to be split first and the remaining fragments result in having a lot of small blocks near the beginning of the list. These small memory blocks can increase search times. Because many small free blocks accumulate and the search must go through them each time a larger block is requested.

3.2.2. Next Fit

The greatest difference of this technique from others is that search starts always different place of the list. The pointer records the position where the last search was satisfied and the next search begins from there. By means of this feature, Small and awkward memory hole can not accumulate at the beginning of the list as first fit technique.

3.2.3. Best Fit

A best fit allocator searches the list to find smallest free blocks large enough to satisfy a request and then allocate this blocks. Best fit is slower than first fit and next fit. Because it must search the entire list every time it is called.

It is expected from this strategy that unused holes are decreased. But, this is not performed and first fit algorithm may be more successful from this point of view. Because, after allocation are performed, the remainder may be quite small and perhaps unusable for next larger request. A little time later, memory has a lot of small blocks and total unused area is too much than that of first fit.

3.2.4. Worst Fit

Working manner of this technique is more similar to best fit. Difference between them is that while best fit select the smallest hole, worst fit take the largest available hole. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover from a best fit approach.

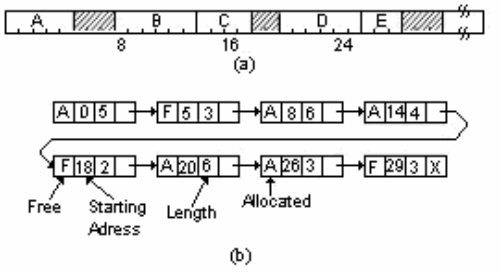

Figure 2: sample representation of linked lists (a) a part of memory (b) Corresponding linked list

Figure 2 shows a part of memory and a linked list represents status of block. For example, When 2 blocks are requested, First fit allocates free hole at 5, Best fit allocates free hole at 18 and worst fit allocates free hole at 8.

3.3. BUDDY SYSTEMS

Important property for the buddy systems over first fit or best fit memory management schemes was its reduction in search time to find and allocate an available block of appropriate size. By means of using tree structure, finding appropriate hole is performed faster.

Although Buddy system is faster, it has a very important disadvantage. Buddy systems are inefficient in memory usage. Though the linked list techniques have only external fragmentation, buddy systems have both internal and external fragmentation. Basis reason of this problem is that allocated blocks are power of two (for Binary buddy system). Therefore, first request size round up to smallest number that is power of two (for binary buddy) then allocation process is performed.

3.3.1. Binary Buddy System

Working mechanism of Binary buddy system is as follows: We start with the entire block of size 2U. When a request of size S is made: If 2U-1 < S<= 2U then allocate the entire block of size 2U. Else, split this block into two buddies, each of size 2U-1. If U-2< S<= 2U-1 then allocate one of the two buddies. Otherwise one of the two buddies is split in half again. This process is repeated until the smallest block greater or equal to S is generated [7].

Figure 3 shows the settlement of binary buddy system at binary tree. Striped circle, white circle and gray circle represent respectively split blocks for allocation, leave nodes that show free blocks, nodes that shows used memory chunks.

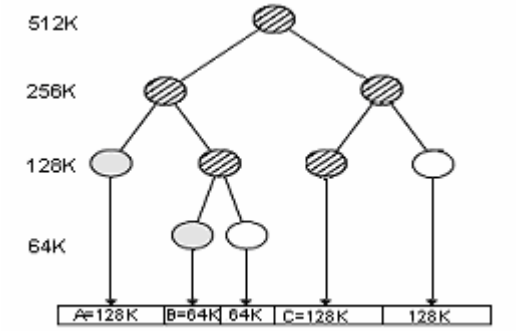

Figure 3: Tree structure of Binary Buddy System

3.3.2. Fibonacci Buddy System

In this system, blocks are split in respect of Fibonacci numbers. Working mechanism of it is similar to binary buddy system. Fibonacci series are defined as follows.

$$F_0 = 0, F_1 = 1, F_{n+2} = F_{n+1} + F_n (n \geq 0) \qquad (4)$$

According to this definition, the elements of Fibonacci series: *0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987...*

Figure 4 shows sample tree structure of Fibonacci Buddy systems.

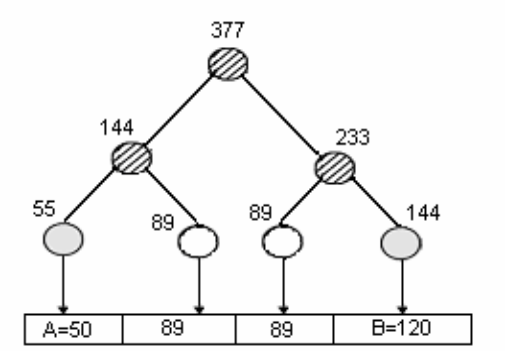

Figure 4: Tree structure of Fibonacci Buddy system

3.3.3. Weighted Buddy System

At the Weighted Buddy system, block sizes are as 2^k or $3* 2^k$. When blocks are split, applied rules are showed in figure 5 and 6.

As showed in figure 5a, If split block size is 2^{k+2}, this block are split to $3*2^{k+2}$ and 2^k block size. If block size is $3*2^{k+2}$(Figure 6b), it separated into 2^{k+1} and 2^k block sizes [$3*2^k$ -> (2^{k+1}, 2^k)].

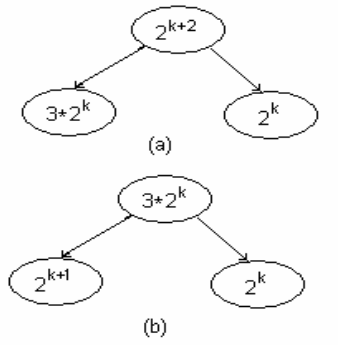

Figure 5: (a) Splitting of 2k+2 block size (b) Splitting of 3*2k+2 block size

4. SIMULATION AND PERFORMANCE ANALYSIS

Some of the examined software techniques are simulated for performance analysis by using C++ programming language. For this purpose, First Fit, Best fit, Worst Fit, Bit map, Binary Buddy and Weighted Buddy software techniques are simulated. Simulation results are obtained for selected performance criteria that are fragmentation, allocation and deallocation duration in respect of memory size and maximum allocatable object size. (Figure 8-10)

In practice, nodes which have a few fields for holding allocation information are defined using "struct NODE" structure. This structure consist of field which hold father, left child, right child in tree structure and next address in list. Another field indicates status of blocks. If the value is '1', blocks are allocated, otherwise they are free. Allocation and deallocation processes are executed by using tree or list which is formed based on this defined structure. Defined specially by Mymalloc and Myfree functions at the program are called randomly with 66%- 33% probabilities respectively. Generated and allocated values are saved in an array. When Myfree function is called, a value is selected randomly and then these blocks are deallocated. This array hold allocated address values. The aim of using this array is that it is not possible to deallocate the address which has not been allocated.

Allocation processes are performed by using generated random numbers in program. Internal and external fragmentation is calculated after each allocation process. In this practice, total fragmentation is calculated by using average internal and external values for 100 steps.

Allocation and deallocation process can not be calculated sensitively by using classic time function in C++. Because, running time implements in very short time. In order to solve this problem, Windows API "QueryPerformanceFrequency*"* is used. By this API, each allocation and deallocation duration and their average are calculated in the micro second type. Program is run 100 times for obtaining more realistic results.

Buddy systems suffer from both internal and external fragmentation. The others have only external fragmentation. Internal and external fragmentation values of binary and weighted buddy systems are showed graphically in figure 7. As shown in figure, it can be observed that Weighted buddy system suffers from internal fragmentation less than Binary buddy. In contrast, in respect of external fragmentation binary buddy is more successful. Another result obtained from this figure is that maximum allocatable block number is directly proportional to external fragmentation.

In this study, although results of all techniques are obtained in respect of performance criteria, Graphics are drawn only for the best method in each category. In this way, the performance of categorically classified methods is evaluated.

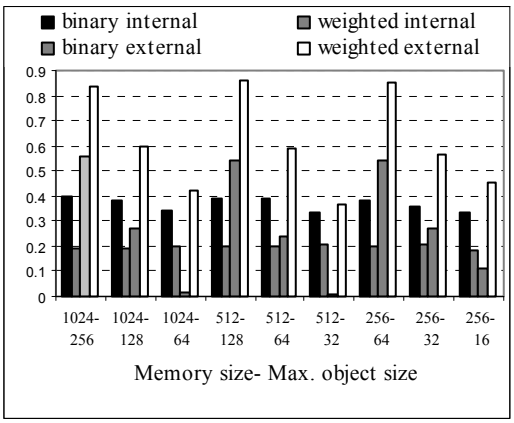

Figure 7: Internal and External fragmentation of Buddy Systems

In figure 8, In respect of fragmentation criteria, First fit is the best and Binary buddy is the worst technique. Allocation of bit map method gives close similarity to first fit. Because, searching bits is performed as first fit.

Figure 9 shows that, according to allocation process, the best performance technique is binary buddy, the worst is bitmap. Search process in tree structure can be done in shorter time than in list structure. So, Buddy systems have the best performance in respect of allocation process duration. In contrast, in bit map techniques, allocation is faster at small memory sizes due to searching at bits and not splitting. The bigger memory size, the slower allocation time in the bit map.

In figure 10, in respect of deallocation duration performance criteria, the most successful technique is observed as bit map. As coalescing process is used in bitmap technique, deallocation process is performed in short time by updating the bitmap by inverting (from 1 to 0) all bits corresponding deallocated blocks.

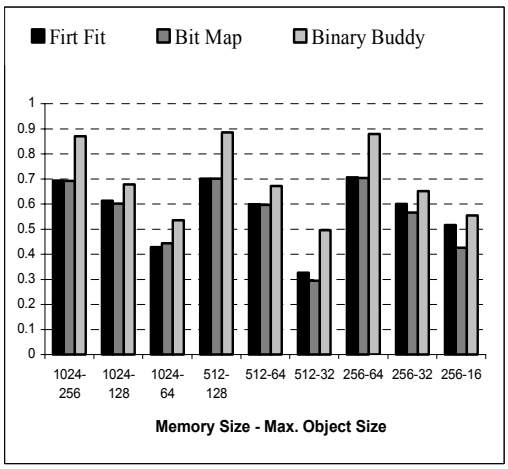

Figure 8: Total fragmentation values of techniques

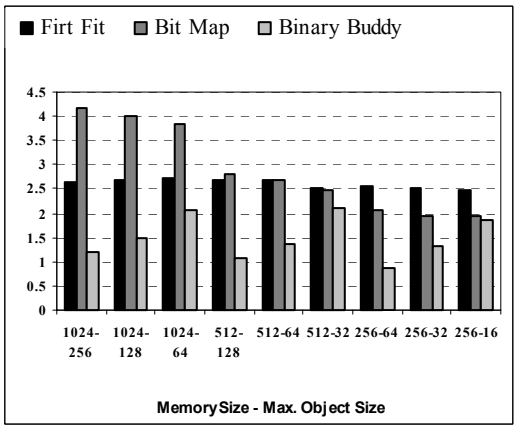

Figure 9: Allocation duration of techniques (micro second)

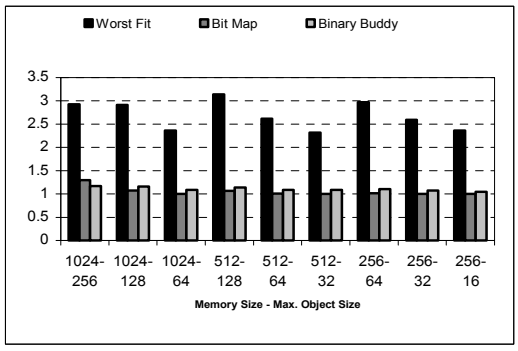

Figure 10: Deallocation duration of techniques (micro second)

5. CONCLUSION

In this paper, Performance analysis of examined memory allocation techniques is performed comparatively in respect of performance criteria.

The performance analysis of examined memory allocation techniques, a Simulator is written in C++ programming language. In this study, although results of all techniques are obtained, in respect of performance criteria, Graphics are drawn only for the best method in each category. In this way, the performance of methods classified categorically is evaluated. Despite First Fit listing technique shows better performance according to fragmentation, it can not show the same success in respect of allocation and deallocation time. Binary buddy is very fast but it has the worst at fragmentation rate. According to these results, it is seen that there are not yet the best performance techniques point from all performance criteria.

Consequently, it can not be said that the basic memory allocation techniques are successful.
To minimize the fragmentation problem and to allocate faster, new memory allocation techniques can be implemented by using hardware structure.

It is suggested that by using new and different data structures, faster and more efficient memory allocators can be achieved.

6. REFERENCES

[1] Zorn B., July 1993, The measured cost of conservative garbage collection, *Software-Practıce And Experıence*, Vol. 23, No. 7, 733-756

[2] Shen. K.K. and Peterson, J.L., Oct. 1974, A weighted buddy method for dynamic storage allocation. *Communications Of The ACM 17,* 10, 558-562

[3] Barkle, R.E. ve Lee T.P., Dec 1989, A Lazy Buddy System Bounded by Two Coalescing Delays per Class, *Proc. 12th Symp. Opertatıng System Prıncples*, vol:23, no:5, 167-176

[4] Hırschberg, D.S., Oct. 1973, A class of dynamic memory allocation algorithms. *Communications Of The ACM 16,* 10, 615-618

[5] Tanenbaum A.S. ve Woodhull A. S., 1997, *Operating Systems Design and Implementation,* Prentice Hall, Portland, 0-13-638677-6

[6]Wilson P. R., Johnstone M. S., Neely M. ve Boles D., September 1995, Dynamic Storage Allocation: A Survey and Critical Review, *International Workshop on Memory Management*, Kinross, Scotland, UK

[7] Peterson J.L. ve Norman T.A., June 1977, Buddy systems, *Communications Of The ACM*, Vol. 20, 421-431

Fethullah Karabiber *was born in Adıyaman, Turkey in 1977. He received B.Sc. degree from İstanbul Technical University, Department of Electronics and Communication Engineering in 2001. Since 2001 he has been working as a reserch assistant in the department of Computer engineering in İstanbul University. His research interests include VLSI, Computer architecture, digital design, VHDL, programming*

Ahmet Sertbaş *was born in İstanbul in 1965. He received the B.S. and M.Sc. degrees in electronic engineering from the Istanbul Technical University in 1990, the Ph.D. degree in electronic department from Istanbul University in 1997 respectively. He has worked as Research Assistant at I.T.U. during 1987-1990, research engineer at Grundıg firm during 1990-1992, an instructor at the Vocational School of Istanbul University during 1993-1999. He is currently an Assoc. Professor in the Department of Computer Engineering at the University of Istanbul. His research interests include computer arithmetic circuit design, computer architecture and computer-aided circuit design, circuit theory and applications.*

A. Halim Zaim *received the B.Sc. degree (Honor) in computer science and engineering from Yildiz Technical University, Istanbul, Turkey, in 1993, the M.Sc. degree in computer engineering from Bogazici University, Istanbul, Turkey, in 1996 and the Ph.D. degree in electrical and computer engineering from North Carolina State University, Raleigh, in 200. He is currently working as an Associate Prof. at Istanbul University. His research interests include computer performance evaluation, satellite and high-speed networks, network protocols, optical networks and computer network design.*