Parameters Affecting the Functionality of Memory Allocators

Ghassem Barootkoob
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
gbarootkoob@comp.iust.ac.ir

Mohsen Sharifi
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
msharifi@iust.ac.ir

Ehsan Musavi Khaneghah
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
emousavi@iust.ac.ir

Seyedeh Leili Mirtaheri
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
mirtahei@iust.ac.ir

Abstract- **Parameters affecting the functionality of an operating system's memory management unit depend on a number of factors such as allocation and deallocation strategies, localization, internal and external fragmentation, regional clustering, allocation and deallocation speeds, multi-threading, reusability, wasted memory reusability, implementation level and dynamicity. In this paper, we examine different memory allocation methods used in the BSD operating system as well as parameters affecting the functionality of its memory management unit extracted from these methods. Besides identifying the relationships and dependencies between these parameters, we report our experimental measurements of the effect of each parameter on the performance of different memory management methods used in BSD. Our evaluations not only provide a comparative view of different allocation methods in BSD that have been deployed over time to complement each other, they also put into perspective different memory allocation methods used in different operating systems with respect to parameters such as multithreading ability, number of requests served and memory fragmentation rate.**

Keywords: Operating System Kernel, Memory Management, Memory Allocation Methods, Performance.

1. INTRODUCTION

Memory management unit in an operating system kernel is responsible to provide a fair share of the limited address space of physical memory of a computer to many contesting execution units namely processes [1]. Due to high frequency of access of processes to this limited address space at runtime, the performance of memory management unit is most detrimental on the overall performance of processes running on this memory system. It is thus critical to identify parameters that most affect the performance of memory management units in operating systems.

Triggered by the lack of published public reports in academia on the parameters affecting the functionality of memory management units, we have selected BSD [2] as a well-established operating system to examine its different complementary memory management methods that it had deployed over time. Having determined the influential properties and parameters in BSD memory management (specially, allocation) methods, we have rated their relative criticality by studying memory management methods of Linux [3], Mac [4], Solaris [5], and Sun OS [6]. The results of this study showed that all these operating systems except Linux have inherited their memory management methods from BSD, making the results applicable to a wide range of operating systems. We lastly analyzed these parameters to find their interdependencies with respect to allocation speed, loss rate, and localization, as well as their influence on the performance of memory management unit.

We have organized the rest of paper as follows. Section 2 examines three BSD memory allocation methods namely PHKmalloc [7, 8], Slab [9,10], and Jemalloc [11], and compares their properties with those in Linux, Mac, Solaris, and Sun OS. Based on the results of our examinations, Section 3 presents the effective parameters on functionality of memory management units. Section 4 contains the conclusions and suggestions for future studies.

I. MEMORY ALLOCATION METHODS

In this section, we examine memory allocation in BSD operating system and try to express its properties and changes over time and also the reasons for its success among other operating systems will be determined.

A. PHKmalloc Allocator

Most applications nowadays require as much memory space as possible at runtime regardless of how much they have space they have already been granted and irrespective of similar requirements of other processes running other applications on the memory system. This describes why most operating systems deploy one or more dynamic memory allocation methods.

PHKmalloc is a distinguished memory allocator in the BSD operating system that has evolved over time and gets full support from both the hardware and the system software. [8, 12, 13]. Here we study its specification from different

perspectives including its operational specification, allocation speed, drawbacks, and performance.

Operational specification. We may summarize the specification in eight items: 1) it is a page-based allocator, 2) it aims to minimize access to pages for both applications and itself in order to increase overall performance, 3) every page is divided into equal size objects that are managed more efficiently in order to increase overall speed and performance, 4) more pages are allocated to larger objects, 5) for objects smaller than a half page, the object size is rounded to the nearest power 2 and requests for a page larger than the page size is rounded to the first power of several pages, 6) the allocator maintains a directory of all assigned pages by creating a bitmap containing this information at the beginning of each small object page, 7) for allocation of small objects a linear search of the bitmap finds the first available place on that page, and 8) the allocator takes all necessary supports for virtual memory from both hardware and system software.

Allocation speed. There are three main reasons for reasonable speed of PHKmalloc: 1) it only needs simple conversion and rounding thus it is so fast, 2) it has a method that calls for memory whenever the amount of free space memory is lower than a threshold so it does not decrease the overall performance, and 3) the rounding of several pages leads to higher performance through improvements in data localization operations [14].

Drawback. The most critical drawback of PHKmalloc is its high rate of memory waste [15] caused by rounding mentioned aove. We later explain in the last section of this paper how this weakness can be alleviated by a tradeoff between memory waste and other performance-wise parameters.

Performance. There are two performance issues on this allocator: 1) how much time is spent to search and manipulate data structures denoting its overload?, and 2) how well does this allocator manages the memory denoting its quality of allocation?

B. Slab Allocator

The Slab allocator or regional is another famous allocator in BSD designed for specific applications and not useful to all situations [9]. This allocator has tried to resolve the memory waste and external fragmentation problems. There are two forms of fragmentation: external and internal. External fragmentation is a measure, which affects the virtual memory in a physical form. Internal fragmentation is a measure to evaluate the amount of memory waste in single allocation.

Objects are allocated equally and the number of objects is a power of 2. Memory blocks are filled as much as possible. If a block is not completely filled, it can be filled by a simple management approach. This method suffers from internal fragmentation because objects are considered as a power of 2 and smaller size objects requiring less memory may well exist. Slab can perform fast in special cases where memory space consists of a list of equal size objects (e.g. a dynamic array of structures) [9, 10].

C. Jemalloc Allocator

BSD operating systems have tried to provide the best support for multi-processor systems. Jemalloc allocator has thus been developed to replace previous allocators of BSD to specifically better support multi-processor systems [10].

We may summarize the strengths of Jemalloc in four respects: 1) support for multi-processor programs, 2) increased speed and efficiency of memory management in multi-processor programs in multi-kernel environments, 3) compatibility with PHKmalloc to run single processor programs, and 4) achieving a fast allocation compared to other allocators of BSD. A disadvantage of Jemalloc is its relatively high rate of memory fragmentation although its memory waste is less than PHKmalloc and still comparatively reasonable.

II. MEMORY ALLOCATORS COMPARED

We study different comparisons in different papers and present one of them as a example and summarize the other in below tables In this workcompare and experimentally evaluate PHKmalloc and Jemalloc thread-based BSD allocators as well as the Linux dlmalloc allocator [16] based on different criteria. he has selected dlmalloc as a basis for his experimental comparison due to its wide usage in these days and age.

Evans [11] has compared PHKmalloc, Jemalloc and dlmalloc allocators. To do the comparisons experimentally, he used a benchmark malloc-test[17 with two rates and five threads on a system with four processors using the 6th version of BSD operating system. We ran the three chosen allocators on this platform. and however ignored many parameters of dlmalloc in hisexperiments for brevity since they did not affect the overall conclusions we could make from his experimental comparisons.

Figure 1 shows the increasing patterns of allocations per second of the three allocators with respect to the number of threads. malloc-test ran Three times where each run consisted of forty million allocations and lack of allocations, and averaged the results . The number of allocations per second decreased by increases in the number of threads in PHKmalloc and dlmalloc, while the number of allocations per second increased for the first 4 threads and then reached a steady state in Jemalloc.

Figure 2 shows the same allocators benchmarked with a different set of queries.

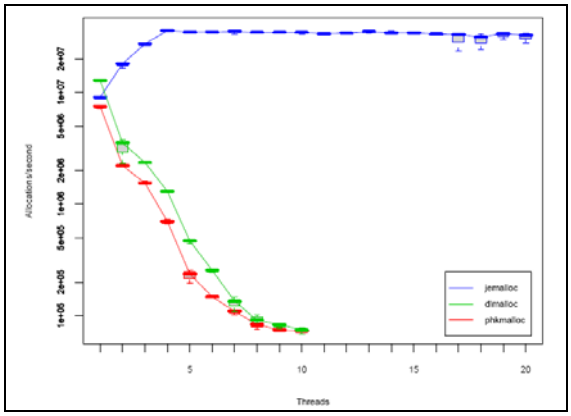

Figure 1. The number of allocations per second versus the number of threads [11]

Although all three methods acted similarly, they were different in details. Both dlmalloc and PHKmalloc had a high scattering rate enabling them to process forty or more million queries while Jemalloc had a lower scattering rate and standard deviation than dlmalloc and PHKmalloc. Lower scattering rate has lead to a more stable behavior of Jemalloc as Figure 2 shows. Jemalloc has a similar performance to dlmalloc but with a lower fluctuation rate over time.

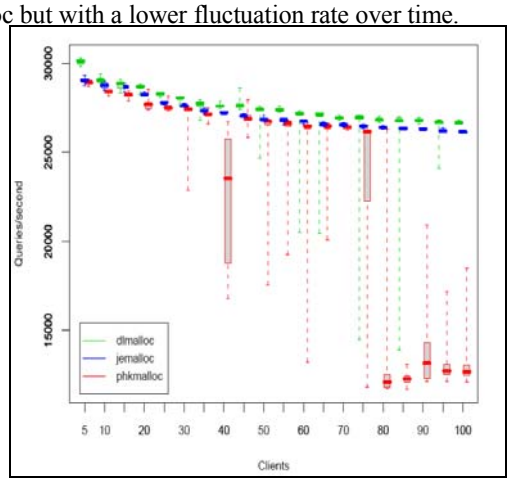

Figure 2. The number of queries per second versus the number of clients [11]

All in all Jemalloc performed better than PHKmalloc in all cases and also in some cases it performed better than dlmalloc. As stated before, Jemalloc operates best in multi-processor systems but it has the same performance as other allocators in single processor systems.

Figure 3 shows a comparison between allocators in Linux and BSD operating systems based on the Jemalloc method and demonstrates the superiority of BSD methods and shows that it [17] has the ability to perform higher number of operations per second than Linux.

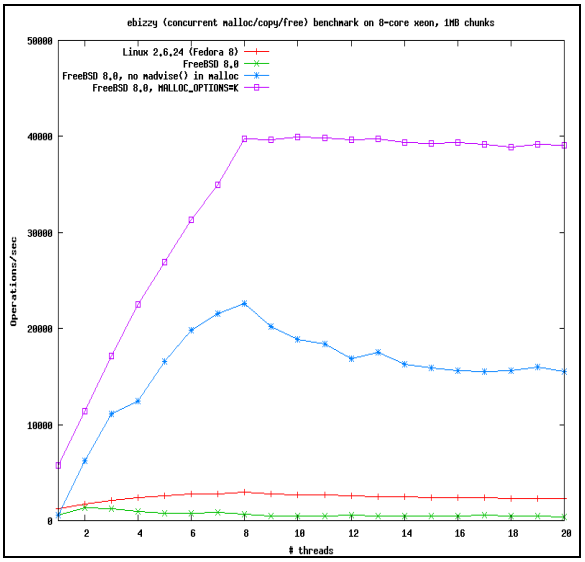

Figure 3. The number of queries per second versus the number of threads [18]

For another comparisons of this methods we refers to this papers [9,10,11,14,15] that we will summarize in below tables and present analysis of them.

A. Comparing different memory allocation methods in BSD based on the effective parameters in memory management

In our studies, we observed that BSD has used different allocators over time for different reasons. We tried to assess notable allocation methods BSD has used. Note that BSD has had the best allocators in its lifetime and it has required heavy changes to the operating system code in order to equip itself with a super fast allocator.

In the following we compare the three main allocators, PHKmalloc, Slab and Jemalloc, based on effective parameters in memory management. Tables 1, 2, 3 and 4 show the results.

TABLE I. EFFECTIVE PARAMETERS IN MEMORY MANAGEMENT IN DIFFERENT ALLOCATION METHODS

	Strategy Parameter	**Region Clustering Parameter**	**External Fragmentation Parameter**
PhkMalloc [8]	Minimum access to page[15,19,20]	No [14,15]	Low [14,15]
Slab [9,10]	Dynamic Management	Yes [9]	Low [10]
Jemalloc [11]	Similar to Phk and differnet in multi-thereaded [11]	Not Known	Low [11]

TABLE II. EFFECTIVE PARAMETERS IN MEMORY MANAGEMENT IN DIFFERENT ALLOCATION METHODS

	Internal Fragmentation Parameter	Locality Parameter	Speed Parameter
PhkMalloc[8]	High[14,15,19]	High Medium [14,15]	High[14,15]
Slab[9,10]	Low[10,20]	High Medium[15]	High[10,13]
Jemalloc[11]	Lower Phk[11]	High Medium [11,15]	High[11,18]

TABLE III. EFFECTIVE PARAMETERS IN MEMORY MANAGEMENT IN DIFFERENT ALLOCATION METHODS

	Partial Allocation Parameter	Implementation Level Parameter	Soft/Hard Parameter
PhkMalloc[8]	No[8]	Kernel[11]	Hybrid[12]
Slab[9,10]	Yes[10]	kernel[21]	Soft
Jemalloc[11]	No[11]	User[11]	Soft[11]

TABLE IV. EFFECTIVE PARAMETERS IN MEMORY MANAGEMENT IN DIFFERENT ALLOCATION METHODS

	Dynamicity Parameter	Alloc/ Dealloc Parameter	Frag Reuse Parameter	Multi Threads Parameter
PhkMalloc[8]	Yes [8,11,12]	Yes	Yes [14]	No [8,11]
Slab[9,10]	Yes[10]	Yes	Yes	No[11]
Jemalloc[11]	Yes [11,15]	Yes	Yes	No[11]

In the following, we explain the parameters we have used in the above presented tables.

Regional clustering. As its name implies, this parameter works in the memory management and allocation using the area clustering and tries to break areas into several parts and place them into clusters because when the areas are clustered the amount of memory waste is decreased and therefore localization is increased. So this parameter indirectly affects localization.

Support of multi-threading systems. This parameter states whether an allocator can be used in new modern systems or not? Multi-threading in a system means that the system has requirement for rapid execution of programs. This parameter is extremely effective in improving both allocation speed and memory retrieval. **Fragmentation reuse.** This is a parameter that affects allocators and shows if wasted memory can be used again or not? If wasted memory can be used again it shows that the amount of memory loss is lower in that allocator that other allocators. So this parameter indirectly affects the waste of memory.

Fragmentation. This parameter includes the internal and external fragmentation parameters and generally shows the amount of memory waste for an allocator.

Implementation level. Implementation level is another parameter that is considered in this field and is divided into two parts.

Hardware/Software Support. This parameter shows whether an allocator uses hardware or software supports in its implementation or not? Hardware-based allocators are. In turn, software-based allocators are more flexible. It is possible to use any of them or a combination of hardware and software supports in implementing an allocator.

Kernel/User. Another issue is the implementation level of an allocator. Kernel-level implementations are faster.

Memory deallocation. This parameter helps to retrieve allocated memory spaces according to some algorithms and affects the speed and the amount of memory waste based on the proposed algorithm. For example, if memories are retrieved using FIFO, the system may suffer from high memory waste specially when faced with sudden large memory releases and reallocate of released memory because the best memory space might not have been selected for retrieval.

Partial allocation. Another effective parameter in memory waste is the ability to retrieve allocated memory but this parameter is only used in methods with a structured allocation method, for example in our tables only Slab has this characteristic. This parameter indirectly affects wasted memory and localization parameters.

Strategy This parameter determines the strategy used in allocating and retrieving memory. Many parameters are affected by this parameter.

Dynamicity. Another parameter is the implementation dynamics. This helps the dynamic allocator of memory to change based on the selected strategy to make the best decisions to speed up and reduce memory waste.

Having analyzed the above parameters and evaluated their effects on the three main parameters, localization, memory waste and speed, it is necessary to investigate the effects of these main parameters on each other. Three different cases are considered.

Speed versus fragmentation. Different studies [20, 21] have shown that there is an inverse relation between these two parameters. Allocators with high speed act extremely bad in memory waste parameter. For example we can name dlmalloc with high speed but extremely bad memory waste. This refers to the used strategies because strategies express the rate of memory waste. The more a strategy bounds itself to speed, the more it is forced to accept memory waste because in order to find the fastest possible solution it has to use simpler search algorithms.

Localization versus speed. Studies have shown [15] that the localization has an inverse relation with speed parameter because in order to increase localization, allocator must search more to find smaller areas and this affects the search algorithms and therefore it is time-wasting.

Fragmentation versus localization. These two parameters are also against each other. From the localization parameter point of view, decreasing the memory waste rate causes less memory blocks to be available to be searched because the localization request of available memory is requested at runtime. Any effort to reduce the memory waste rate reduces localization too. Indeed when the availability of memory increases, the possibility to place related parts near each other increases too and this enhances localization.

III. CONCLUSIONS

We analyzed the memory management methods in BSD operating systems to identify the parameters affecting the performance of memory management, memory allocation and the relationship between these parameters. To assess different memory management methods in BSD, their characteristics, strengths and weaknesses, we extracted these parameters. Parameters include allocation and retrieving speed, waste rate, localization, strategies of allocation and retrieving, implementation level, support for multi-threading, support for retrieving allocated memory, area clustering, paging and dynamicity. We identified the dependency of these parameters and their effects on each other. We showed that speed, the amount of memory waste and localization were the most important parameters affecting the performance of memory management. A reasonable tradeoff between these three parameters in the design of memory managers can lead to a high performance general memory manager.

REFRENCES

[1] A. S. Tanenbaum. Modern Operating Systems, Second Edition, Prentice-Hall, 2001.

[2] M.K. McKusick and G.V. Neville-Neil, The Design and Implementation of the FreeBSD Operating System, Addison Wesley, 2004.

[3] M.Gorman,Understanding the Linux Virtual Memory Manager, Prentice- Hall, 2004.

[4] A.Singh, Mac OS X Internals: A Systems Approach, Addison Wesley, 2006.

[5] G.Mann,The Solaris Book of New Science Fiction*: Volume 2* , Addison Wesley, 2008

[6] B.D.Heslop and D.Angell,Mastering Sunos, Prentice-Hall, 1990.

[7] M. K. McKusick and M. J. Karels, "Design of a General Purpose Memory Allocator for the 4.3BSD UNIX† Kernel*,"* Proceedings of the San Francisco USENIX Conference, pp. 295-303, 1988.

[8] P. H. Kamp, "Malloc(3) revisited." In USENIX 1998 Annual Technical Conference: Invited Talks and FREENIX Track pp 193–198, 1998.

[9] J Bonwick. "The Slab Allocator: An Object−Caching Kernel Memory Allocator." Usenix Conference, pp. 87−98, 1994.

[10] J Bonwick and J Adams ,Extending the Slab Allocator to Many CPUs and Arbitrary Resources. USENIX Annual Technical Conference, 2001.

[11] J Evans, A Scalable Concurrent malloc(3) Implementation for FreeBSD, BSDConference, 2006.

[12] W.Li, S.P.Mohanty and K.Kavi, "A Page-based Hybrid (Software-Hardware) Dynamic Memory Allocator," IEEE Computer Architecture Letters 2006.

[13] J.M. Chang and E.F.Gehringer, A High-Performance Memory Allocator for Object-oriented Systems, IEEE Transactions on Computers, pp 357-366, 1996.

[14] Y.Feng and E.Berger , A Locality-Improving Dynamic Memory Allocator. MSP, 2005.

[15] A Jula and L Rauchwerger, "Balancing Allocation Speed, Locality and Fragmentation in a Locality Improving Allocator," Technical Report TR08-002 Parasol Lab Department of Computer Science Texas A&M University College Station, TX 77843-3112, 2008.

[16] Doug Lea, A Memory Allocator, http://g.oswego.edu/dl/html/malloc.html, see at 2011

[17] C.Lever , and D Boreham, "malloc() Performance in a Multithreaded Linux Environment." In USENIX 2000 Annual Technical Conference: 2000

[18] K. Kennaway,Memory allocation performance on FreeBSD and Linux with ebizzy , Technical Report FreeBSD Org, 2008.

[19] A. Jula and L.Rauc, "Two Memory Allocators that Use Hints to Improve Locality," International Symposium on Memory Managemen, Dublin, Ireland 2009

[20] P. R Wilson, M.S. Johnstone, M. Neely, and D.Boles. "Dynamic Storage Allocation: A Survey and Critical Review." '95: Proceedings of the International Workshop on Memory Management, pages 1–116, London, UK, 1995. Springer-Verlag.

[21] M.S. Johnstone and P.R. Wilson. "The Memory Fragmentation Problem: Solved?"'98 Symposium on Memory Management, pages 26–36, NewYork, NY, USA, 1998.