VARIATIONS ON THE BINARY BUDDY SYSTEM FOR DYNAMIC MEMORY MANAGEMENT

Arie Kaufman

Florida International University
Miami, Florida 33199

ABSTRACT

Two new variations of the binary buddy system for dynamic storage bookkeeping are introduced. These variations, the revised buddy system and the tailored-lists buddy system, recombine smaller memory blocks only upon necessity in an attempt to save on processor execution time. Simulation results comparing the three systems reveal that the two new variations are superior to the original buddy system, since they are faster and the difference in memory utilization is insignificant. A comparison of the revised system and the tailored-lists system indicates that they are equal in performance.

KEY WORDS AND PHRASES: dynamic storage allocation, binary buddy system, revised buddy system, tailored-lists buddy system, fragmentation

CR CATEGORIES: 3.89, 4.32, 4.39

1. INTRODUCTION

A fast algorithm for dynamic storage bookkeeping and management, known as the binary buddy system, was introduced by Knowlton [4], Knuth [5], and others (e.g., [1]). This scheme provides blocks whose sizes are powers of 2. A block of size 2^k is called a k-block. The scheme maintains separate lists of available blocks for each size 2^k $(1<k<M)$. The k-th list of available k-blocks is called the k-list. Initially, the entire pool of memory, one M-block, is free and available on the M-list.

When a request for a piece of memory is received, the requested size is rounded-up to the next larger power of two, 2^k, and a k-block is allocated from the k-list. If a k-block is not immediately available for allocation, i.e., the k-list is empty, a block of the next larger size, a (k+1)-block, is obtained and split into two equal parts, called buddies. Each of these buddies is a k-block. One of these buddies is used to satisfy the original request and the other k-block buddy is placed on the k-list. If a (k+1)-block is also unavailable the same allocation process is repeated for a (k+2)-block, and this search may recursively continue up to the largest size, 2^M. This algorithm is termed the allocation (reservation) algorithm.

When a k-block is released it will be recombined with its unique k-buddy to reform the original (k+1)-block, provided that the buddy is presently free. Recombination, like splitting, is recursive and the recombination process is carried out up to the largest size, 2^M, or until a buddy is unavailable for recombination. The resulting block is placed on the list of available blocks of its size. This algorithm is termed the de-allocation (recombination) algorithm.

Knuth [5] showed, based on simulation conducted, that the binary buddy system is preferred over other bookkeeping techniques of arbitrary-sized blocks. Peterson and Norman [6] compared several buddy systems: the binary buddy system, the Fibonacci buddy system [2] which provides blocks whose sizes are Fibonacci numbers, and the weighted buddy system [7] with block sizes 2^k and $3 \cdot 2^k$. They concluded that the memory utilization in a buddy system is relatively independent of the technique used, yet, based on the lowest execution time of the binary buddy algorithm they recommended it for general use. This algorithm is especially appropriate for real-time computer systems which are critically based on their

extreme awareness and fast response to the functioning real-time environment and its input.

Two important performance criteria of memory management algorithms are execution speed and memory utilization. The binary buddy system, as mentioned, is one of the fastest memory management schemes yet designed. The three dominant factors determining the running time of this system are:

a. The number of searches involved in finding a non-empty r-list ($k<r<M$) for a request of a k-block. There is no searching through the lists themselves since the first block of a list may be utilized.
b. The number of splits during the allocation process.
c. The number of recominations during the de-allocation process.

The second performance criterion is the effectiveness in memory utilization. Two different types of losses in memory utilization are distinguished: internal fragmentation and external fragmentation. Internal fragmentation is the loss of memory by rounding-up a request to the next larger power of two. External fragmentation is the result of breaking down the available memory into small blocks which cannot be recombined unless they are buddies. Consequently, overflows may occur, i.e., requests for large blocks may have to be rejected or stacked, even though the total extent of space available in smaller blocks is sufficient to fulfill the request.

Two improved versions of the original binary buddy system: the revised buddy system, and the tailored-lists buddy system, are introduced in Sections 2 and 3, respectively. The differences between the three systems are related to the criteria used for determining when to recombine buddies, resulting in variations in both the allocation and de-allocation strategies. The two new modified versions were inspired by Knowlton's statement [4], that in order to improve "the speed of the system it may not be desired to recombine free blocks every time recombination is possible." In Section 4 the three systems are explored and compared by means of comparative simulation experiments. The results, discussed in Section 5, reveal that the two modified versions perform faster than the original binary buddy system.

2. THE REVISED SYSTEM

In this revised version of the original binary buddy system an attempt is made to decrease the overhead of the memory manager by recombining buddies only when necessary, i.e., when an allocation request is received and no k-block is directly available. The de-allocation process of the revised system merely returns the freed k-block to the k-list, not attempting to recombine buddies as in the original buddy system. A pair of buddies may, therefore, co-exist on the same k-list.

When a request for a k-block is received, the allocation process allocates a k-block from the k-list. If the k-list is empty the allocation process endeavors to form a k-block by coalescing smaller available blocks. It first locates the largest j smaller than k, provided that the j-list contains buddies. Then the recursive recombination procedure takes place, starting from the j-list up to the k-list. If this attempt to form a k-block is unsuccessful, a recursive attempt is made to locate even smaller available buddies which would be recombined up to a k-block. If the whole coalescing process fails to form a k-block, a larger available block is recursively split as in the original buddy algorithm.

3. THE TAILORED-LISTS SYSTEM

In the tailored-lists binary buddy system an increase in the speed of the buddy system is attempted. The tailored-lists system strives to keep the number of blocks on each k-list in accordance with the proportion of requests that are expected of size 2^k. When a request for a k-block is received the probability that the k-list is not empty is higher, and hence no recombinations or splits are necessary. This technique assumes that the request size distribution is known, and the desired number of blocks in each k-list is determined (tailored) according to this distribution. For example, if the request sizes are uniformly distributed the aim is to keep the number of blocks on each k-list proportional to the size of the interval $[2^{k-1}+1, 2^k]$, i.e., the interval of request sizes that are allocated k-blocks. Table 1 shows the tailored lists for a uniform distribution of request sizes between 1 and 300, and an available storage pool of size $4096 = 2^{12}$. Notice that the entire storage pool has been spread over several lists: 5-list, 6-list, 7-list, 8-list, and 9-list.

The tailored-lists system requires two additional fields in each "head" list: the desired number of blocks on the list, and a dynamic counter of the actual number of blocks on the list.

An initialization procedure breaks down the entire available storage pool into smaller blocks and sets up the initial k-lists according to the desired number of blocks on each k-list. At this stage the desired numbers and the actual numbers of blocks on each k-list are the same. The de-allocation process of the tailored-

Table 1: Tailored Lists for UNIFORM (1,300)

k-list Available Space List	2^k Block Size	$[2^{k-1}+1, 2^k]$ Interval of Requests	2^k-2^{k-1} Size of Interval	Desired Number of Blocks
2-list	4	1-4	4	0
3-list	8	5-8	4	0
4-list	16	9-16	8	0
5-list	32	17-32	16	2
6-list	64	33-64	32	3
7-list	128	65-128	64	4
8-list	256	129-256	128	7
9-list	512	257-300	44	3
10-list	1024	-	-	0
11-list	2048	-	-	0
12-list	4096	-	-	0

lists buddy system endeavors to keep these two sets of numbers as close as possible. When a k-block is freed, it is returned to its list, the k-list. If the actual number of blocks on the k-list is greater than the desired number, and the k-block's buddy is also available on the k-list, the two buddies are recombined and placed on the (k+1)-list. Otherwise, the k-block is left on the k-list, as in the revised system described in Section 2. This process continues recursively up to the largest list, the M-list, or until no further recombinations are possible (because the buddy is occupied) or recommended (because the actual number of blocks is less or equal to the desired number of blocks).

The allocation process of the tailored-lists system is identical to that of the revised system described in Section 2, namely, the process either allocates a block directly from the k-list, or attempts to coalesce smaller blocks, or splits larger blocks.

4. SIMULATION EXPERIMENTS

Several simulation experiments were conducted comparing the three binary buddy systems. The simulator and the recursive algorithms were implemented in PL/1 on a UNIVAC 1100/80 at Florida International University.

The simulations were performed using a total available memory size of 4096. The largest block which may be allocated, however, was 512 (1/8 of the total memory), and the smallest block was 4. Three theoretical request size distributions were used in the simulation experiments: negative-exponential, normal and uniform. The mean request size ranged from 25 to 250.

The input request interarrival-time was negative-exponential with a mean of 10 time units. The life-time, i.e., the interval of time a block is occupied, was also negative-exponential with a mean ranging from 20 to 100 incremented by 20. It should be noted, however, that the significant characteristic of the simulation is the loading factor of the system, which is the ratio of life-time and interarrival-time. The loading factor ranged, therefore, from 2 to 10 incremented by 2.

The simulations ran for 5000 time units, and in order to accomodate system start-up the system reset all the parameters and accumulators at time 500. Statistics were, therefore, compiled for only 4500 time units.

The output of the simulator includes snapshots of the system every 500 time units, and a final report after 5000 time units or upon an overflow. The final report contains the simulation time, reset time, total number of requests, total number and percentage of requests rejected. In order to estimate execution speed the following statistics are reported: the total number of searches to find a non-empty k-list, the total number of splits, and the total number of recombinations, and the averages thereof per request.

The final report also contains information on memory utilization. The measure for internal fragmentation was defined as the ratio of overallocation (i.e., allocated size minus request size) and allocated size (see [6]). The measure for external fragmentation, reported by the simulator, is the percentage of the total memory which is available when overflow occurs (see e.g., [6]). Unlike the measure for internal fragmentation, which represents a continuous behavior of the buddy system, the measure for external fragmentation is a momentary measure. A continuous measure for external fragmentation, however, can be defined as the proportion of total memory in available blocks where the buddies thereof are occupied.

5. THE RESULTS

The principal results concerning execution speed of the three buddy systems are demonstrated in Figures 1, 2 and 3. Figure 1 plots the average number of searches per request as a function of the loading factor (life-time/interarrival-time), comparing the three versions of the binary buddy system. Figure 2 plots a similar comparison for the average number of splits per request versus the loading factor, while Figure 3 plots the average number of recombinations per request versus the loading factor. All graphs show results with a negative-exponential request size distribution with a mean of 100. These results fully confirm the predictions that the revised system and the tailored-lists system are faster than the original buddy system which wastes time in unnecessary recombinations, splits, and searches.

Since the tailored-lists system attempts to keep the desired number of blocks on each k-list, an even faster overall performance would be expected. The search, split and recombination results of this system, however, are not greatly improved upon those obtained by the revised system, which recombines buddies only if larger ones are not immediately available. This is probably due to the fact that the revised system tunes itself

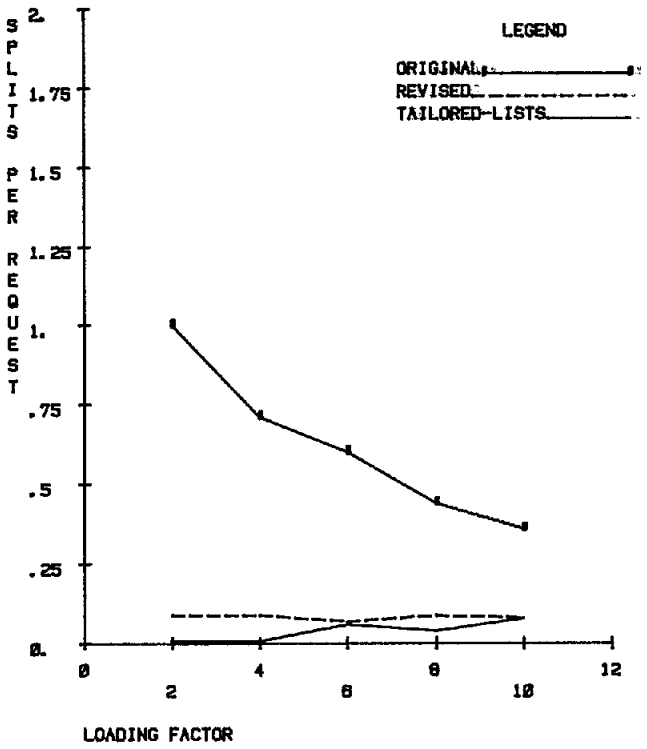

Figure 2: Number of splits per request as a function of the loading factor

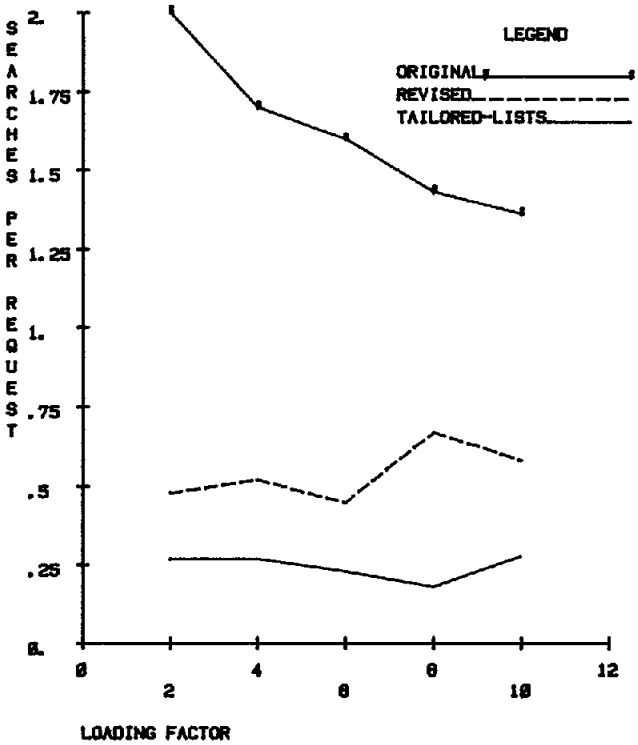

Figure 1: Number of searches per request as a function of the loading factor

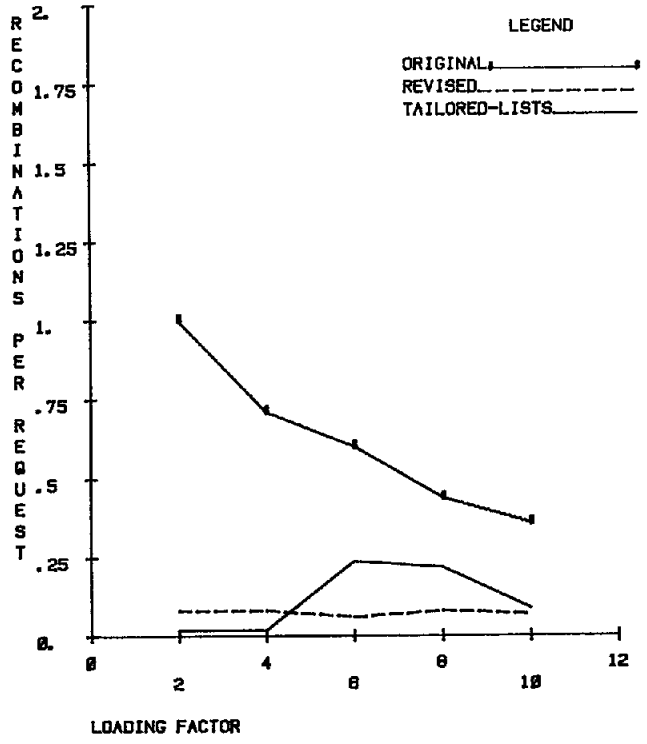

Figure 3: Number of recombinations per request as a function of the loading factor

up to the request distribution and at equilibrium, after a start-up period, the actual number of available blocks on each k-list is fairly close to the desired number of blocks induced by the distribution. Note that the recombination values for the tailored-lists system are on the average greater than those for the revised system. This was expected since the tailored-lists system recombines buddies more often in order to maintain the desired number of blocks on each k-list. The number of searches and splits for this system, however, are slightly smaller than those for the revised system. Presumably this is due to the fact that the k-list in the tailored-lists system are tailored to the distribution and therefore fewer searches and splits are required.

The results for internal fragmentation, given in Table 2, show that for all the three versions the average internal fragmentation for a negative-exponential distribution is 29.1%. This is in accordance with Knuth's computations [5]. The measure for internal fragmentation suggested in Section 4 is independent of the system used, since all the three systems, running with the same request size distribution, have identical sequence of request sizes. A difference in internal fragmentation occurs only between different request size distributions, (e.g., normal distribution produces an average of 29.4% and uniform distribution produces 30.2%).

Table 2 also summarizes the results for external fragmentation. As was expected the original system has the smallest amount of external fragmentation, since buddies are immediately recombined whenever possible, unlike the revised system which allows sets of k-buddies to co-exist on the k-list. The allocation algorithm of the revised system may choose the "wrong" block, i.e., a k-block whose k-buddy is also available, and may allocate or split up sets of buddies, disabling the system from satisfying larger requests, hence increasing external fragmentation. The external fragmentation of the tailored-lists system lies between that of the original and the revised systems, since on the one hand it allows some of the buddies to be recombined, and on the other hand it allows sets of k-buddies to co-exist on the k-list.

Comparative conclusions on overall memory utilization can be derived using the total-fragmentation statistic. This measure is the amount of memory made unusable by the system as a whole,i.e., due to the loss of memory by both internal and external fragmentations. The normalized weighted combination of the two sources of fragmentation, as discussed by Peterson and Norman [6], is summarized in the fourth column of Table 2. This column demonstrates that the three memory utilization values merely vary by about 2%, which may indicate that there is an insignificant difference between the memory utilization of the three versions of the binary buddy system.

6. CONCLUDING REMARKS

The two new variations of the binary buddy system, presented in this paper, may serve as alternatives to the original binary buddy system for dynamic memory bookkeeping and management. These improved versions have an advantage over the original system in execution speed, since they do not waste time in unnecessarily recombining available buddies, whereas in the original system recombination occurs regardless of the need for larger blocks. Analysis of the execution time graphs leads to the conclusion that the two new systems are nearly five times as fast as the original system. Regarding space efficiency there appears to be a relatively insignificant difference between the three systems. Overall, the two new versions, the revised system and the tailored-lists system, seem to be preferable to the original binary buddy system.

In light of this conclusion the two faster variations are especially appropriate for real-time systems, where processor overhead plays a major role. The two variations were actually implemented and installed, one at a time, in the real-time BGRAF2 system - a supporting system for a real-time interactive graphics language

Table 2: Memory Utilization

System	Internal Fragmentation	External Fragmentation	Total Fragmentation
Original	29.1%	12.6%	38.0%
Revised	29.1%	16.0%	40.4%
Tailored-lists	29.1%	13.6%	38.7%

[3]. The new variations replaced the original buddy system, partially improving upon the capability of the system to cope with real-time constraints.

In comparing the revised system and the tailored-lists system it appears that they operate equally well in both memory utilization and execution speed, with a slight, but apparently insignificant, advantage to the tailored-lists system. It appears, however, that in general the revised system may be superior to the tailored-lists system, since the latter requires the actual request size distribution to be known and relatively static. If the actual distribution changes from the expected distribution, unnecessary block recombinations are forced, thereby reducing the efficiency of the system. In contrast, the revised system dynamically determines its own k-list block counts based on the size distribution of blocks being released, which is, at equilibrium, identical to the actual request size distribution.

It should be noted, however, that the conclusions derived herein are based on the results of a pilot study with a limited number of experiments. Further investigations and comparisons of the two new variations are underway, utilizing both theoretical and actual request size distributions, with an increased number of experiments. Future research might focus on implementing the two new revised algorithms with other reported variations of the buddy system, such as the Fibonacci buddy system and the weighted buddy system.

ACKNOWLEDGMENT

The author gratefully acknowledges the assistance of Steve Good in the programming phase of the research.

REFERENCES

1. Augenstein, M. J., and Tenenbaum, A. M., Data Structures and PL/1 Programming. Prentice-Hall, 1979.

2. Hirschberg, D. S., "A class of dynamic memory allocation algorithms." Comm. ACM 16, 10 (Oct. 1973), 615-618.

3. Kaufman, A., "System design and implementation of BGRAF2." Computer Graphics 12, 3 (Aug. 1978), 87-92.

4. Knowlton, K. C., "A fast storage allocator." Comm. ACM 8, 10 (Oct. 1965), 623-625.

5. Knuth, D. E., The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley, 1968.

6. Peterson, J. L., and Norman, T. A., "Buddy systems." Comm. ACM 20, 6 (June 1977), 421-431.

7. Shen, K. K., and Peterson, J. L., "A weighted buddy method for dynamic storage allocation." Comm. ACM 17, 10 (Oct. 1974), 558-562. Corrigendum, Comm. ACM 18, 4 (April 1975), 202.