Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

STATISTICAL INVESTIGATION OF THREE STORAGE ALLOCATION ALGORITHMS

P. W. Purdom, S. M. Stigler
& Tat-Ong Cheam

Technical Report #98

August 1970

STATISTICAL INVESTIGATION OF THREE STORAGE ALLOCATION ALGORITHMS

1. Introduction

In many applications it is necessary to allocate memory in consecutive blocks of varying size. To this end, a number of dynamic storage allocation algorithms have been devised, many of which are described by Knuth [1]. The purpose of this paper is to compare the behavior of three of the better of these algorithms, based on measurements of both their efficiency (in utilization of memory space) and speed.

The three algorithms considered are the buddy system, the first fit method, and the segregated storage method. The buddy system, first published by K. Knowlton [2], allocates available space only in blocks which are a power of two times some basic size. It is discussed in Knuth [1], and has been partly analyzed by Purdom and Stigler [3] who found that the buddy system preforms well from the point of view of speed, but that it can be somewhat inefficient with respect to space utilization when subjected to random requests (see R. L. Graham and L. M. Robson [4] for a discussion of how well the buddy system performs when users are trying to cause trouble for the storage allocater). The first fit method resulted from combining the techniques of several researchers. Knuth [1] gives a discussion of the method, its historical development, and some analyses of its performance. Very little, however, of a theoretical nature

is known about its performance. The first fit method keeps a list of all available space from which it fills requests with the first encountered block of sufficient size. As we see later, this algorithm uses space efficiently, but for some distributions of requests it can do poorly with respect to speed. The third storage allocation algorithm, which we call "segregated storage", has not been previously published, and will be described in the next section. It combines some of the basic ideas of the buddy system and the first fit method. As shall be shown in section 4, segregated storage is generally more efficient in space usage (under the conditions tested) than either of the other algorithms, and is much faster than the first fit method under conditions of heavy load.

2. The Segregated Storage Allocation Algorithm

By combining some of the basic ideas of the buddy system and the first fit algorithm one can obtain a fast storage allocation algorithm which does not have the buddy system's disadvantage of being somewhat wasteful of space when requests are for sizes that are not powers of two. In the combined system k lists of available space are kept. The i^{th} list will have all available blocks whose size is greater than a_{i-1} and less than or equal to a_i, where a_o is zero and a_k is the total amount of space to be allocated. The remaining a_i's can be chosen in any way so long as $a_i > a_{i-1}$ for $0 < i \leq k$. One possibility is to let $a_1 = 1$ and $a_i = 2a_{i-1}$. This choice of a's was made for the version tested in section 4. When a block of size b is requested, the first block from list $i + 1$ is taken where i is chosen so that $a_i \geq b > a_{i-1}$. If list $i + 1$ is empty, then list $i + 2$

is used; if list $i + 2$ is also empty, then list $i + 3$ is used; etc. If all the lists from i to k are empty, then one searches list i for a block which is large enough. If there is still no space, of course, the request cannot be met (unless one is willing to repack memory). Once the block to be used is found, b cells are used to meet the original request and the remaining cells are put on the appropriate available space list (when only a small number of cells remain, it may be better to give the requester all of the cells in the block). When a block of storage is freed, it is combined with its neighbors (as is done in the first fit method) and put on the appropriate available space list.

The values of the a_i will have a major effect on how fast the algorithm runs. For any particular application of the algorithm one should investigate what values work best, the choice being based on the request distribution size.

3. Method of Investigation

Simulation programs were constructed to gather statistics on the operation of each of the three methods. Whenever the simulation requested memory space, the program counted the number of list elements that were searched before either finding a block for the item or deciding that there was no room available. When there was no room available to accommodate the request, it was rejected and no further action was taken on that request (except to count the number of requests rejected.)

In the simulation, requests arrived as a Poisson process with rate λ, the size of blocks requested was selected (independently) according to a geometric

distribution with mean size c , and the length of time the block of memory was required was distributed according to an exponential distribution with parameter one (independently of arrival time and size.) These distributions are similar in shape to many such distributions measured on actual systems. (See Bryan [5] and Batson et al [6].) These distributions were chosen to give a general indication of the relative merits of the three algorithms; investigators interested in more precise results for particular systems would, of course, need to perform further studies.

Tables 1, 2, 3, give the mean space utilization (ratio of total number of requests to requests accepted) and mean speed (mean number of searches required to accept or reject requests) for the three algorithms, based on runs of 1600 accepted requests. Multiple and longer runs were performed at various values of the parameters in order to investigate the accuracy of the simulations. The results of these runs are given in Table 4.

In many applications one is interested in how these storage allocation schemes behave when those requests which cannot be served immediately are queued and then served as soon as space is available. The figures presented here can serve as a rough guide to how a queued system would perform in those cases where the load on the system is light (i.e. when the corresponding non-queued system does not reject many requests), but they are not very useful for predicting the behavior of a heavily loaded queued system. Comparisons between queued and non-queued systems are made particularly difficult by the fact that the average size block rejected by the non-queued systems is much larger

(up to about twice as large) as the size of the blocks accepted.

4. Discussion of Results

Inspection of Tables 1, 2, 3 reveal that as far as utilization of memory space is concerned, the segregated storage and first fit methods exhibit almost identical performance, with segregated storage storage perhaps slightly more efficient. The buddy system, however, rejects up to about 15% more requests under conditions of heavy loading.

The comparisons of the running time of the algorithms, as measured by the mean number of searches per request, reveal that the buddy system ran faster than the first fit method except for some runs with light loading and some runs with large mean block size (where both methods were fast.) The buddy system was much faster than the first fit method for the important cases of heavy loading caused by many small blocks.

The running time of the segregated storage method will of course depend critically on choice of list sizes (the a_i's of section 2.) For the case considered here, where $a_i = 2^{i-1}$, the buddy system was always faster than the segregated storage method, although the speeds were about the same except for runs with light loading. The segregated system can be made to run much faster under light loading by choosing a smaller set of a_i's, (i.e. where the individual a_i's are further apart) because for light loading most of the time is taken by searching over empty lists.

For applications where the loading rate changes slowly, it probably would be worthwhile to modify the segregated storage method so that the algorithm would change the a_i's as the load changes. If the a_i's are kept in a table, they may be changed quickly. Just associate the i^{th} old list of storage with that new a_j which satisfies $(a_j)_{new} > (a_i)_{old} \geq (a_{j-1})_{new}$.

5. Tables

Tables 1, 2, and 3 give the ratio of the total number of requests received to the total number of requests filled, and the mean number of searches per request (in parentheses), for the three algorithms based on runs of 1600 filled requests. Table 1 gives the statistics for memory of total size 256 cells, mean rate of requests (λ) ranging from 2 to 96, and mean size of blocks requested (c) ranging from 2 to 64. Table 2 gives the statistics for memory of total size 128 cells, λ ranging from 3 to 50, and c ranging from 2 to 45. Table 3 gives the statistics for memory of total size 512 cells, λ ranging from 6 to 104, and c ranging from 3 to 85. In all cases, the first of the three pairs of numbers gives the results for the segregated storage method, the second pair refers to the first fit method, and the third pair refers to the buddy system.

Table 4 gives the results of multiple and longer runs for λ and c equal to (1,256), (16,16), (24,16), and (64,4). In each case the table gives means of the data collected over three runs together with one standard deviation bounds at various stages in the run, from 400 requests filled to 4800 requests filled. In

each case the results are given for both the ratio of requests received to requests filled and the mean number of searches (in parentheses.) As in tables 1, 2, and 3, the three sets of numbers refer to the segregated storage method, the first fit method, and the buddy system, respectively.

References

1. Knuth, Donald E., The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, Mass. (1968), pp. 435-451.

2. Knowlton, Kenneth C., "A Fast Storage Allocator" Communications of the ACM 8 (1965), pp. 623-625.

3. Purdom, Paul W., Jr. and Stigler, Stephen M., "Statistical Properties of the Buddy System," Journal of the A.C.M., 17 (1970), pp. 683-697.

4. Graham, R. L. and Robson L. M., to be published.

5. Bryan, G. E., "Joss: 20,000 Hours at a Console - a Statistical Summary," AFIPS Conference Proceedings, Vol. 31, 1967, Fall Joint Computer Conference.

6. Batson, Alan, Lu, Shy-Mingand, Wood, David C., "Measurements of Segment Size" Communications of the ACM 13 (1970), pp. 155-159.

Mean Size of Blocks Requested (c)

Mean Rate of Requests (λ)	2	4	6	8	10	12	16	20	24	32	40	48	56	64
2									1.01 (1.08) 1.00 (1.18) 1.03 (2.19)	1.05 (2.81) 1.03 (1.29) 1.07 (2.13)	1.07 (2.59) 1.06 (1.36) 1.15 (2.23)	1.12 (2.41) 1.10 (1.37) 1.18 (2.24)	1.1c 2.24 1.17 (1.46 1.28 (2.17)	1.20 (2.20) 1.23 (1.47) 1.33 (2.25)
4							1.00 (2.69) 1.00 (1.29) 1.03 (1.87)	1.01 (2.59) 1.02 (1.41) 1.05 (1.81)	1.04 (2.38) 1.03 (1.43) 1.08 (1.89)	1.10 (2.29) 1.10 (1.75) 1.21 (1.94)	1.15 (2.27) 1.15 (1.71) 1.27 (1.91)	1.22 (2.18) 1.24 (1.88) 1.42 (2.02)	1.3[illegible] (2.15 1.31 (1.90 1.49 (1.96)	1.37 (2.03) 1.41 (1.95) 1.59 (1.97)
6							1.02 (2.29) 1.02 (1.55) 1.05 (1.69)	1.04 (2.31) 1.04 (1.70) 1.11 (1.71)	1.10 (2.27) 1.06 (1.81) 1.16 (1.76)	1.14 (2.18) 1.18 (2.14) 1.28 (1.85)	1.29 (2.10) 1.22 (2.17) 1.37 (1.84)			
8						1.02 (2.14) 1.01 (1.47) 1.03 (1.57)	1.05 (2.10) 1.04 (1.78) 1.08 (1.65)	1.09 (2.03) 1.10 (2.04) 1.18 (1.79)	1.15 (2.11) 1.13 (2.27) 1.24 (1.80)	1.26 (2.09) 1.27 (2.63) 1.38 (1.85)				
10					1.01 (2.00) 1.01 (1.54) 1.03 (1.49)	1.03 (1.94) 1.02 (1.74) 1.05 (1.56)	1.09 (1.99) 1.08 (2.17) 1.16 (1.73)	1.13 (1.95) 1.15 (2.60) 1.24 (1.81)	1.19 (2.09) 1.20 (2.70) 1.28 (1.84)					
12					1.01 (1.80) 1.02 (1.83) 1.03 (1.47)	1.04 (1.88) 1.05 (2.06) 1.07 (1.55)	1.10 (1.96) 1.10 (2.53) 1.14 (1.66)	1.19 (2.00) 1.21 (3.17) 1.26 (1.78)	1.26 (2.06) 1.29 (3.35) 1.43 (1.94)					
16				1.01 (1.78) 1.01 (1.85) 1.04 (1.44)	1.03 (1.69) 1.03 (2.04) 1.09 (1.54)	1.09 (1.87) 1.08 (2.60) 1.15 (1.63)	1.19 (1.95) 1.18 (3.45) 1.26 (1.81)	1.28 (2.10) 1.31 (4.10) 1.38 (1.92)						
20			1.00 (1.63) 1.00 (1.65) 1.02 (1.37)	1.03 (1.65) 1.05 (2.55) 1.07 (1.49)	1.06 (1.74) 1.09 (3.04) 1.12 (1.60)	1.14 (1.94) 1.13 (3.17) 1.20 (1.75)	1.25 (2.10) 1.28 (4.4) 1.34 (1.87)							
24			1.01 (1.57) 1.03 (2.34) 1.03 (1.34)	1.06 (1.79) 1.07 (2.86) 1.09 (1.50)	1.15 (1.94) 1.15 (3.97) 1.23 (1.82)	1.18 (1.99) 1.19 (4.35) 1.32 (1.93)	1.32 (2.22) 1.31 (4.8) 1.51 (2.13)							
32		1.00 (1.53) 1.00 (1.65) 1.01 (1.24)	1.04 (1.67) 1.02 (2.4) 1.10 (1.54)	1.14 (1.94) 1.14 (4.8) 1.20 (1.79)	1.16 (2.00) 1.22 (5.4) 1.30 (1.95)	1.30 (2.22) 1.32 (5.9) 1.41 (2.08)								
40	1.00 (1.70) 1.00 (1.33) 1.00 (1.22)	1.00 (1.50) 1.02 (2.5) 1.02 (1.31)	1.06 (1.72) 1.09 (4.0) 1.17 (1.73)	1.18 (2.02) 1.23 (5.9) 1.28 (1.97)	1.28 (2.30) 1.31 (6.7) 1.45 (2.24)									
48	1.00 (1.72) 1.00 (1.44) 1.00 (1.19)	1.02 (1.59) 1.02 (2.8) 1.08 (1.52)	1.13 (1.98) 1.13 (5.3) 1.17 (1.76)	1.29 (2.41) 1.31 (7.5) 1.36 (2.14)	1.40 (2.51) 1.41 (8.1) 1.54 (2.38)									
56	1.00 (1.70) 1.00 (1.44) 1.00 (1.18)	1.03 (1.63) 1.07 (4.1) 1.10 (1.58)	1.18 (2.16) 1.23 (6.7) 1.28 (2.05)	1.27 (2.30) 1.34 (7.8) 1.42 (2.24)										
64	1.00 (1.72) 1.00 (1.58) 1.00 (1.18)	1.10 (2.05) 1.09 (5.3) 1.14 (1.71)	1.25 (2.44) 1.29 (8.3) 1.35 (2.20)	1.35 (2.60) 1.40 (9.4) 1.53 (2.45)										
72	1.00 (1.69) 1.00 (1.69) 1.00 (1.17)	1.09 (1.92) 1.10 (5.3) 1.18 (1.84)	1.26 (2.45) 1.30 (8.4) 1.42 (2.37)											
80	1.00 (1.70) 1.00 (1.74) 1.00 (1.20)	1.17 (2.26) 1.18 (7.6) 1.24 (2.00)												
96	1.00 (1.76) 1.00 (2.0) 1.00 (1.22)	1.27 (2.64) 1.26 (8.5) 1.26 (2.14)												

Table 1: Space utilization and (speed) for memory of total size 256 cells for the segregated storage and first fit methods and the buddy system, respectively.

Mean Size of Blocks Requested (c)

Mean Rate of Requests (λ)	2	3	4	7	11	17	45
3					1.01 (2.95) 1.01 (1.30) 1.04 (1.92)	1.05 (2.52) 1.06 (1.40) 1.15 (2.01)	1.52 (1.99) 1.51 (1.70) 1.68 (1.90)
7				1.01 (2.34) 1.01 (1.43) 1.04 (1.56)	1.10 (2.05) 1.10 (2.02) 1.16 (1.72)	1.25 (2.16) 1.26 (2.37) 1.35 (1.82)	
14			1.00 (1.91) 1.00 (1.50) 1.01 (1.35)	1.08 (1.83) 1.10 (2.63) 1.16 (1.66)	1.26 (2.05) 1.27 (3.23) 1.39 (1.89)		
28		1.02 (1.64) 1.02 (2.05) 1.05 (1.34)	1.07 (1.76) 1.08 (3.12) 1.13 (1.58)	1.29 (2.15) 1.33 (4.95) 1.43 (2.04)			
50	1.01 (1.49) 1.02 (2.39) 1.04 (1.28)	1.16 (2.14) 1.20 (4.80) 1.20 (1.79)	1.35 (2.43) 1.31 (5.82) 1.38 (2.14)				

Table 2: Space utilization and (speed) for memory of total size 128 cells for the segregated storage and first fit methods and the buddy system, respectively.

Mean Size of Blocks Requested (c)

Mean Rate of Requests (λ)	3	6	9	14	23	34	85
6					1.00 (2.50) 1.00 (1.38) 1.01 (1.71)	1.02 (2.26) 1.02 (1.50) 1.06 (1.72)	1.31 (2.08) 1.35 (2.46) 1.43 (1.90)
14				1.00 (1.81) 1.00 (1.59) 1.01 (1.40)	1.07 (1.82) 1.06 (2.17) 1.11 (1.63)	1.18 (2.03) 1.18 (3.29) 1.31 (1.85)	
28			1.00 (1.53) 1.01 (2.07) 1.00 (1.29)	1.06 (1.76) 1.05 (3.02) 1.12 (1.58)	1.21 (2.13) 1.20 (4.87) 1.37 (2.06)		
56		1.00 (1.47) 1.01 (2.39) 1.03 (1.33)	1.11 (2.05) 1.09 (5.77) 1.15 (1.71)	1.30 (2.52) 1.28 (9.00) 1.44 (2.32)			
104	1.00 (1.72) 1.00 (2.21) 1.00 (1.23)	1.15 (2.44) 1.16 (9.89) 1.23 (2.10)	1.38 (2.98) 1.34 (13.58) 1.44 (2.57)				

Table 3: Space utilization and (speed) for memory of total size 512 cells for the segregated storage and first fit methods and the buddy system, respectively.

Table 4: The Results of Multiple and Longer Runs

number of requests filled	$\lambda = 1$. ave-size = 256	$\lambda = 16$. ave size = 16	$\lambda = 24$. ave-size 16	$\lambda = 64$. ave-size = 4
400	1.89 ± .03 (1.50 ± .02)	1.19 ± .03 (2.07 ± .04)	1.31 ± .03 (2.22 ± .06)	1.08 + .02 (2.12 ± .13)
	1.89 ± .03 (1.13 ± .01)	1.21 + .03 (3.59 ± .40)	1.31 ± .04 (4.77 ± .42)	1.07 ± .02 (4.23 ± .81)
	2.16 ± .07 (1.69 ± .04)	1.25 ± .05 (1.81 ± .09)	1.39 ± .11 (1.97 ± .16)	1.15 ± .04 (1.80 ± .11)
800	1.90 ± .02 (1.51 ± .02)	1.19 ± .02 (2.02 ± .03)	1.32 ± .02 (2.17 ± .01)	1.09 ± .01 (1.95 ± .06)
	1.90 ± .02 (1.13 ± .01)	1.21 ± .02 (3.74 ± .18)	1.33 ± .04 (5.09 ± .58)	1.09 ± .01 (4.86 ± .32)
	2.13 ± .07 (1.69 ± .03)	1.28 ± .03 (1.83 ± .03)	1.41 ± .08 (1.99 ± .09)	1.15 ± .03 (1.73 ± .08)
1200	1.88 ± .06 (1.49 ± .03)	1.18 ± .02 (2.00 ± .01)	1.33 ± .02 (2.21 ± .03)	1.10 ± .01 (1.93 ± .07)
	1.88 ± .06 (1.13 ± .01)	1.21 ± .02 (3.58 ± .12)	1.33 ± .03 (5.11 ± .57)	1.10 ± .01 (5.29 ± .34)
	2.15 + .11 (1.69 ± .02)	1.27 ± .01 (1.81 ± .02)	1.43 ± .07 (2.02 ± .08)	1.16 ± .03 (1.73 ± .09)
1600	1.90 ± .04 (1.49 ± .02)	1.20 ± .01 (2.01 ± .01)	1.33 ± .01 (2.22 ± .02)	1.10 ± .01 (1.86 ± .03)
	1.90 ± .04 (1.14 ± .02)	1.21 ± .02 (3.59 ± .12)	1.34 ± .02 (5.24 ± .40)	1.09 ± .01 (5.04 ± .24)
	2.12 ± .07 (1.70 ± .02)	1.28 ± .01 (1.82 ± .02)	1.42 ± .04 (2.01 ± .04)	1.15 ± .02 (1.70 ± .07)
2000	1.92 ± .03 (1.48 ± .02)	1.20 ± .01 (2.01 ± .02)	1.33 ± .02 (2.22 ± .03)	1.09 ± .01 (1.80 ± .04)
	1.92 ± .03 (1.13 ± .01)	1.21 ± .01 (3.64 ± .08)	1.34 ± .01 (5.21 ± .28)	1.10 ± .01 (5.14 ± .02)
	2.12 ± .05 (1.70 ± .02)	1.27 ± .01 (1.81 ± .02)	1.42 ± .03 (2.02 ± .03)	1.15 ± .02 (1.69 ± .06)
2400	1.92 ± .02 (1.48 ± .02)	1.20 ± .01 (2.00 ± .03)	1.33 ± .02 (2.21 ± .03)	1.09 ± .01 (1.79 ± .05)
	1.92 ± .02 (1.13 ± .01)	1.21 ± .02 (3.59 ± .08)	1.34 ± .01 (5.19 ± .17)	1.09 ± .01 (5.14 ± .19)
	2.12 ± .05 (1.70 ± .02)	1.27 ± .01 (1.81 ± .02)	1.42 ± .03 (2.01 ± .02)	1.14 ± .02 (1.67 ± .06)
3200	1.92 ± .03 (1.48 ± .02)	1.19 ± .00 (1.98 ± .01)	1.33 ± .01 (2.21 ± .01)	1.09 ± .01 (1.75 ± .06)
	1.92 ± .03 (1.14 ± .01)	1.20 ± .00 (3.54 ± .04)	1.34 ± .01 (5.21 ± .19)	1.14 ± .00 (5.05 ± .12)
	2.12 ± .02 (1.70 ± .01)	1.27 ± .01 (1.81 ± .02)	1.44 ± .02 (2.03 ± .03)	1.14 ± .02 (1.66 ± .05)
4000	1.94 ± .04 (1.47 ± .03)	1.19 ± .01 (1.98 ± .02)	1.34 ± .01 (2.22 ± .02)	1.09 ± .01 (1.75 ± .03)
	1.94 ± .04 (1.14 ± .01)	1.20 ± .01 (3.50 ± .08)	1.33 ± .01 (5.19 ± .14)	1.09 ± .00 (5.05 ± .05)
	2.12 ± .02 (1.69 ± .01)	1.28 ± .02 (1.81 ± .03)	1.44 ± .02 (2.04 ± .02)	1.14 ± .01 (1.65 ± .04)
4800	1.94 ± .04 (1.47 ± .02)	1.19 ± .01 (1.98 ± .02)	1.34 ± .01 (2.21 ± .01)	1.09 ± .00 (1.75 ± .01)
	1.94 ± .04 (1.14 ± .01)	1.19 ± .01 (3.47 ± .05)	1.34 ± .01 (5.21 ± .14)	1.09 ± .01 (5.15 ± .09)
	2.12 ± .01 (1.70 ± .01)	1.28 ± .02 (1.81 ± .04)	1.45 ± .01 (2.03 ± .01)	1.14 ± .02 (1.65 ± .05)