Buddy Systems with Selective Splitting

N. M. PITMAN*, F. W. BURTON† AND E. W. HADDON*‡

* *School of Computing Studies and Accountancy, University of East Anglia, Norwich NR4 7TJ*

† *Department of Electrical and Computer Engineering, University of Colorado at Denver, Denver, Colorado 80202, U.S.A.*

A new approach to the buddy system for dynamic storage allocation is presented in which a block of a given size may be split in more than one way. It is demonstrated that this approach can be used to reduce the memory fragmentation of a buddy system.

Received April 1983

1. INTRODUCTION

In various computer applications there exists a need for dynamic storage allocation. Some sufficiently large contiguous memory pool is devoted to the transient storage of variable sized blocks of data. When the storage allocator receives a request to store a particular block it must find, and allocate, a block of unused store sufficiently large to accommodate the block of data. Subsequently, the block of data is no longer needed and the storage block can be returned to an available space list. It is generally impractical to relocate the blocks of live data to maintain a contiguous block of available space, and attention is thus focused on managing the fragmented available space.

Allocation by 'best-fit' involves a lengthy search whilst 'first-fit' is more wasteful of space.[1] The buddy system, introduced by Knowlton,[2] 1965, provides a more satisfactory solution to the problem. The binary buddy system of Knowlton has a memory pool of size 2^m (which may be any suitable measure of bytes, words, etc.). A request for storage of a block of size R requires the allocation of a block of size 2^k where $2^{k-1} < R \leqslant 2^k$ (where R includes any necessary housekeeping data). The available space list is examined for a block of size 2^k. If such a block is found it is allocated, otherwise the available space list is scanned, sequentially, for a larger block of size 2^{k+i}, $i > 0$, which will be split into two equal parts, of sizes 2^{k+i-1}, called buddies. One of these buddies will be broken down, if necessary, until a block of size 2^k is produced. All other unused buddies will be placed on the available space list. One buddy is then allocated and the other is placed on the available space list. When a block is released it is recombined with its buddy if that block is free – it may subsequently have been allocated or split further. The recombination process continues as far as possible and the block so formed is placed on the available space list.

A buddy system must be timewise-efficient in allocation of space and similarly in returning used blocks to the free space list, which requires rapid recognition of respective buddies. It must also be efficient in its utilization of the total memory pool, and fragmentation is the difficulty in this respect. A buddy system has a predetermined set of block sizes which can be allocated. The binary buddy system can allocate blocks of sizes 1, 2, 4, 8, 16, ... which means that if a request is received to store a block of size 11 the system must allocate a block of size 16 and then, for the lifetime of the block of size 11, space of size 5 is unused and unavailable in the allocated block of size 16.

‡ To whom correspondence should be addressed.

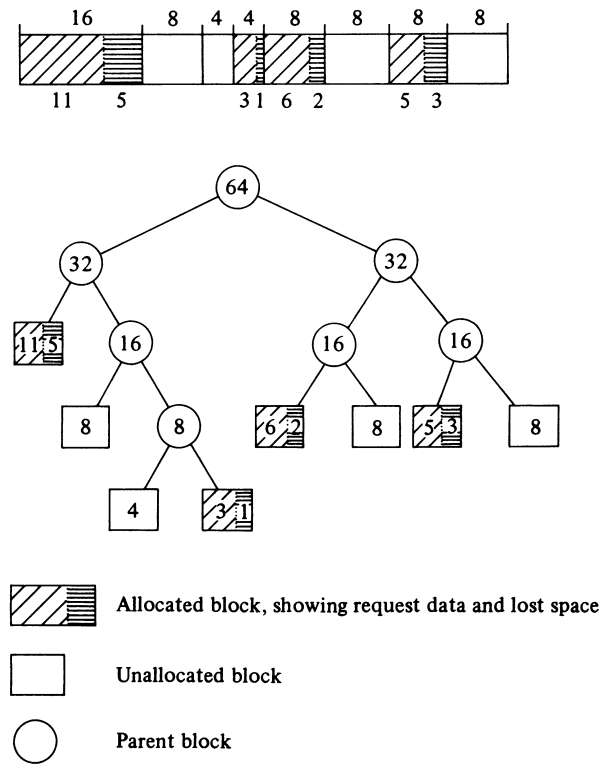


Figure 1. Typical usage of a memory pool of size 64 and a tree representation of the buddy structure.

This is termed *internal fragmentation*. At any time the storage allocator has lists of the available space for each allocatable block size, but it is possible that a request is received for storage of a block which is larger than any currently allocatable block, for example there may be several available blocks of size 16 but no larger block available. A request to store a block of size 20 would then fail even though there is, in total, ample free space. *External fragmentation* describes this scattering of available space in blocks throughout the memory pool which cannot be combined because they are not respective buddies. (Note that a dynamic allocation scheme based on relocation could allocate blocks of the requested size and would eliminate both internal and external fragmentation.) A buddy system can therefore be judged also on its spacewise efficiency in respect of the two types of fragmentation. Internal fragmentation can be monitored continuously and defined as the proportion of allocated space which is holding data. External fragmentation is only relevant when a request cannot be satisfied, and this can be defined in terms of the proportion of the

total memory pool which is unallocated when a request fails. Figure 1 illustrates internal and external fragmentation on a memory pool of size 64. As shown, requested data blocks of sizes 11, 3, 6 and 5 are stored in allocated blocks of sizes 16, 4, 8 and 8. Now a request to store a block of size 10 will fail – the adjacent free blocks of sizes 8 and 4 cannot be used because they are not brother buddies as is shown by the tree structure diagram. We have at this stage

storage requested = 11+3+6+5 = 25
storage allocated = 16+4+8+8 = 36
storage unallocated = 8+4+8+8 = 28
internal fragmentation = (36−25)/36 = 0.31
external fragmentation = 28/64 = 0.44

Since Knowlton introduced the original binary buddy system several variations on the technique have been presented. Hirschberg designed a Fibonacci buddy system in which allocatable block sizes are Fibonacci numbers.[3] Shen and Peterson introduced a weighted buddy system whose block sizes are 2^k and 3.2^k.[4] Peterson and Norman discussed a generalised buddy system in terms of general recurrence relationships between the block sizes.[5] The performance of any buddy system depends upon the distribution of requested block sizes and the frequency of requests for each size. Results of comparative studies of the binary, Fibonacci, weighted and F-2 (Fibonacci type) systems have been reported by Peterson and Norman, and by Russell.[6] Norman,[7] and Burton,[8] discussed the tailoring of the buddy system for applications in which the request distribution might be known, *a priori*, or the storage device (disk) might impose some physical restrictions.

In each of the buddy systems previously proposed, an available block of a particular size could only be split into two buddies in one way. We propose a technique of selective splitting of blocks in which, for example, a block of size 16 might be split into buddies of size 8 and 8 or into buddies of size 10 and 6. The latter choice would give less internal fragmentation if allocating for a request of size 5 and also leave a larger block for subsequent allocation. Simulation studies by Peterson and Norman suggest that, for previous buddy systems, as internal fragmentation decreases, external fragmentation increases and the amount of unusable memory from both sources (total fragmentation) remains fairly constant. We present results of our simulation studies on the same request distributions[9, 5, 7] which suggest that selective splitting can make a significant improvement.

2. OUTLINE OF A BUDDY SYSTEM

Generally, a buddy system is based upon a sequence of numbers $S_1, S_2, \ldots, S_n$ (allocatable block sizes) with $S_1 < S_2 < \ldots < S_n$ which satisfy a set of recurrence relations

$$S_i = S_{i-f(i)} + S_{i-g(i)}, \quad g(i) \geqslant f(i) > 0, \qquad (1)$$

where $f(i)$, $g(i)$ are meaningful integral functions. Table 1 gives values for the systems mentioned in Section 1. More complex systems can be constructed.

Control of the memory space is achieved through an available-space list of n elements, accessed through the size index i for each allocatable block size. Each element of this list contains two pointers to the front and rear of a doubly linked list chaining the storage addresses of all free blocks of size S_i.[2] General algorithms for the allocation of storage and the return of a block to the available-space list are presented below.

Table 1. $f(i)$ and $g(i)$ functions for simple buddy systems

Buddy system	$f(i)$	$g(i)$
Binary	1	1
Fibonacci	1	2
F-2	1	3
Weighted	1	4, i even, $i > 6$ 3, i odd, $i > 5$ 2, $i = 4$ 1, $i = 3$

Allocation of storage

A1. A request is received for a block in which to store R units, where R includes the space required for housekeeping data.
A2. The list of block sizes is scanned to obtain the target block size S_t where $S_t \geqslant R$.
A3. If a block of size S_t is available it is removed from the available space list and allocated. Otherwise proceed to step A4.
A4. The available space list is scanned from size index t for an available candidate block S_c which would split into buddies of sizes $S_c - f(c)$ and $S_c - g(c)$ with at least one buddy being not smaller than S_t.
A5. A split of the current candidate block is considered. If one buddy would be of size S_t then the split is made, that buddy is allocated and the other buddy is placed on the appropriate free block chain. Otherwise continue to step A6.
A6. If a split of the current candidate block would give buddies which are both smaller than S_t then the current candidate block is allocated. Otherwise the split is made, with the smaller buddy being retained as a new candidate block S_c and the other buddy being placed on a free block chain. Repeat from step A5.

Returning a freed block

R1. A block F_a is to be returned to the available space list.
R2. The block F_b, the buddy of F_a, is located.
R3. If F_b has been split further into sub-buddies, or F_b has been allocated, then F_a is returned to the appropriate free block chain. Otherwise F_a and F_b are recombined to form a new free block F_a and step R2 is repeated.

Note that the merging process in step R3 is the inverse of a splitting operation performed in A4 or A5: two adjacent free blocks in the memory pool can only be combined if each is the brother buddy of the other.

The recombination of buddies in any system requires the preservation in a block, whether allocated or available, of some housekeeping information. This includes a SIZE field, containing the size index, a one-bit TAG field to indicate the block's status (allocated or available) and the location of the buddy block. Although a block at a buddy address may be available for allocation it may be unavailable for recombination because it is now only a sub-buddy if the original buddy – see Fig. 2. The original block A is split into buddy blocks B and C, and

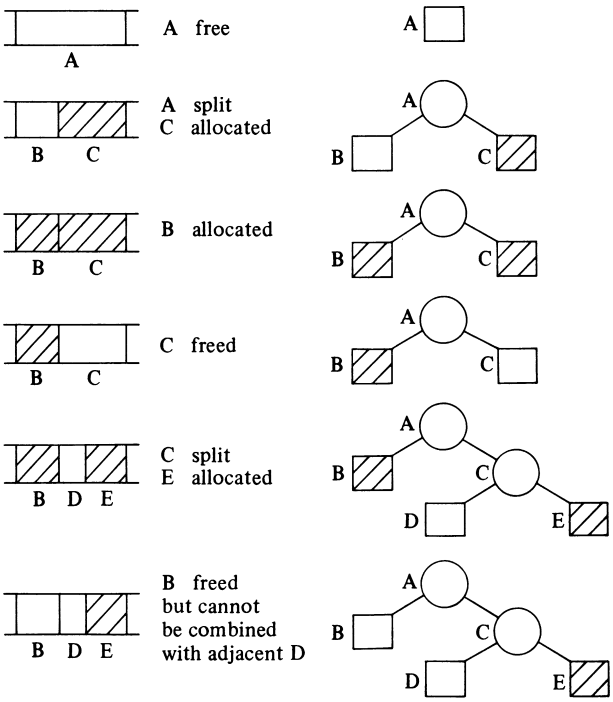


Figure 2. Stages in the life history of a block of the memory pool.

C is allocated. Subsequently, B is allocated and then C is freed, but A cannot be re-formed since B is not free. Now let C be split into buddy blocks D and E, and E be allocated. If B now becomes free the address of its buddy accesses D, which is free, but recombination cannot take place since D is not the brother buddy of B. The technique introduced by Cranston and Thomas,[11] and extended by Burton,[8] which involves the use of a buddy bit, a memory bit and two-bit *I*-field, is used to enable the recombination of blocks in the work described in this paper.

3. BUDDY SYSTEMS WITH SELECTIVE SPLITTING

The studies of Peterson and Norman suggest that the weighted buddy system gives a reduction in internal fragmentation when compared to other systems, largely due to the fact that it provides nearly twice as many available block sizes as in a standard buddy system. The greater number of block sizes allows a better fit of requested blocks, but conversely smaller and less usable buddies may be created in splitting blocks, thereby giving higher external fragmentation.

The technique of selective splitting which we propose similarly offers a greater number of available block sizes. The splitting rule (1) of the usual buddy systems is now replaced by the alternative recurrence relations

$$\left.\begin{aligned} &S_i = S_{i-f(i)} + S_{i-g(i)}, \quad g(i) \geqslant f(i) > 0, \\ \text{and} \quad &S_i = S_{i-k(i)} + S_{i-l(i)}, \quad l(i) \geqslant k(i) > 0. \end{aligned}\right\} \qquad (2)$$

If $f(i) \neq k(i)$ and $g(i) \neq l(i)$ there is a choice of two splittings of the block of size S_i, but we need not have an alternative splitting for every S_i. In the examples of Tables 2–5 the absence of an alternative splitting is shown by null entries for $k(i)$ and $l(i)$, but within the form (2) we could define $f(i) = k(i)$ and $g(i) = l(i)$ when there is only one split for that value of i.

Peterson and Norman, and also Russell, show that the performance of conventional buddy systems varies according to the distribution of requested block sizes. If a buddy system is being designed for a particular environment in which there is *a priori* knowledge of the likely request distribution, it is possible to tailor the buddy system to improve the performance.[7,8] The main simulation studies reported in this paper are for selective splitting alternatives on the weighted buddy system and do not introduce any new block sizes. The design of a

Table 2. Block sizes and splitting for disk storage (Burton)

i	S_i (block size)	$f(i)$	$g(i)$
1	1	—	—
2	2	1	1
3	4	1	1
4	6	1	2
5	10	1	2
6	20	1	1

Table 3. Two selective splitting schemes for disk storage

i	S_i	$f(i)$	$g(i)$	$k(i)$	$l(i)$
			(*a*)		
1	1	—	—	—	—
2	2	1	1	—	—
3	4	1	1	—	—
4	6	1	2	—	—
5	8	1	3	2	2
6	10	1	4	2	3
7	20	1	1	1	1
			(*b*)		
1	1	—	—	—	—
2	2	1	1	—	—
3	3	1	2	—	—
4	4	1	3	2	2
5	6	1	3	2	2
6	8	1	4	2	2
7	10	1	5	2	3
8	20	1	1	1	1

Table 4. A selective splitting scheme based on Fibonacci

i	S_i	$f(i)$	$g(i)$	$k(i)$	$l(i)$
1	1	—	—	—	—
2	2	1	1	—	—
3	3	1	2	—	—
4	4	1	3	2	2
5	5	1	4	2	3
6	6	1	5	2	4
7	8	1	5	2	4
8	10	1	6	2	4
9	13	1	6	2	4
10	16	1	7	2	4
11	21	1	6	2	4
12	26	1	7	2	4
13	34	1	6	2	4
14	42	1	7	2	4
15	55	1	6	2	4 *

* Note the emergence of a cycle with period 2.

Table 5. A selective splitting scheme including blocks of size 3^n

i	S_i	$f(i)$	$g(i)$	$k(i)$	$l(i)$	
1	1	—	—	—	—	
2	2	1	1	—	—	
3	3	1	2	—	—	
4	4	1	3	2	2	
5	6	1	3	2	2	
6	7	1	5	2	3	
7	9	1	5	2	4	
8	12	1	5	3	3	
9	18	1	4	2	2	
10	21	1	7	2	3	
11	27	1	6	2	4	
12	36	1	5	3	3	*
13	54	1	4	2	2	
14	63	1	7	2	3	
15	81	1	6	2	4	

* Note the emergence of a cycle with period 4.

selective splitting system is not, however, limited in its block sizes, and we give examples to illustrate this point.

Burton considered the problem of a buddy system for disk storage allocation where there are 10 tracks per cylinder, and for efficiency a logical block would only be allowed to overlap a cylinder boundary if it were too large to fit into a single cylinder. The buddy system must have blocks of size 1 and 10 (1024 and 10240 words in Burton) and further, any block of size greater than 10 must be composed only of blocks of size 10 or greater. The solution given by Burton is shown in Table 2. We show two selective splitting schemes in Table 3*a* and Table 3*b*, both of which include all the splits of the original scheme of Table 2. They also provide the addition of new block sizes ($S_5 = 8$ in Table 3*a* and $S_3 = 3$ and $S_6 = 8$ in Table 3*b*), which should reduce internal fragmentation. In all cases a block of size 20 can only be split into two blocks of size 10.

If we consider the Fibonacci system and extend the number of block sizes by including blocks which are twice the size of the existing Fibonacci blocks we can produce the selective splitting system shown in Table 4. The original Fibonacci system is included in this scheme, which could be varied further by choosing to split some of the even-sized blocks into two equal blocks.

If, when designing a buddy system, it were known that a significant number of request blocks would be of sizes given by powers of 3, the scheme given in Table 5 would meet the requirement. We notice that a block of size 27 can be obtained in many ways as shown in Fig. 3. Similarly there are many ways to obtain a block of size 9.

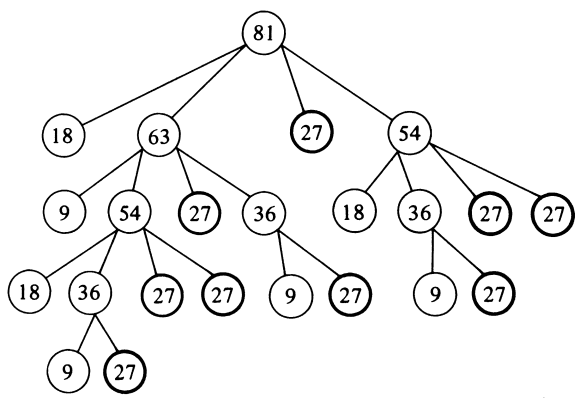
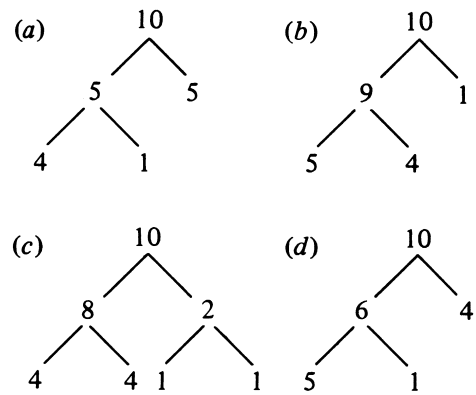


Figure 3. Alternative splits yielding a block of 27 from an original block of 81 in the scheme of Table 5.

For another example, consider a situation in which it were known that request blocks of size 6 would be fairly common. The binary buddy system has block sizes 1, 2, 4, 8, 16, ..., but if we provided the additional splits of $16 \rightarrow 12+4$, $12 \rightarrow 8+4$ or $12 \rightarrow 6+6$ and $8 \rightarrow 6+2$ we could expect a reduction in internal fragmentation.

Since the performance of a buddy system varies with the request distribution and, moreover, varies according to the sequence of requests within a particular distribution, there is no 'best' buddy system. The examples above illustrate how selective splitting could tailor a system towards the requests when we have some limited information regarding the expected requests. If more specific knowledge of the request distribution is available we could hope to approach an optimal buddy system by designing it heuristically.

For example, consider the distribution of memory requests on an IBM 360 CP-67 system. The relative frequencies of request blocks are shown in Table 6 in order of decreasing frequency. (Note that there is no request for blocks of any other size.) We observe that approximately 85% of requests are for blocks of sizes 4,5,29,8 or 1 and it is obviously essential that the buddy system can offer blocks of these sizes. We consider ways in which the desired small blocks of sizes 1,4 and 5 could be produced from larger blocks which also feature in Table 6. We see the possible splits



of which we may select two. (*d*) appears less attractive since a block of 6 will hardly ever be required, and although a block of 8 occurs in (*c*) we can also obtain blocks of 8 and 1 from the block of 9 which occurs in (*b*). We therefore choose (*a*) and (*b*) as useful splits of a block of size 10, and with the alternative for a block of 9 we have so far

S_i	$f(i)$	$g(i)$	$k(i)$	$l(i)$
1	—	—	—	—
4	—	—	—	—
5	1	2	—	—
8	2	2	—	—
9	2	3	1	4
10	3	3	1	5
29				

Table 6. Actual request distributions

Univ. of Maryland (UM)		Brigham Young Univ. (BYU)		IBM CP-67 System (CP67)	
Block size	Cumulative distribution function	Block size	Cumulative distribution function	Block size	Probability density function
2	0.000	3	0.000	4	0.248
8	0.360	16	0.064	5	0.219
10	0.440	32	0.168	29	0.156
15	0.540	48	0.276	8	0.112
25	0.840	64	0.400	1	0.111
30	0.940	80	0.458	10	0.041
35	0.965	96	0.627	3	0.037
40	0.975	112	0.826	9	0.020
50	0.985	128	0.949	18	0.019
70	0.993	144	0.953	17	0.009
100	0.996	160	0.957	7	0.006
200	1.000	176	0.961	31	0.004
		192	0.964	6	0.003
		208	0.970	23	0.003
		224	0.983	50	0.003
		256	0.994	2	0.002
		272	0.996	11	0.002
		304	0.998	12	0.002
		352	0.999	21	0.002
		511	1.000	27	0.001
Continuous distribution, average request size 15.99		Continuous distribution, average request size 80.26		Discrete distribution, average request size 9.34	

A block of size 3 appears useful and will allow a split of 8 into 5+3. We could then allow a block of size 4 to split into 3+1. A block of size 29 will give a split into 19+10, with the 19 giving 10+9. We can also provide a block of size 8 by the alternative split 29 = 21+8.

Since blocks of sizes 21 and 50 occur in the request distribution, it is tempting to include 50 = 21+29. However, neither 21 nor 50 is a frequently requested size, and the consequence of this splitting would be the generation of a large number of unwanted blocks of size 21. We therefore introduce a block of size 58, which will hold the few requested blocks of size 50, and which will always split into two blocks of size 29. Tests showed that the effect of this decision was a reduction in external fragmentation from 0.27 to 0.09. Finally, we include the unique splitting 21 = 12+9 and the alternatives 12 = 9+3 and 12 = 8+4. We now have the scheme

S_i	$f(i)$	$g(i)$	$k(i)$	$l(i)$
1	—	—	—	—
3	—	—	—	—
4	1	2	—	—
5	1	3	—	—
8	1	3	2	2
9	1	5	2	3
10	1	6	3	3
12	2	6	3	5
19	2	3	—	—
21	2	4	—	—
29	1	6	2	4
58	1	1	—	—

If we adopt the usual measure of expected internal fragmentation as

$$\sum_{i=1}^{m} p_i\,(S_i - i) \Big/ \sum_{i=1}^{m} p_i i,$$

where p_i is the relative frequency of a request block of size i and S_i is the allocated block size, the expected internal fragmentation is 0.016, i.e. only between 1 and 2% of allocated space is not holding data. This compares with values between 10 and 18% for internal fragmentation using the binary, Fibonacci, F−2 and weighted buddy systems. (See results in Table 8 which confirm the expected internal fragmentation performance.) External fragmentation can only be determined by simulation experiments, but we would not expect a high value because the most frequently requested block sizes can be obtained in several ways from larger blocks.

4. THE WEIGHTED BUDDY SYSTEM WITH SELECTIVE SPLITTING

Our simulation testing of a selective splitting scheme used the weighted buddy system as a test case. The details of the selective splitting are given in Table 7, in which the $f(i)$ and $g(i)$ entries represent the original splitting rules and $k(i)$ and $l(i)$ give our alternative splits. We note again that for a given range we have exactly the same set of allocatable block sizes as in the original weighted buddy system. We would therefore expect to obtain the same values for internal fragmentation as were reported in other investigations when using the same request distributions.

Table 7. A selective splitting scheme for the weighted buddy system

i	S_i	$f(i)$	$g(i)$	$k(i)$	$l(i)$
1	1	—	—	—	—
2	2	1	1	—	—
3	3	1	2	—	—
4	4	1	3	2	2
5	6	1	3	2	2
6	8	1	4	2	2
7	12	1	3	2	2
8	16	1	4	2	2
9	24	1	3	2	2
10	32	1	4	2	2
11	48	1	3	2	2
12	64	1	4	2	2
13	96	1	3	2	2
14	128	1	4	2	2
15	192	1	3	2	2
16	256	1	4	2	2

Theoretical results have been presented by Russell for expected internal fragmentation with simple buddy systems (binary, Fibonacci and F−2). The weighted buddy system is more complex because $g(i)$ is not constant. Analysis of external fragmentation is less tractable, and in buddy systems with a unique split of any block the level of external fragmentation is a direct consequence of the buddy system and the request stream. Only in buddy systems with selective splitting can the level of external fragmentation be influenced by run-time decisions. Selective splitting may improve the external fragmentation if an acceptable technique can be found for choosing one or other of the alternative splits.

Consider a situation in which we have a request block of size $R = 5$ and the smallest available candidate block is of size $S_c = 16$. The target block size for allocation is $S_t = 6$. In the original weighted buddy system the block S_c would be split as shown in Fig. 4*a*, producing two free blocks of size 4 and one free block of size 2. Our selective splitting includes this split and allows an alternative split of the candidate block of size 16 into two blocks of size 8, one of which would obviously be split to give a block of size 6 as in Fig. 4*b*. We thus create one free block of size 8 and one free block of size 2. A third possibility is to split the candidate block into blocks of 12 and 4 and then subdivide the 12 into two blocks of 6 as in Fig. 4*c*. This gives one free block of size 6 and one of size 4. We wish to choose one of the three options to advantage. We can advance two arguments against the original weighted buddy split, option 1. First, it entails three splits, whereas options 2 and 3 only make two. The fewer the number of splits, the faster is the execution of the system. Likewise, more splits generally give smaller free blocks, which are less likely to be allocated unless the request distribution is strongly biased towards small blocks. The choice between options 2 and 3 is harder since both have two splits. Option 2 has a larger free block, size 8, which is more versatile since it can be used as it is or split into two blocks of 6 and 2 or split into two blocks of 4. The choice may influence the level of external fragmentation, but it is obvious that the optimal choice is a complex function of the current state and the total future sequence of requests. A refined system could base the decision upon

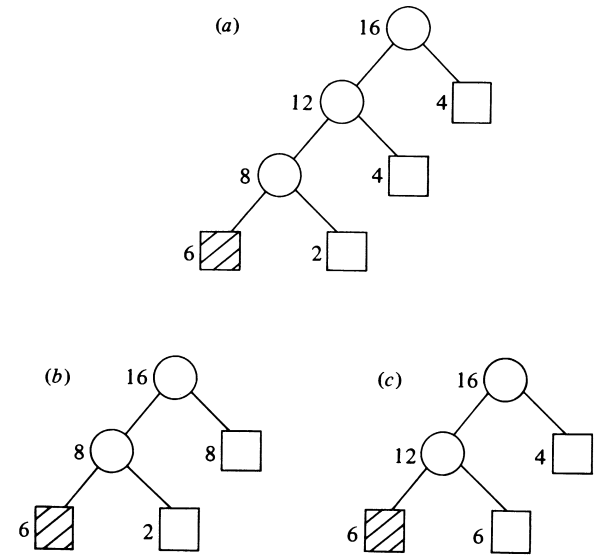


Figure 4. Splittings for weighted buddy system with $R = 5$, $S_t = 6$, $S_c = 16$. (*a*) Option 1 – the only splitting in the original weighted buddy system. (*b*) Selective splitting – Option 2. (*c*) Selective splitting – Option 3.

the number of free blocks which already exist of the sizes which would be created by the alternative splits. Knowledge of the frequency distribution of requests (which could be accumulated during the run of the buddy system) could assist in deciding which free blocks were most likely to be useful. However, the greater the number of factors which are considered in making a decision, the slower will be the execution time of the allocation.

We conducted a limited investigation of strategies for choosing a particular split, using identical request sequences generated randomly from the test distributions using various seeds to initiate the sequences, but without consideration of the current state of free block lists or cumulative distributions of requests. These tests showed that the strategy of minimising the number of splits was preferable. Different methods of choosing between equal minimum split situations were tried, but overall the marginally most successful method was to choose the split which minimised the difference between the largest and smallest free buddies which would be created. Thus in the above example option 3 would be chosen. The argument in favour of option 2, based on the versatility of the block of size 8, could well be valid locally, but the given strategy appeared best globally.

5. THE SIMULATION STUDY

A simulation study for memory management must assess efficiency both in terms of memory utilisation and overall running time. The memory utilisation requires statistics to be gathered for internal and external fragmentation whilst run time is inversely proportional to the number of splits, or combinations, since these will be the same in the equilibrium state.

Previous studies have not been entirely consistent in their simulation procedures though results have been comparable. We based our design on Peterson and Norman, who had given comprehensive results for simple buddy systems. Although we were concerned with the effect of selective splitting, our testing program was

Table 8. Simulation results

Buddy system	Internal fragmentation	External fragmentation	Total fragmentation	Average no. of splits	Average no. of searches
		University of Maryland request distribution			
Binary	0.28	0.05	0.32	0.32	1.32
Fibonacci	0.20	0.09	0.27	0.41	1.56
Weighted	0.14	0.23	0.34	0.83	1.94
Weighted SS	0.14	0.10	0.23	0.52	1.84
		Brigham Young University request distribution			
Binary	0.22	0.08	0.28	0.39	1.39
Fibonacci	0.22	0.14	0.33	0.57	1.80
Weighted	0.13	0.30	0.39	1.02	2.20
Weighted SS	0.13	0.15	0.26	0.60	1.99
		CP-67 request distribution			
Binary	0.18	0.06	0.23	0.24	1.24
Fibonacci	0.13	0.12	0.23	0.44	1.60
Weighted	0.10	0.20	0.28	0.58	1.68
Weighted SS	0.10	0.08	0.17	0.32	1.54
Norman	0.03	0.15	0.18	0.37	1.63
PBH	0.02	0.09	0.11	0.38	1.73

constructed so that we could also simulate the simple buddy systems and thereby validate our work against earlier results.

External fragmentation is only meaningful when there is no available block to satisfy a request and the system is said to have overflowed. We therefore measure the level of external fragmentation at every occurrence of overflow and measure this aspect of the buddy system by the average of these values. In order to assess the overall utilisation of memory the internal fragmentation statistic is also gathered at overflow, although there is always a meaningful value associated with internal fragmentation. The measures of fragmentation must be functions of the buddy system and the request distribution, but independent of the size of the memory pool and memory residence times. We define internal fragmentation as the ratio

$$\frac{\text{total memory allocated} - \text{total memory requested}}{\text{total memory allocated}}$$

and external fragmentation as the ratio

$$\frac{\text{unallocated memory at overflow}}{\text{total memory size}}$$

Each of these is a normalised measure but with respect to different bases. If we define total fragmentation to be the proportion of the total memory size which is not actively holding data we have

$$\text{total fragmentation} = (1 - \text{external}) * \text{internal} + \text{external}.$$

These ratios are then independent of memory size.

A practical difficulty arises when attempting to measure external fragmentation because it only occurs at overflow. The frequency of occurrence of overflow, for a specific application, is obviously a function of the size of the memory pool. If speed of execution were the dominant factor and memory were freely available, an excessively large memory pool would make delays due to overflow extremely unlikely, at the expense of poor utilisation of the memory space. Conversely, a small memory pool will give frequent overflow. In order to generate overflow and produce external fragmentation values which would be independent of memory residence times and comparable across different buddy systems and request distributions, we used the same approach as Shen and Peterson, Peterson and Norman. This method uses a simulation timer and associates with a block allocated at time T a residence lifetime L drawn from a uniform distribution from 1 to 10. The allocated block is due to be released at time $T+L$, but overflow is forced by not releasing blocks at the due times. When overflow occurs external and internal fragmentation is measured and blocks released, with incrementation of the simulation timer, until the block which caused overflow can be accommodated. Requests and allocations are then made until overflow again occurs. Whilst this is a contrived situation it does give the necessary independence of memory size and memory residence times. Previous work, like this study, has found it to give consistent results which agree with observations of real systems. Simulation runs typically simulated a memory pool of size 1024 and covered 2000 request allocations. These parameters were varied in the course of the study, but with insignificant effect upon the results.

In addition to measuring fragmentation we also recorded the number of splits and recombinations of blocks and the number of searches (i.e. the number of free block lists examined to find a suitable free block). These two statistics give some measure of the run-time overheads of a buddy system and enable a comparison of different systems in terms of throughput.

6. RESULTS AND CONCLUSIONS

We tested our technique using published data for three actual request distributions. These are distribution for buffer requests for the UNIVAC 1108 Exec8 system at the University of Maryland (UM), the distribution of partition size requests on the IBM 360/65 OS MVT system at Brigham Young University (BYU) and the

distribution of memory requests on an IBM CP-67 system (CP67). For the UM and BYU distributions a cumulative distribution function is given and the probability density function is assumed to be linear between tabulated points. The CP67 distribution is described by a probability density function and there is no request for blocks of sizes other than those shown. The three distributions are given in Table 6.

In Table 8 we show the results of using selective splitting on the weighted buddy system as described in Section 4 and Table 7. We include results from our simulation for the binary, Fibonacci and simple weighted systems, and remark that the results obtained for these three systems are in accord with the results of Peterson and Norman. In all cases, as expected, the internal fragmentation for the weighted system with selective splitting is the same as for the simple weighted system. The main objective of this work was to investigate the effect of selective splitting on external fragmentation, and these tables show that, for the weighted system, a significant reduction can be achieved in this respect with less than half the wastage arising from the simple weighted system. In comparison with the Fibonacci system we have no significant difference in external fragmentation, but the better performance of weighted-with-selective splitting for internal fragmentation is noticeable. For these three distributions the binary system has a low external fragmentation, but is not competitive in respect of internal fragmentation. When we consider total fragmentation the lowest results are shown by weighted-with-selective splitting for all three distributions, if we exclude our tailored system (PBH) for the CP67 distribution. We conclude that this new approach can be applied with advantage in terms of memory utilisation.

The figures for the average number of splits and the average number of searches give some indication of the execution time for those methods to process a sequence of requests. We recall that our weighted system with selective splitting only has the same block sizes as in the weighted system, and for all three distributions we observe a marked decrease in the average number of splits, and hence recombinations, with the selective splitting which can only be attributed to the effect of the options. The average number of searches also compares favourably. There is no marked difference in splits and searches between Fibonacci and weighted-with-selective splitting. The binary system has the lowest figures in all cases. When comparing these values for splits it should be remembered that selective splitting has more work to perform in selecting the split to make and in the housekeeping to record the necessary information regarding buddies. This extra work per split is unlikely to exceed the savings achieved by fewer splits in comparison with the simple weighted system.

In Table 8 for the CP67 request distribution we include the results for Norman's tailored buddy system[7] and for our heuristically designed system (PBH) developed in Section 3. We note that for PBH the internal fragmentation (0.017) is close to our expected value of 0.016 and a little better than Norman achieves. Further justification of the merit of selective splitting is seen in the external fragmentation for PBH of 0.09, markedly better than the 0.15 of Norman.

We noted in Section 1 the observation of Peterson and Norman that, in previous buddy systems, as internal fragmentation decreased external fragmentation increased and total fragmentation remained steady. Table 8 shows that the buddy systems with selective splitting can reduce internal fragmentation without a significant increase in external fragmentation and hence give improved total fragmentation. The PBH scheme for the CP67 distribution is particularly good.

Acknowledgement

The authors wish to thank the referee for his helpful comments and suggestions, and to thank colleagues for useful discussions.

REFERENCES

1. D.E. Knuth, *The Art of Computer Programming*, vol. 1, *Fundamental Algorithms*, 2nd ed. pp. 435–455. Addison-Wesley, Reading, Mass. (1973).
2. K.C. Knowlton, A fast storage allocator. *Communications of the ACM* **8**, (10), 623–625 (1965).
3. D.S. Hirschberg, A class of dynamic memory allocation algorithms. *Communications of the ACM* **16** (10), 615–618 (1973).
4. K.K. Shen and J.L. Peterson, A weighted buddy method for dynamic storage allocation, *Communications of the ACM* **17** (10), 558–562; and Corrigendum, *Communications of the ACM* **18** (4), 202 (1974).
5. J.L. Peterson and T.A. Norman, Buddy systems. *Communications of the ACM* **20**(6), 421–431 (1977).
6. D.L. Russell, Internal fragmentation in a class of buddy systems. *Siam Journal of Computing* (4), 607–621 (1977).
7. T.A. Norman, Tailored buddy systems for dynamic storage allocation. *Proceedings of the Fourth Texas Conference on Computing Systems*, pp. 2B-3. 1–2B-3.5 (1975).
8. F.W. Burton, A buddy system variation for disk storage allocation. *Communications of the ACM* **19** (7), 416–417 (1976).
9. J. Minker, *et al. Analysis of Data Processing Systems.* Technical Report 69–99, University of Maryland, College Park, Maryland (1969).
10. B.H. Margolin *et al.* Analysis of free-storage algorithms. *IBM Systems Journal* **10** (4), 283–304 (1971).
11. B. Cranston and R. Thomas, A simplified recombination scheme for the Fibonacci buddy system. *Communications of the ACM* **18** (6), 331–332 (1975).