Acta Informatica 24, 555–564 (1987)

Worst Case Performance of Weighted Buddy Systems

Shyamal K. Chowdhury and Pradip K. Srimani
Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, USA

Summary. In this paper we have studied the worst case performance of the weighted buddy system of memory management. Specifically we have derived lower bounds of two system parameters NETREQ and NETALLOC for a weighted buddy system in case of unrestricted request sequence and stated a few preliminary results for those parameters in case of allocation only request sequence. We have also given bounds for the same system parameters for exact-fit memory management algorithms to compare the results with those for buddy systems.

1. Introduction

The buddy systems are well known algorithms for dynamic memory management [1–5]. There are, in general, three different buddy systems: binary buddy system, Fibonacci buddy system and the weighted buddy system. The basic idea behind any of these buddy systems is that it provides storage in blocks of some fixed sizes from a pool of available memory and an available space list keeps track of the available blocks. The effectiveness of such algorithms are generally determined [2, 3] by measuring two types of memory wastage: internal and external memory fragmentation. Extensive simulation experiments have been done [2, 4, 5] to investigate the relative performance of these three buddy systems. To summarize the experimental findings very briefly, on the average internal fragmentation of the binary buddy system is larger than that of Fibonacci buddy system which again is larger than that of weighted buddy system. And external fragmentation of binary buddy system is less than that of Fibonacci buddy system which is less than that of weighted buddy system. It also appears from the studies that the total fragmentation of the three systems are almost the same, the difference between the best and the worst being only 5–10% of the total memory. Very little was known about the theoretical analysis of these buddy systems until recently when authors in [1] analyzed the worst case performance of binary and Fibonacci buddy systems. The objective of the

present paper is to analyze the worst case performance of the weighted buddy systems. In Sect. 2 we introduce the necessary basic concepts and in Sect. 3 we present our results on weighted buddy systems. In Sect. 4 we attempt to extend the same analysis for another general class of memory management algorithms (e.g. best-fit, first-fit etc.) where the size of an allocated block is exactly equal to the size of memory request. Section 5 concludes the paper.

2. Basic Concepts

In weighted buddy system blocks may be of sizes 2^k, $0 \leqq k \leqq n$, and 3.2^k, $0 \leqq k \leqq n-2$ when we start with a total memory of size 2^n. So the allowed block sizes are 1, 2, 3, 4, 6, 8, 12, ... Evidently in this buddy system there are nearly twice as many block sizes as are available in binary buddy systems and that is precisely the reason why internal fragmentation is least in this system. The detailed memory management algorithm can be found in [5]. We briefly describe the algorithm here for the sake of completeness. As in binary or Fibonacci buddy method, the total memory consists of 2^n words addressed from 0 to 2^n-1. Initially this memory is unallocated and the entire memory forms the first block. The blocks are split in two different ways depending on the size of the block to be split. A block of size 2^i is split into two blocks of size 3.2^{i-2} and 2^{i-2} respectively and a block of size 3.2^i is split into two blocks of size 2^{i+1} and 2^i respectively. Blocks split from same parent block are called weighted buddies since they are of different sizes. To process an allocation request to size R, the system locates a minimum size block B such that the size of B is not less than that of R. If there are more than one such blocks, then the leftmost is chosen. If no such block is found the request is said to saturate the system. If one such B exists, there are two possibilities: i) B has a size equal to that of B', or ii) B has a size greater than that of B', where B' is defined to be a block of such size that it can accomodate request R but none of its buddies can. In the first case we simply assign the block B to the request. In the second case we remove B from the available space list and split it into two blocks. We continue splitting the leftmost smallest subblock until a block of size B' is obtained when we fulfill the request. Now in order to measure the external as well as total fragmentation we define the following terms as in [1].

A request sequence is a sequence $R_1, R_2, \ldots, R_m$ where a) $R_i > 0$ is a request to allocate a block of memory of size R_i. Let A_i be the minimum size of the buddy system block such that $A_i \geqq R_i$, b) $R_i < 0$ is a request to deallocate a particular memory block (previously allocated). The request that caused this block to be allocated was of size $-R_i$. Let $-A_i$ be the buddy system block size which was allocated to service the request. We here assume that the information as to which block is to be deallocated is available separately. A request sequence is called saturating if the buddy system can satisfy the requests $R_1, \ldots, R_{m-1}$ but cannot satisfy the request R_m. Two kinds of request sequences are considered. A request sequence is called unrestricted when each request R_i may be either an allocation or a deallocation request and is called an allocation only request sequence if all the requests are of allocation type. It may

be noted that the request sequences in real life are in general unrestricted while the allocation only request sequences correspond to reallocation of freed memory blocks (when saturation occurs) in order to reduce external fragmentation. Two quantities NETREQ and NETALLOC are defined as

$$\text{NETREQ} = \sum_{i=1}^{m} R_i,$$

$$\text{NETALLOC} = \sum_{i=1}^{m} A_i.$$

These two quantities NETALLOC and NETREQ can be interpreted as follows: If the total amount of memory in the system is $M = 2^n$, then $(M - \text{NETALLOC}) + A_m$ represents the amount of memory that is not allocated at all. Thus NETALLOC is a measure of what is called external fragmentation [5]. Again when a block A_i is allocated to a request R_i, $(A_i - R_i)$ amount of memory is wasted within block A_i (it may be zero for some blocks). The internal fragmentation, which is defined as total memory wasted within allocated blocks, is then given by $\sum_{i=1}^{m-i} (A_i - R_i) = (\text{NETALLOC} - A_m) - (\text{NETREQ} - R_m)$. Hence total fragmentation, which is defined as sum of internal and external fragmentation, is given by $M - (\text{NETREQ} - R_m)$. Thus the two parameters, NETALLOC and NETREQ, completely characterize both types of fragmentation in the system for any memory management scheme. Authors in [1] determined the worst case values of NETREQ and NETALLOC for binary and Fibonacci buddy systems in case of both unrestricted and allocation only request sequences. We attempt to do the same for weighted buddy systems in the following section.

3. Weighted Buddy Systems

Theorem 1. *For a weighted buddy system (total memory size 2^n) and a saturating unrestricted request sequence,*

a) $\text{NETREQ} \geqq (1/3)\cdot(-1)^{\lfloor n/2 \rfloor} + (1/3)\cdot 2^{\lceil n/2 \rceil + 1} + 3\cdot 2^{\lfloor n/2 \rfloor - 2} + 1$ *if R_m is of such a size that*

$$2^{n-1} \geqq R_m > 3.2^{n-i-2}, \quad 0 \leqq i \leqq n-2$$

b) $\text{NETREQ} \geqq (1/3)\cdot(-1)^{\lfloor n/2 \rfloor} + (1/3)\cdot 2^{\lfloor n/2 \rfloor + 1} + 2^{\lceil n/2 \rceil - 1} + 1$ *if R_m is of such a size that*

$$3.2^{n-i-2} \geqq R_m > 2^{n-i-1}, \quad 0 \leqq i \leqq n-2.$$

Moreover, these are the best possible results.

Before we can prove this result, we need the following lemma.

Lemma 1. *Suppose the total memory 2^n is so divided that we get maximum number of weighted buddy blocks of size just large enough to hold R_m. This maximum*

number is given by

$$B_i=(1/3)\cdot(-1)^i+(1/3)\cdot 2^{i+1}$$

where i is an integer, $0\leqq i\leqq n-2$ *such that*

a) $2^{n-i}\geqq R_m>3.2^{n-i-2}$

or

b) $3.2^{n-i-2}\geqq R_m>2^{n-i-1}$.

Proof. a) Consider the case when for some i, $0\leqq i\leqq n-2$, we have $2^{n-i}\geqq R_m>3.2^{n-i-2}$. The weighted buddy system has maximum number of blocks just large enough to hold R_m when each block is of size 2^{n-i} or 2^{n-i+1}. If a block of size 2^{n-i} is further split we get two blocks of sizes 2^{n-i-2} and 3.2^{n-i-2} neither of which can hold R_m. If a block of size 2^{n-i+1} is split, we get only one block large enough to hold R_m. Hence division of a block of size 2^{n-i} or a block of size 2^{n-i+1} does not increase the number of blocks large enough to hold R_m.

Let x be the number of blocks of size 2^{n-i} and y be the number of blocks of size 2^{n-i+1}. So

$$B_i=x+y.$$

Let each block of size 2^{n-i+1} be split to get two blocks each of size 2^{n-i-1} and one block of size 2^{n-i}. So the maximum number of blocks large enough to hold request R of size, $2^{n-i-1}\leqq R<3.2^{n-i-3}$ is given by

$$B_{i+1}=2y+(x+y).$$

Now if we split a block of size 2^{n-i} we get two blocks each of size 2^{n-i-2} and one block of size 2^{n-i-1}. So maximum number of blocks large enough to hold request R of size $2^{n-i-2}\leqq R<3.2^{n-i-4}$ is given by

$$B_{i+2}=2(x+y)+(2y+(x+y)).$$

Combining the three equations we get the recurrence relation

$$B_{i+2}=B_{i+1}+2B_i \tag{A}$$

where

$B_0=$ maximum number of blocks large enough to hold R of size

$$2^n\geqq R>3.2^{n-2}$$

$=1$

and

$B_1=$ maximum number of blocks large enough to hold r of size

$$2^{n-1}\geqq R>3.2^{n-3}$$

$=1$.

To solve the recurrence relation (A), we define a generating function

$$G(z)=\sum_{i\geqslant 0} B_i\cdot z^i.$$

Multiplying both sides of (A) by z^{i+2} and summing over $i\geqq 0$, we get

$$\sum_{i\geqq 0} B_{i+2}\cdot z^{i+2}=\sum_{i\geqq 0} B_{i+1}\cdot z^{i+2}+2\sum_{i\geqq 0} B_i\cdot z^{i+2}$$

or

$$G(z)-B_0-B_1\cdot z=z\cdot(G(z)-B_0)+2\cdot z^2\cdot G(z).$$

Substitution of values of B_0 and B_1 yields

$$\begin{aligned}G(z)&=1/(1-z-2\cdot z^2)\\&=(1/3)\cdot\{(1+z)^{-1}+2\cdot(1-2z)^{-1}\}\\&=\sum_{i\geqq 0}(1/3)\cdot\{(-1)^i+2^{i+1}\}\cdot z^i.\end{aligned}$$

Or we have

$$B_i=(1/3)\cdot(-1)^i+(1/3)\cdot 2^{i+1}.$$

b) Consider the case when $3.2^{n-i-2}\geqq R_m>2^{n-i-1}$, $0\leqq i\leqq n-2$. Here, we have maximum number of blocks large enough to hold R_m when again all blocks are of sizes 2^{n-i} or 2^{n-i+1}. If we split a block of size 2^{n-i} we get only one block large enough to hold R_m. Also if we split a block of size 2^{n-i+1} we get only one block large enough to hold R_m. Rest of the proof is similar to that in case a).

Proof of Theorem 1a. Since R_m cannot be accomodated in a memory block it is clear that for each block of size 2^{n-i} or 2^{n-i+1} either a descendent or an ancestor must have been allocated. If a descendent of the block is allocated then the size of the corresponding request has to be at least one. If an ancestor A of a block of size 2^{n-i} or 2^{n-i+1} is allocated, the size of the corresponding request is such that we can allocate at least one word to each of the 2^{n-i} and 2^{n-i+1} descendents of A. To justify this claim we note that if a block of size 2^j is allocated the corresponding request must be of a size greater than or equal to $3.2^{j-2}+1$. Now by Lemma 1, the number of descendents of sizes 2^i and 2^{i+1} of a block of size 2^j is given by $(1/3)\cdot(-1)^{j-i}+(1/3)\cdot 2^{j-i+1}$. Since $3.2^{j-2}+1>(1/3)\cdot(-1)^{j-i}+(1/3)\cdot 2^{j-i+1}$ for $j>i$, we can allocate at least one word to each of these descendents. So,

$$\text{NETREQ}\geqq(1/3)\cdot(-1)^i+(1/3)\cdot 2^{i+1}+3.2^{n-i-2}+1.$$

The right hand side of this inequality is minimum for $i=\lceil n/2\rceil$ and hence

$$\text{NETREQ}\geqq(1/3)\cdot(-1)^{\lceil n/2\rceil}+(1/3)\cdot 2^{\lceil n/2\rceil+1}+3.2^{\lfloor n/2\rfloor-2}+1.$$

To see that this is the best possible result, let us consider a request sequence

consisting of two parts. In the first part allocation requests are made so that the entire memory is allocated in blocks of sizes 1 or 2. In the second part all of the storage is deallocated except for one block of size 1 in each memory block of size $2^{\lceil n/2 \rceil}$ or $2^{\lceil n/2 \rceil + 1}$. Then let the last request have a size $3.2^{\lceil n/2 \rceil - 2} + 1$. Combining these allocation and deallocation requests, the result follows.

Proof of Theorem 1b. The proof is similar to that of Theorem 1a. Here also since R_m cannot be accomodated in a memory block, it is clear from Lemma 1b) that for each block of size 2^{n-i} or 2^{n-i+1} either a descendent or an ancestor must have been allocated. By using the same argument as in the proof of part 1a) we have that at least one word of each of these blocks of size 2^{n-i} or 2^{n-i+1} is occupied. Since in this case minimum size of R_m is $2^{n-i-1}+1$, we can write

$$\text{NETREQ} \geqq (1/3)\cdot(-1)^i + (1/3)\cdot 2^{i+1} + 2^{n-i-1} + 1.$$

The right hand side of this inequality is minimum when $i = \lfloor n/2 \rfloor$ and hence

$$\text{NETREQ} \geqq (1/3)\cdot(-1)^{\lfloor n/2 \rfloor} + (1/3)\cdot 2^{\lfloor n/2 \rfloor + 1} + 2^{\lceil n/2 \rceil - 1} + 1.$$

To see that this is the best possible result, let us consider a request sequence consisting of two parts. In the first part allocation requests are made so that the entire memory is allocated in blocks of sizes 1 or 2. In the second part all of the storage is deallocated except for one block of size 1 in each memory block of size $2^{\lceil n/2 \rceil}$ or $2^{\lfloor n/2 \rfloor + 1}$. Then let the last request have a size $2^{\lceil n/2 \rceil - 1} + 1$. Combining these allocation and deallocation requests, the result follows.

Theorem 2. *For a weighted buddy system (memory size 2^n) and unrestricted request sequence*

a) $\text{NETALLOC} \geqq (1/3)\cdot(-1)^{\lceil n/2 \rceil} + (1/3)\cdot 2^{\lceil n/2 \rceil + 1} + 2^{\lfloor n/2 \rfloor}$, *if R_m is of such a size that*

$$2^{n-i} \geqq R_m > 3.2^{n-i-2}, \quad 0 \leqq i \leqq n-2.$$

b) $\text{NETALLOC} \geqq (1/3)\cdot(-1)^{\lceil n/2 \rceil} + (1/3)\cdot 2^{\lceil n/2 \rceil + 1} + 3.2^{\lceil n/2 \rceil - 2}$ *if R_m is of such a size that*

$$3.2^{n-i-2} \geqq R_m > 2^{n-i-1}, \quad 0 \leqq i \leqq n-2.$$

Proof a. In this case R_m needs a buddy block of size 2^{n-i} and does not get it. Following the proof of Theorem 1a, we get

$$\text{NETALLOC} \geqq (1/3)\cdot(-1)^i + (1/3)\cdot 2^{i+1} + 2^{n-i}.$$

Differentiation yields that the right hand side is minimum when $i = \lceil n/2 \rceil$ and hence

$$\text{NETALLOC} \geqq (1/3)\cdot(-1)^{\lceil n/2 \rceil} + (1/3)\cdot 2^{\lceil n/2 \rceil + 1} + 2^{\lfloor n/2 \rfloor}.$$

b) Similar to above (R_m needs a block of size 3.2^{n-i-2})

Note. We can show that this result is the best possible by giving similar arguments as in Theorem 1.

It may be recalled from [1] that for binary buddy system ($n \geqq 4$)

$$\text{NETREQ} \geqq 2^{\lfloor n/2 \rfloor} + 2^{\lceil n/2 \rceil - 1} + 1,$$
$$\text{NETALLOC} \geqq 2^{\lfloor n/2 \rfloor} + 2^{\lceil n/2 \rceil}.$$

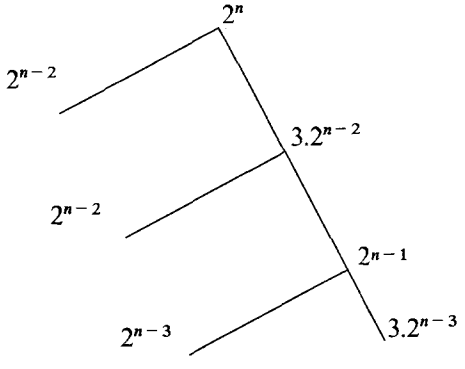
Hence, even in the worst case the internal fragmentation in weighted buddy system is less than that in binary buddy system but the external fragmentation in weighted buddy system is greater than that in binary buddy system.

In case of allocation only request sequence we have not been able to compute the exact lower bounds of NETREQ and NETALLOC. Instead we give the following two preliminary results.

Theorem 3. *For a weighted buddy system of total memory size 2^n, $n \geqq 3$ and a saturating allocation only request sequence with $R_m < 2^n$, external fragmentation exceeds one half of total memory size in the worst case.*

Proof. Let us consider a request sequence $R_1 = R_2 = 3.2^{n-3}$. From the following figure we find that R_1 can be accomodated and R_2 causes the saturation of memory. So external fragmentation

$$= 2^{n-2} + 2^{n-2} + 2^{n-3}$$
$$= 2^{n-1} + 2^{n-3}$$
$$> 2^{n-1}, \quad \text{for } n \geqq 3.$$



Theorem 4. *For a weighted buddy system of total memory size 2^n, $n \geqq 4$, and a saturating allocation only request sequence with $R_m < 2^n$, total fragmentation exceeds two thirds of total memory size in the worst case.*

Proof. Let us consider a request sequence $R_1 = R_2 = 2^{n-2} + 1$. From the previous figure we see that R_1 can be accomodated and R_2 causes a saturation of the memory. So total fragmentation (internal and external)

$$= 2^n - (2^{n-2} + 1)$$
$$= 3.2^{n-2} - 1 > (2/3) \cdot 2^n \quad \text{for } n \geqq 4.$$

For a saturating unrestricted request sequence comparing the worst case

values of NETREQ and NETALLOC for a binary buddy system [1] and a weighted buddy system we find that the worst case fragmentation of a weighted buddy system is slightly larger than that of a binary buddy system. This is because in the worst case a weighted buddy system has a larger external fragmentation compared to a binary buddy system. Also, for a weighted buddy system we note that even when a request smaller than the total memory size causes saturation in an allocation only request sequence case, as much as half of the memory will be lost due to external fragmentation and two thirds of the memory will be lost due to total fragmentation in the worst case.

4. Another Class of Memory Management Algorithms

Here we consider another well known general class of memory management algorithms where the size of allocated memory block is exactly equal to the size of the request. We call them exact fit algorithms. First-fit, best-fit etc. algorithms fall into this category. We assume as before the size of the total memory is 2^n.

Theorem 5. *For an exact-fit memory management system and a saturating unrestricted request sequence*

$$\text{NETREQ} \geqq 2\lfloor\sqrt{2^n}\rfloor.$$

Moreover, this is the best possible result.

Proof. Let $R_m = a$. We assume that the memory is divided into as many segments of size a as possible. Clearly, there are $2^n/a$ such segments. The rest of the memory consists of a single segment of size $2^n - a\lfloor 2^n/a\rfloor$ and this is smaller than a. Since R_m could not be satisfied, one or more words of each segments of size a must be allocated. So, the value of $R_1 + \ldots + R_{m-1}$ must be at least $\lfloor 2^n/a\rfloor$. Thus, the worst case values of NETREQ can be found by minimizing the expression $\lfloor 2^n/a\rfloor + a$. The minimum value of this expression is found to be $2\lfloor\sqrt{2^n}\rfloor$.

To see that this is the best possible result, let us consider a memory of size 2^n, for any given n. The request sequence consists of two parts. In the first part, allocation requests are made so that the entire memory is allocated in blocks of size 1. In the second part, all of the storage is deallocated, except for one block of size 1 in each memory block of size $\lfloor\sqrt{2^n}\rfloor$. The final request has size $\lfloor\sqrt{2^n}\rfloor$. Summing these allocation and deallocation requests the result follows.

Theorem 6. *For an exact-fit memory management system and a saturating unrestricted request sequence*

$$\text{NETALLOC} \geqq 2\lfloor\sqrt{2^n}\rfloor.$$

Moreover, this is the best possible result.

Proof. This theorem follows from Theorem 1 and the fact that the size of the allocated block is the same as the size of the requested block in exact-fit algorithms. The request sequence used in the proof of Theorem 1 also shows that this is the best possible result.

Theorem 7. *For an exact-fit memory management system and a saturating allocation only request sequence*

$$\text{NETREQ} \geqq 2^n + 1$$

Moreover, this is the best possible result.

Proof. We allocate contiguous memory blocks to $R_1, R_2, \ldots, R_{m-1}$ from one end of the memory to the other. Since $R_m = a$ causes saturation of the memory

$$R_1 + R_2 + \ldots + R_m \geqq 2^n - (a-1).$$

Therefore,

$$\text{NETREQ} \geqq 2^n + 1.$$

That this is the best possible result follows from the saturating request sequence $R_1 = 1$ and $R_2 = 2^n$. In this case $R_1 + R_2$ is $2^n + 1$.

Theorem 8. *For an exact-fit memory management system and a saturating allocation only request sequence*

$$\text{NETALLOC} \geqq 2^n + 1.$$

Moreover, this is the best possible result.

Proof. The result follows from the proof of Theorem 3 and the fact that the size of the allocated block is the same as the corresponding request size in exact-fit algorithms. The request sequence used in Theorem 3 shows that this is the best possible result.

Comparing the values of NETREQ and NETALLOC for an exact-fit algorithm with those for a binary buddy system (from [1]) we find that the values are almost the same. Hence, we conclude that in the worst case an exact-fit algorithm and a binary buddy system will cause almost the same fragmentation of the memory.

5. Conclusion

In this paper we have presented worst case values for two important system parameters NETREQ and NETALLOC for a weighted buddy system in case of unrestricted request sequence. In case of allocation only request sequence for a weighted buddy system it appears that the lower bound of NETREQ will be $2^{n-1} + 2$ and that for NETALLOC will be 3.2^{n-2}, but we have not been able to prove them [Note: In Theorem 3, $\text{NETALLOC} = R_1 + R_2 = 3.2^{n-3} + 3.2^{n-3} = 3.2^{n-2}$ and in Theorem 4, $\text{NETREQ} = R_1 + R_2 = 2(2^{n-2}$

$+1)=2^{n-1}+2]$. We have also derived bounds of NETREQ and NETALLOC for exact-fit memory management algorithms like best-fit or first-fit and to compare the results with those for buddy systems.

Acknowledgement. The authors are grateful for the detailed comments from two referees which greatly improved the presentation of the paper.

References

1. Lloyd, E.L., Loui, M.C.: On the worst case performance of buddy systems. Acta Inf. **22**, 451–473 (1985)
2. Bromley, A.G.: Memory fragmentation in buddy methods for dynamic storage allocation. Acta Inf. **14**, 107–118 (1980)
3. Knuth, D.E.: The Art of Computer Programming, Vol I. Reading, MA: Addison Wesley 1973
4. Peterson, D.L., Norman, T.A.: Buddy systems, Commun. ACM **20**, 421–431 (1977)
5. Shen, K.K., Peterson, J.L.: A weighted buddy system method for dynamic storage allocation. Commun. ACM **17**, 558–562 (1974)
6. Madnick, S.E., Donovan, J.J.: Operating Systems. New York: McGraw-Hill 1974

Received October 14, 1986 / May 18, 1987