Tertiary Buddy System for Efficient Dynamic Memory Allocation

DIVAKAR YADAV
AK SHARMA

Computer Science & Information Technology
Jaypee Institute of Information Technology University
A-10, Sector -62, Noida
INDIA
dsv99@rediffmail.com, ashokkale2@rediffmail.com

Abstract: - An extension of the binary buddy system, called “tertiary buddy system" for dynamic storage allocation is presented in this work. Tertiary buddy system allows block sizes of $2^k$ and $3 \cdot 2^{k-3}$ whereas the original binary buddy system allows only block sizes of $2^k$. This extension is achieved at an additional cost of two bits per block. Simulation of the proposed algorithm has been implemented in C programming language. The performance analysis in terms of internal fragmentation for the tertiary buddy system with other existing schemes such as binary buddy system, fibonacci buddy system and weighted buddy system is given in this work. Further the comparison of simulation results for number of splits and average number of merges for the above systems are also being discussed.

Key-Words: - Tertiary buddy system (TBS), binary buddy system, Fibonacci buddy system, weighted buddy system, internal fragmentation, external fragmentation, dynamic storage allocation and available space list (ASL).

1 Introduction

Dynamic memory allocation is used in arranging the memory effectively, which is one of the most critical components of any computer system. It is memory manager who performs the memory allocation process. Three dynamic storage allocation algorithms derived from the buddy systems have been proposed. Knowlton [7] and Knuth [8] described the original Binary buddy system. Under this memory management scheme blocks of sizes of powers of 2 are allocated. Inspired from Knuth’s suggestion [8], Hirschberg [10] has designed a Fibonacci Buddy system with block sizes which are Fibonacci numbers. Latter on Shen and Peterson designed weighted buddy system for memory allocation, which provides blocks whose sizes are $2^k$ and $3 \cdot 2^k$. In this work a new approach known as Tertiary Buddy System has been proposed, which allows blocks whose sizes are of the order of $2^k$ and $3 \cdot 2^{k-3}$. L. Beck has described another dynamic storage allocation scheme known as the release-match method in [5]. Purdom et al [9] have discussed the statistical properties of the buddy systems. A variation of buddy system for storing geometrical data has been discussed by W. Burton [6].

Two most important aspects of all the algorithms mentioned above are:

- Its execution speed, and
- Effectiveness in storage utilization.

According to K. Shen et al [2], the execution speed or the running time of the buddy system is computed by the number of blocks which are split and recombined whereas the effectiveness of these algorithms are measured on the basis of internal and external fragmentations created while allocating the memory using the above mentioned methods. Internal fragmentation is the result of allocating memory only in predefined block sizes. A request for a block of memory which is not one of these specified block sizes must be satisfied by allocating the next larger block size with a resulting loss in available memory. The memory wasted due to this over allocation is internal fragmentation [1]. The amount of internal fragmentation will vary depending upon the set of provided block sizes and distribution of requests for memory. Thus it may be used as point of comparison for buddy systems. External fragmentation is the result of breaking down memory into separate blocks which can not be recombined into a desired larger block. Thus a request for
memory may have to be rejected because no single contiguous block of large enough size is available, although the total amount of available memory, combining the smaller unallocated blocks may be sufficiently large to satisfy the request but scatted at different locations. This situation is illustrated in Fig.1.

Fig. 1: External fragmentation in the buddy system

Knuth through simulation has shown that external fragmentation is not a significant problem for the buddy systems rather it is internal fragmentation which is major problem for them [8]. Further he has shown than there may be a loss of between one fourth and one third of memory due to internal fragmentation.

Peterson et al [1] has discussed that the amount of internal and external fragmentation in a buddy system depends upon the distribution of requests for memory which must be satisfied and the block sizes provided. Further in his work he has discussed that for a particular distribution, one buddy system may have lower fragmentation than the other systems, while the situation may be reversed for another distribution. Since it is generally not easy to change the memory distribution to match the allocation strategy, it would be useful to have available a class of dynamic storage allocation algorithms. For a particular Problem, an algorithm could be selected from this class to minimize fragmentation and hence maximize memory utilization.

Nielsen et al [4] has investigated the performance of 35 dynamic memory allocation algorithms and has measured their performance in terms of processing time, memory usage and external fragmentation.

The paper is organized as follows: Section 2 discusses the buddy systems including proposed tertiary buddy system. Section 3 presents the simulation results and comparison of the proposed tertiary buddy system with other algorithms. Finally the work has been concluded in section 4 followed by references.

2 Buddy System

There are number of buddy systems, proposed by researchers at different time for dynamic memory allocation, which are capable of reducing execution time and increase memory utilization. The biggest disadvantages with these techniques are internal and external fragmentation. Let we describe few of the buddy systems before discussing about the proposed technique of tertiary buddy system.

2.1 Binary Buddy System

In binary buddy system the memory block of $2^m$ is divided into two equal parts of $2^{m-1}$. It satisfies the following recurrence relation [11].

$$L_i = L_{i-1} + L_{i-1}$$

For example if in beginning the available memory is of size $2^m$, a request of size $n$ is made and if $2^{m-1} < n < 2^m$ then allocate entire block of size $2^m$ else split this block into two buddies each of size $2^{m-1}$. If $2^{m-2} < n <= 2^{m-1}$ then allocate one of the two buddies other wise one of the two buddies is split in half again. This process is repeated until the smallest block greater or equal to $n$ is generated [1]. Fig.2 shows the tree structure of splitting memory block in a binary buddy system.

Fig.2: Tree structure of binary buddy system. Original memory size is $2^{10} = 1024$. 

NIelsen et al [4] has inv estigated the performance of 35 dynamic memory allocation algorithms and has measured their performance in terms of processing time, memory usage and external fragmentation.
2.2 Fibonacci Buddy System
Under this scheme, memory blocks are split in respect to Fibonacci number. It satisfies following recurrence relation:

\[ L_i = L_{i-1} + L_{i-2} \]

According to the definition, the elements of block sizes generated under this scheme are: 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 144, 233, 377, 610, 987, 1597, 2582...

Fig.3 shows the tree structure of splitting memory block in a Fibonacci buddy system.

Fig.3: Tree structure for Fibonacci buddy system. Original memory size is 610.

2.3 Weighted Buddy System
In weighted buddy system the memory block of size \(2^k\) is split in \(3.2^k\) and \(2^k\) sized blocks. Further \(3.2^k\) sized block is split in two \(2^{k+1}\) and \(2^k\) sized blocks. Diagrammatically the splitting of blocks of sizes \(2^{k+2}\) and \(3.2^k\) are as shown in Fig. 4a and Fig. 4b respectively. Fig. 5 shows the tree structure of weighted buddy system.

Fig 4: Splitting of blocks in weighted buddy system.

2.4 Tertiary Buddy System (TBS)
The tertiary buddy system allows blocks of sizes \(2^k\), \(0 \leq k \leq m\) and \(3.2^k\), \(0 \leq k \leq m-3\). This scheme allows blocks of sizes 4096, 2048, 1536, 1024, 768, 512, 384, 256 and so on.

In tertiary buddy system there are twice as many block sizes as are available in binary buddy system. It decreases the amount of internal fragmentation by allowing more block sizes. The reduction in internal fragmentation is achieved at a cost of two extra bits of “overhead” in each block. Hirschberg [10] has suggested that a class of algorithms could be defined to allocate block sizes which satisfy the following recurrence relation [1]:

\[ L_i = L_{i-1} + L_{i-k}, \quad k > 0 \]

(k = 1 is binary buddy system; k=2 is the Fibonacci buddy system). The weighted buddy system and tertiary buddy system do not satisfy the above recurrence relation. The weighted buddy system satisfies the following recurrence relation:

\[ L_i \geq L_{i+1} + L_{\beta(i)}, \quad i = 1, 2, \ldots, n, \quad L_0 = 0, \quad \beta(i) < i \]

Whereas for tertiary buddy system recurrence relation may be defined as follows:

\[ L_i = L_{i-1} + L_{i-3} + L_{\beta(i)} \]

Where \(\beta\) is any function over positive integers with \(\beta(i) < i\).

The tertiary buddy method is similar to the existing buddy methods except that in this method the blocks of the order of \(2^k\) are split into 3 different blocks to provide the desired smaller blocks and when the blocks are made free, they are combined with their buddy, if the buddy is available, if not, they are attached to the available space list (ASL). The calculation of the address of the buddy of a block, given the block’s address is also similar to other buddy methods. The differences lie in the method of allocating the blocks for a particular request from available space list (ASL). The differences also lie in the mechanism by which the blocks are split, recombined and the process of address calculation in tertiary buddy system.
2.4.1 Splitting Blocks in TBS

Let the total memory consists of $2^m$ words and it is addressed for convenience from 0 to $2^m - 1$. This memory forms the first block. In tertiary buddy system blocks of size $2^k$ is split into blocks of three different sizes. These blocks, in turn, are split into two different ways depending on the size of the block to be split. This is illustrated in Fig. 6a and Fig. 6b. Blocks of size $2^k$ are split into three blocks of sizes $2^{k-1}$, $3.2^{k-3}$ and $2^{k-3}$. Blocks of sizes $3.2^{k-3}$ are split into the blocks of sizes $2^{k-2}$ and $2^{k-3}$. Blocks split from the same parent are called buddies. The name tertiary buddy comes from the fact that in this a block of the order of $2^k$ is split into blocks of three different sizes. The largest sub block resulting from the split is the left most block called as the left buddy denoted by (L), the next larger sub block is the middle buddy (M) and the smallest of all the three is the right most block called as the right buddy (R). Example for splitting 64 sized blocks using tertiary buddy system is shown in fig. 7.

Fig. 5: Tree structure for weighted buddy system. Original memory size is $2^4 = 16$.

Fig. 6: Splitting of blocks in Tertiary Buddy System

Fig. 7: Tree structure for tertiary buddy system. Original memory size is $2^6 = 64$.

2.4.2 Available Space List

Available space lists are used to keep track of all available blocks of storage. The ASL serves as the head and tail of linked lists of available storage of sizes $2^m$, $3.2^{m-3}$, $2^{m-2}$, $2^{m-3}$, ……., 3, 2, 1. The tertiary buddy algorithm maintains two available space lists. ASL1 keeps track of block sizes of the order of $3.2^k$ and ASL2 keeps the record of block sizes of the order of $2^k$. Available blocks have two link fields, forward and backward links which link the blocks to the appropriate element of the ASL. Initially, there is only one block of size $2^m$ attached to the ASL at the top. As a request for block arrives this large block is split. The splitting scheme is as follows:

A block of the order of $3.2^k$, say for example the first element of ASL1 splits into the third and fourth element of ASL2. On the similar lines the first element of ASL2 splits into the second element of ASL2, first element of ASL1 and the third element of ASL2. New available blocks are attached to the end of the linked list and blocks desired for satisfying requests are removed from the front of the desired list i.e. ASL1 or ASL2 as in a queue.

When a requirement for a block of size $2^k$ is made and no blocks of such size are available then the following method is used to determine, how a larger block is to be repeatedly split until a block of size $2^k$ is generated.
Let a request for a block of size $2^k$ is made and it is unavailable then ASL is searched till the first available block of size greater than $2^k$ is found. After removing this block from ASL, it is split into two blocks. The splitting process continues for the smallest sub block which is greater than or equal to the requested size till the requested size is matched. The same process is followed if the request is for a block of size $3.2^k$.

3 Simulation Method and Results
A simulation of four buddy systems namely binary, Fibonacci, weighted and tertiary buddy systems was done to compare their performance in terms of internal and external fragmentation as well as the average number of splits and recombination. The wastage of memory for each block is computed as the difference between the allocated block and the size of request. By adding the wastage of memory for all allocated blocks, total internal fragmentation may be computed. Similarly memory overflow occurs when a request for memory cannot be satisfied due the reason that smaller blocks are available in scattered form i.e. no contiguous single block is available to satisfy the requested size of memory. When this overflow situation is generated then the ratio between the amounts of unallocated memory to the total memory size is known as the external fragmentation.

The basic simulation method used is described below:

A random generator was used to generate a sequence of numbers separately for uniform and exponential distribution. The numbers were generated in accordance with probability distribution function (PDF) for both uniform and exponential distribution.

The statistics for storage fragmentation for both the uniform and exponential block size distribution is shown as in Table-I. It was observed that the weighted and tertiary buddy systems are better than binary and Fibonacci buddy systems [3] for exponentially distributed block requests as there is significant decrement in internal fragmentation in weighted and tertiary buddy systems compared to binary and Fibonacci buddy systems. As for as the uniformly distribution of memory blocks for requests is concerned, the tertiary buddy method and weighted buddy method achieves a significant saving in internal fragmentation as compared to binary and Fibonacci buddy systems but at a greater expense of external fragmentation.

Table 1: Internal fragmentation

<table>
<thead>
<tr>
<th>Buddy systems</th>
<th>Exponential distribution</th>
<th>Uniform distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary buddy system</td>
<td>26%</td>
<td>25%</td>
</tr>
<tr>
<td>Fibonacci buddy system</td>
<td>21%</td>
<td>25.2%</td>
</tr>
<tr>
<td>Weighted buddy system</td>
<td>18.76%</td>
<td>14%</td>
</tr>
<tr>
<td>Tertiary buddy system</td>
<td>18.92%</td>
<td>22.8%</td>
</tr>
</tbody>
</table>

Further analysis was performed for computing average number of split and average number of merges for all the four memory management schemes (Binary, Fibonacci, Weighted and Tertiary buddy systems) as shown in Table-II. As the execution time of these algorithms are proportional to the number of searches, number of splits and number of merges required, the weighted buddy method takes maximum time for execution and same is reflected from the result tabulated in Table II. The figures shown in table also indicate that the binary buddy method and tertiary buddy method take less time for execution as compared to the weighted and Fibonacci buddy methods. This increase in execution time is for only the request and release mechanism, which probably take only a small portion for the overall computing task.

Table 2: Average number of splits and Average number of Merges

<table>
<thead>
<tr>
<th>Buddy system</th>
<th>Exponential distribution</th>
<th>Uniform distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. no. of splits</td>
<td>Avg. no. of merges</td>
</tr>
<tr>
<td>Binary buddy system</td>
<td>0.52</td>
<td>0.42</td>
</tr>
<tr>
<td>Fibonacci buddy system</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>Weighted buddy system</td>
<td>0.9</td>
<td>0.63</td>
</tr>
<tr>
<td>Tertiary buddy system</td>
<td>0.47</td>
<td>0.421</td>
</tr>
</tbody>
</table>
4. Conclusion
A number of properties for dynamic memory allocation schemes, based upon the buddy systems were considered in this paper. Using these algorithms, the fragmentation characteristics and execution time for binary buddy, Fibonacci buddy, weighted buddy and for the proposed one i.e. tertiary buddy system was analysed. The results obtained thereof indicate that as the internal fragmentation decreases, the external fragmentation increases. It was further observed that the total fragmentation remains relatively constant, with 25 to 40 percent of memory being unusable in both forms, internal and external fragmentation. It was also observed that the execution time for the request/release process is directly proportional to the number of splits and merges i.e. if the number of splits and merges increases then execution time also increases.

Some comparisons were made for the binary, Fibonacci, weighted and tertiary buddy systems. The internal fragmentation in weighted and tertiary buddy systems is less than that in binary and Fibonacci buddy system. However, the total fragmentation of weighted and tertiary buddy system is worse than binary and Fibonacci buddy system owing to the external fragmentation. Still the lower execution time of binary buddy and tertiary buddy system would, therefore, recommend them for general use, although the execution time of Fibonacci buddy system is not much greater. If the distribution of block sizes is small, the weighted and tertiary buddy method may be a better allocation algorithm.

Finally it may be concluded that appropriate buddy systems, for requests in dynamic memory allocation can be chosen based on the knowledge of actual request distribution. With the actual request distribution if deemed appropriate, a new buddy system can be designed such that it can replace the original buddy system to improve memory utilization and execution speed.

References: