Bilkent University

Computer Engineering

CS342

Project - 03

Mohammed S. Yaseen
21801331

22/04/2021

Introduction

In this project we are implementing a memory allocation algorithm; basically, having a large memory, how can we allocate portions of it to multiple different processes without causing any segmentation faults. In this experiment, I am observing the amount of fragmentation this algorithm causes based on the allocation requests' sizes.

How the Buddy algorithm works

The Buddy allocation technique uses a tree-like structure to organize the allocations and deallocations. When there is a request for a particular amount of memory, it keeps splitting the blocks until it reaches the smallest block whose size is equal to or larger than the size of the request and the bookkeeping bytes.

If the first block found is allocated, the algorithm moves to the next block and checks its availability. If the block is available but much larger than the request, it continues to divide it and repeats the steps from the beginning.

While freeing memory, the algorithm tries to find the freed block's buddy and checks the possibility of having them merged into one block. If the buddy is not allocated, it can be merged. This process is repeated until the buddy is not mergeable.

To find the buddy, some arithmetic calculations are performed. We first check if the current block is at the beginning or middle of its parent block (the block that existed before the split). If it is at the beginning, then the beginning of its buddy is at a distance equal to the summation of the block's current location and its size. If it is in the middle, we subtract instead of adding. The following calculation is used to determine whether the block is at the beginning or in the middle:

Let k be an integer number and the current address of the block be a multiple C of 2^k that is C2^k. This means that the beginning of its parent was located at some multiple of 2^(k+1); therefore, if 2^k mod 2^(k+1) equals 0, then the block is at the beginning of its parent, and its buddy is located in the middle. Otherwise, That the block is located in the middle of its parent, and its buddy is located at the beginning.

Comparing the amount of memory allocated and the amount asked

Here, the amount of memory that was asked by any process to **sbmemlib** and the amount of memory granted by the library are compared. The difference between the two is called internal fragmentation since it is located inside the memory block. Table 1 and figure 1 summarizes the findings.

Memory requested	Memory allocated	Fragmentation
250	512	262
350	512	162
450	512	62
550	1024	474
650	1024	374
750	1024	274
850	1024	174
950	1024	74
1050	2048	998
1150	2048	898

Table 1: Fragmentation amount based on request amount.

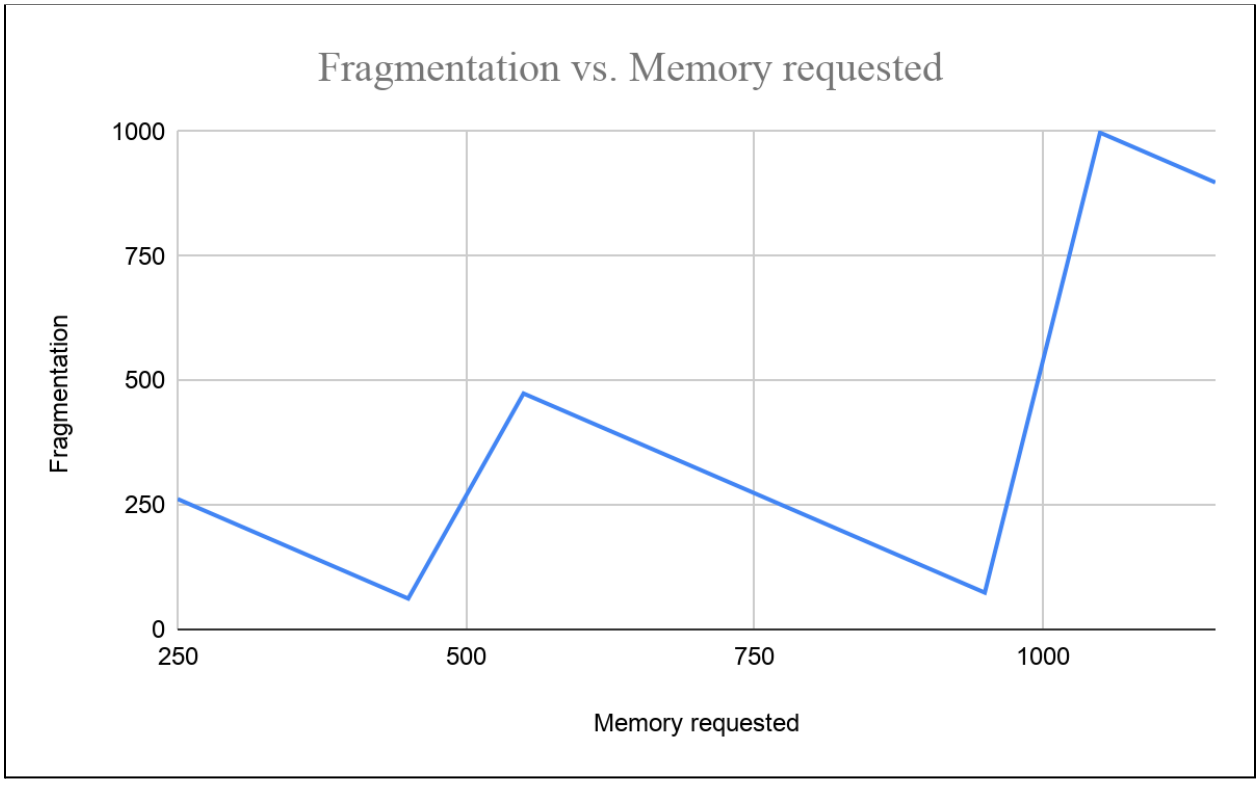

Figure 1: Fragmentation as a function of the amount of memory requested.

From the table and the figure, we can see that the fragmentation amount increases as the size requested gets farther from the next power of 2. For example, the request for 250 bytes is 262 bytes away from the next power of 2 that is 512. It is important to note that we cannot allocate 256 bytes since 8 bytes are being used for bookkeeping. These 8 bytes store the size of the block and whether or not it is allocated to a particular process. Hence, a request of 250 actually requires 258 bytes, which is 2 bytes larger than the previous power of 2 (256.)

Conclusion

The bubby algorithm is pretty efficient in allocating and freeing memory. In terms of time complexity, allocating requires O(n), while freeing requires O(logN) since it performs buddy merging. In terms of space complexity, its efficiency depends on the size of the request. If the requests are equal or close to (but smaller) than (2^k)(-8) the fragmentation amount would be too small and the algorithms would be very efficient. Otherwise, if the sizes of the requests are much smaller than (2^k)(-8) and a little larger than (2^k-1)(-8), then the fragmentation amount would be too high and the algorithms would be too inefficient.