# CS 342 – Operating Systems

## Project 3

21/04/2021

**Maryam Shahid**

21801344

Section 03

## 1. Introduction

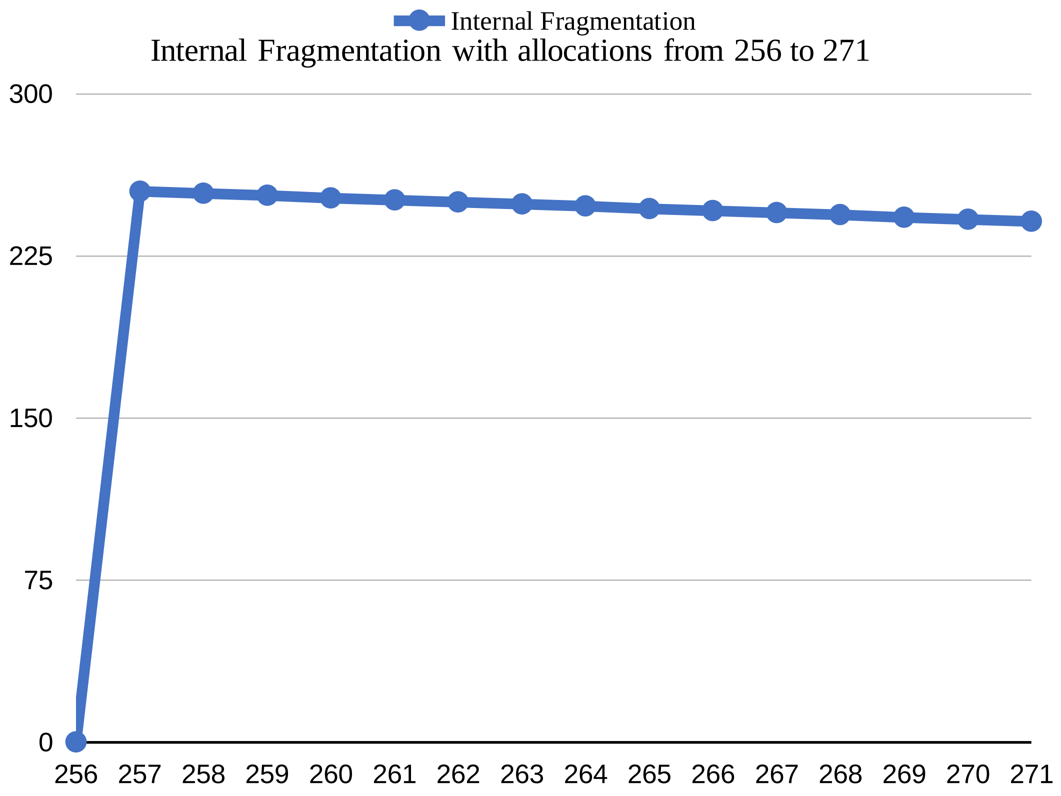
The project aims to implement a memory management library which dynamically allocates memory to processes. It consists of a shared memory segment which allows memory space allocation upon requests. It makes use of the buddy memory allocation algorithm to allocate and free spaces in the memory segment

## 2. Experiments and Results

The buddy algorithm assigns blocks of size 2^k of memory from the shared memory. In our case, the least memory block is 128 (2^7) and the maximum is either 32KB (2^15) or 256KB (2^18). So the block allocated is either equal to or greater than the size of allocation demanded by the process.Thus, internal fragmentation occurs as if a process demands the allocation of size 129, just 1 bye off the 128 mark, it is assigned the entire block of 256; an internal fragmentation of 127 occurs. Internal fragmentation is zero if the allocation request is of size 2^k (k >= 7 && <=8 or 15). We ran several tests for different values of allocations by the processes and obtained this.

1) We ran tests on size requested between 256 and 271. The results are as below:

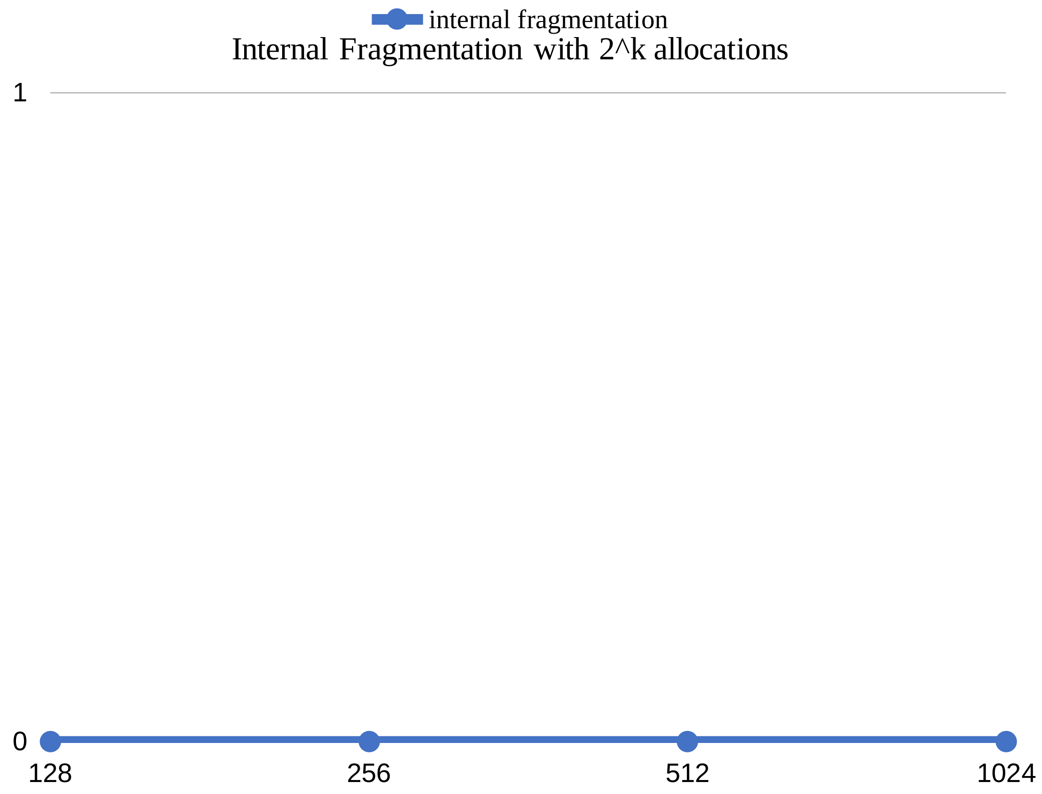
   The graph goes towards a steady decline, and as a gradient of minus one if we take the



allocation towards the size of 512. This is the perfect depiction of its worst case which is when the values are taken between two powers of two which is 256 and 512 in our case.

2) In the second experiment, we took values starting from 128, and then kept multiplying the value by two which gave us 256, 512, 1024, etc. The fragmentation is the case is zero:

This graph shows that internal fragmentation only occurs when the allocation request in not in size of 2^k. We can conclude that, buddy algorithm is pretty disadvantageous when using



allocations that aren’t of power 2. The above graph is the best form of allocation for buddy algorithm (best fit).

3) In the allocation of blocks with random values as input to the allocation, we observe a very random graph. Values that are equal to power of two have zero fragmentation and values that aren’t are scattered, as one can deduce from the above two plots.

Not only this, the buddy algorithm has another side effect of free space being available yet not being able to locate it due to chunks of memory being occupied. The buddy algorithm limits size allocation even if total size available is less than the size required by the process, however due to

occupied ‘buddies’ it becomes impossible to locate the collective free chunks to the request process. This is a very prominent dilemma of the buddy algorithm.
An example would be imagine a shared memory of size 1024 bytes. The lowest memory to be requested is 64 bytes:

**( {1024/free} )**

The requests made are 128, 256, 256, and 128.

**( {128/ occupied} {128/ free} ) ( {128/ occupied} {128/ occupied} ) | ( {128/ occupied} {128/ occupied} ) ( {128/ occupied} {128/ free} )**

The remaining memory after these allocations is 256, but one cannot allocate a space that required 129-256 or more. This is because the 128 byte free chunks aren’t buddies of one other and have their buddies occupied. Although, chunks of 128 or lesser can be allocated but not greater. This is also a major limitation of the buddy algorithm.

## 3. References

[1] http://homepage.cs.uiowa.edu/~jones/opsys/notes/27.shtml

[2] http://cs.boisestate.edu/~amit/teaching/453/handouts/memory-management-handout.pdf

[3] The Art of Programming. Donald Knuth.

[4] Operating System Concepts, 9th edition, Silberschatz et al. Wiley