Operating Systems Principles

ASSIGNMENT 1 REPORT

Labiba Islam

Student ID: s3694372

Contents

Page

1. Introduction

Memory management is the process in which the operating system controls and coordinates computer memory by allocating portions, known as blocks, to multiple running programs for optimising the overall system performance. *([1]Gibilisco, 2012)* In this report, based on the experiment, there are three different types of allocation strategies discussed, that have been used to find some meaningful statistics to conclude which allocation strategy is the best.

The different allocation strategies used for the experiment are:

1. First fit: In this algorithm, a free list (list of free blocks) is kept by the allocator. When the allocator receives a request, it traverses along the list to find the first block that is big enough to satisfy the request. If this block is notably larger than the one requested, it is split and the remaining of the block is added to the free list. *([2]Memorymanagement.org, 2018)*
2. Best fit: In case of best fit, the allocator, on receiving a request, traverses along the list to find the smallest block that will satisfy the request. In most cases, the left over will not be much, so there is no need to split.
3. Worst fit: In this algorithm, the allocator places, after receiving a request, will traverse along the list to find the largest block that will satisfy the request. This placement will make the biggest hold after the allocations. Therefore, it increases the probability of using the remaining block for another process, compared to best fit. *([3]Gokey, 2019)*

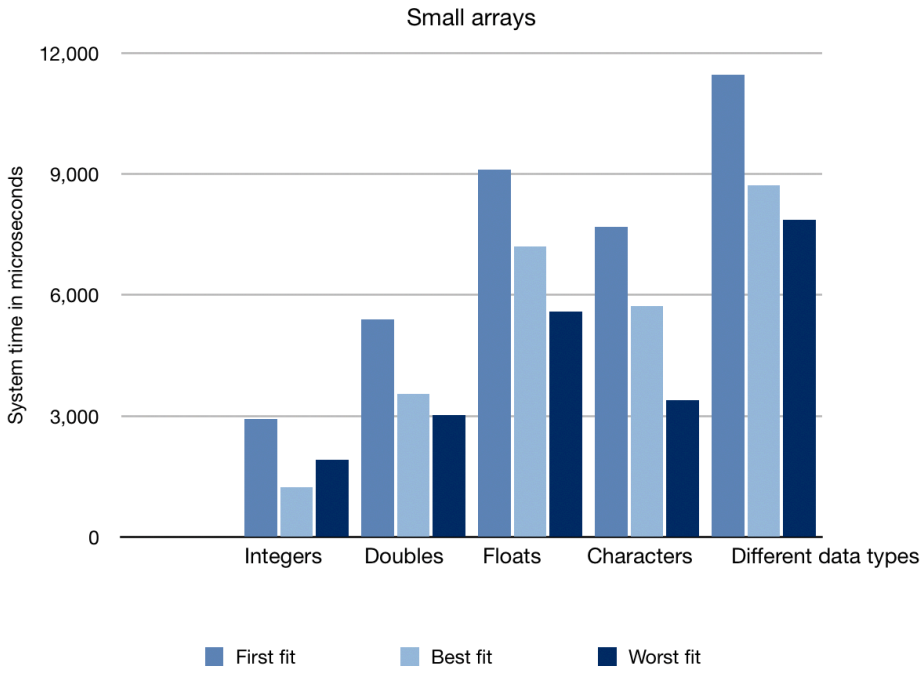
2. Experiment

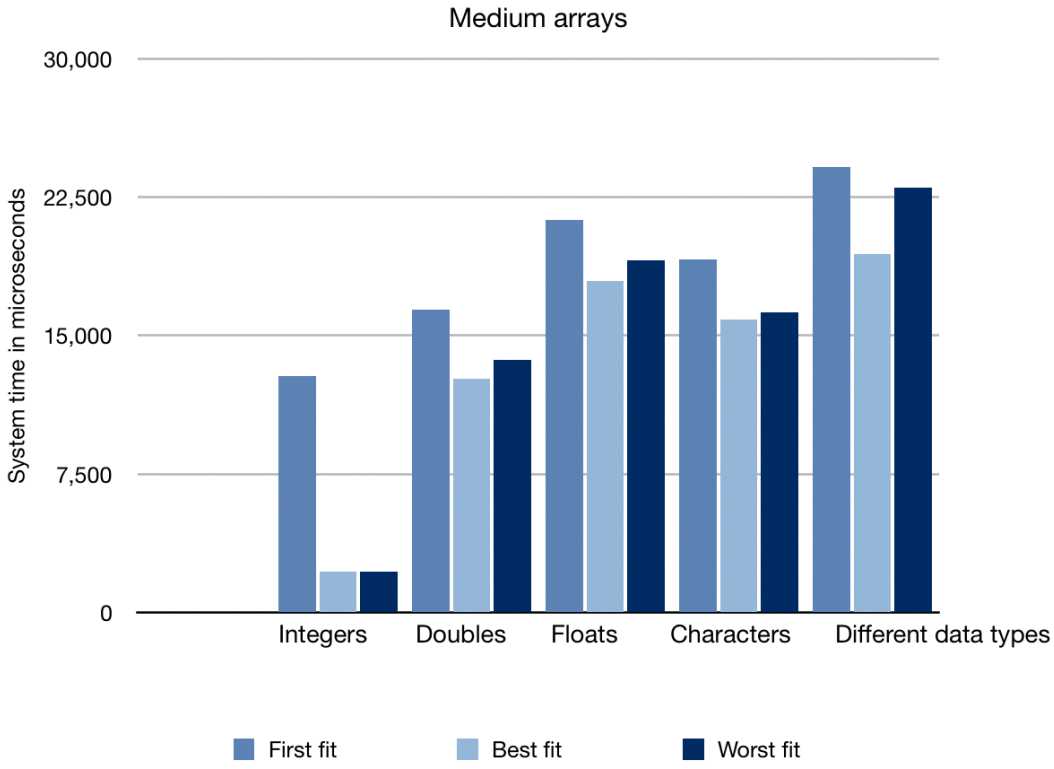
In my experiment, I have integer, double, float, and character arrays that are categorised into three sizes- small, medium and large. The small arrays are within the range of 20 to 50 indices long, while the size of medium ranges from 2000 to 5000 and the large ones from 200,000 to 500,000. My experiment is such that I allocate four integer arrays within the specified range of small arrays, then deallocate three and allocate one more. In this way, I am able to check if the different allocation strategies are working. Similar to the integer arrays, I have arrays of double, characters and floats as mentioned earlier. This will help me compare the time and memory usage for allocation and deallocation of identical array sizes for different data types. Additionally, I also have a case where I allocate and deallocate aforementioned identical sizes of arrays of different data types together to show the impact of allocation and deallocation different data types as opposed to a single type of data. Furthermore, since I have different categories of my array sizes, I can compare how with increasing array size, the allocation and deallocation of each data type changes. Therefore, in my

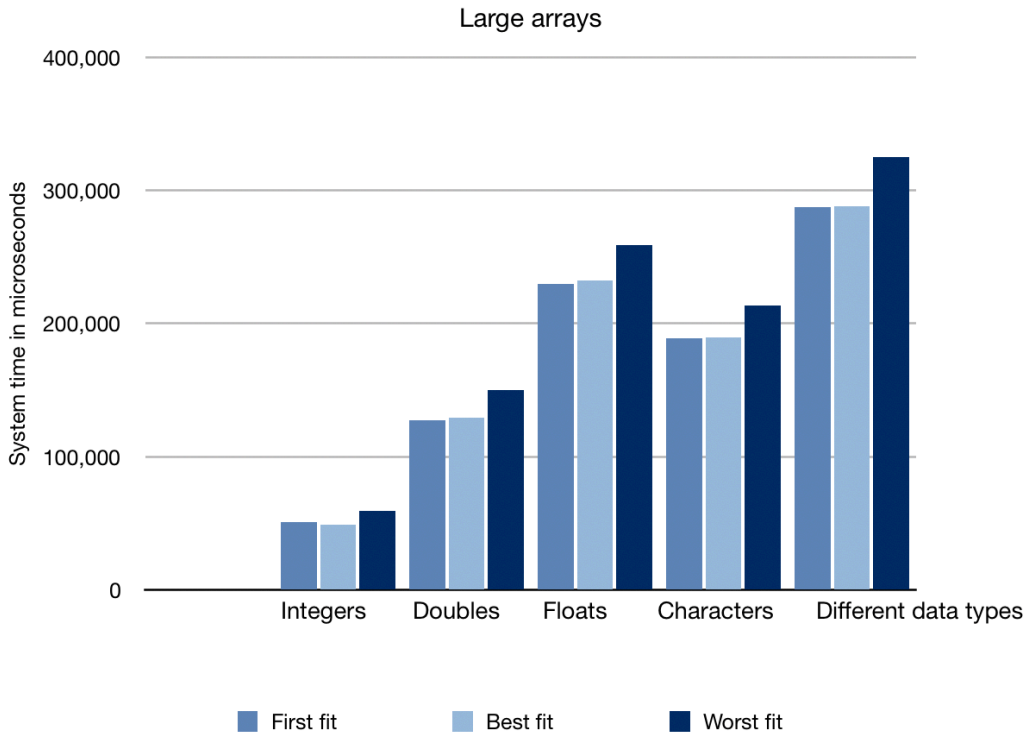
experiment, I can compare the time and memory usage for different data types and different sizes of the data types. I have coded for a loop to iterate 30 times and display me the average time and memory usage for all the cases.

2.1 Experimental data

The three charts below will show the **average** system time for small, medium and large arrays.

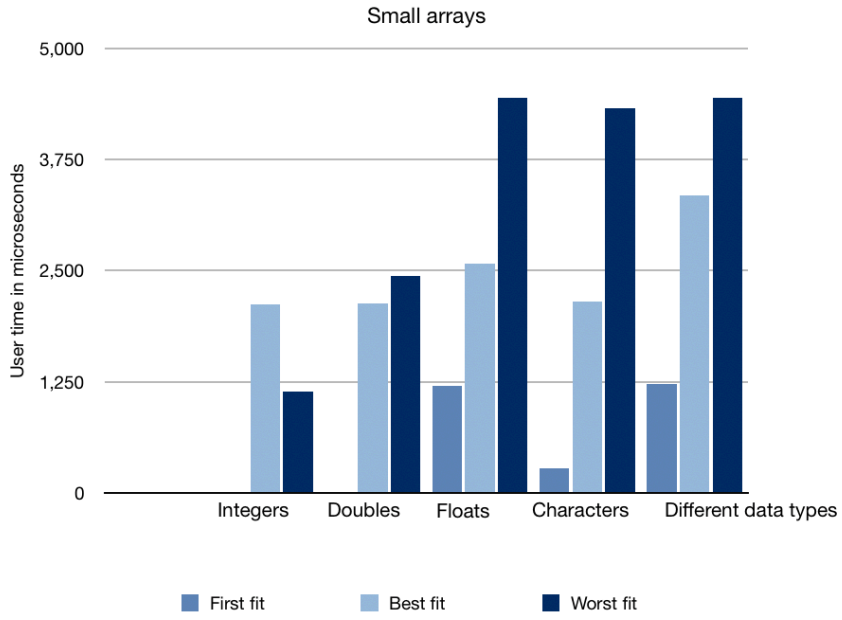


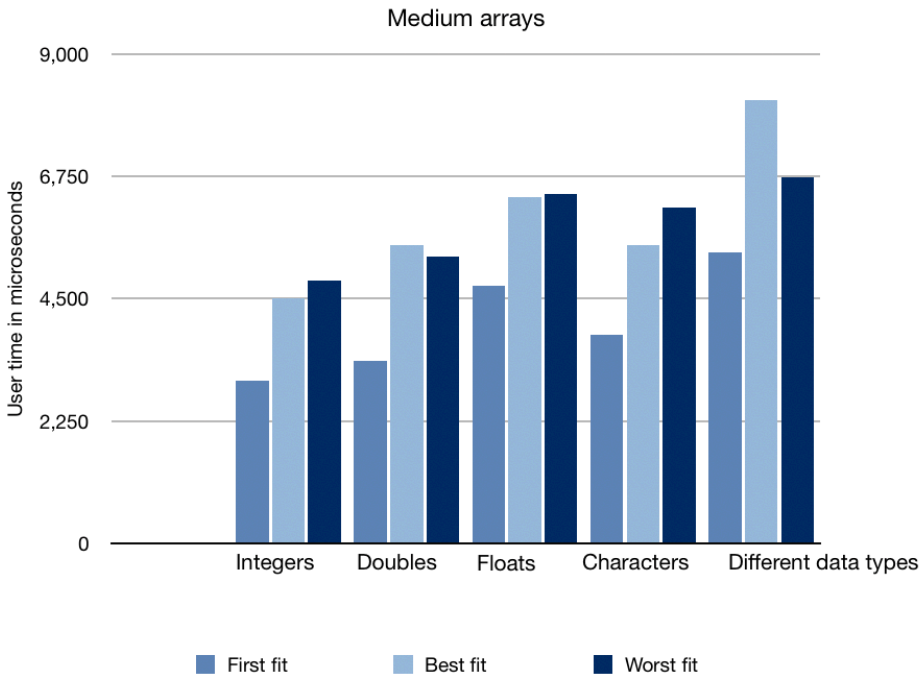

Medium arrays
30,000
22,500
15,000
7,500
0
System time in microseconds
Integers
Doubles
Floats
Characters
Different data types
First fit
Best fit
Worst fit

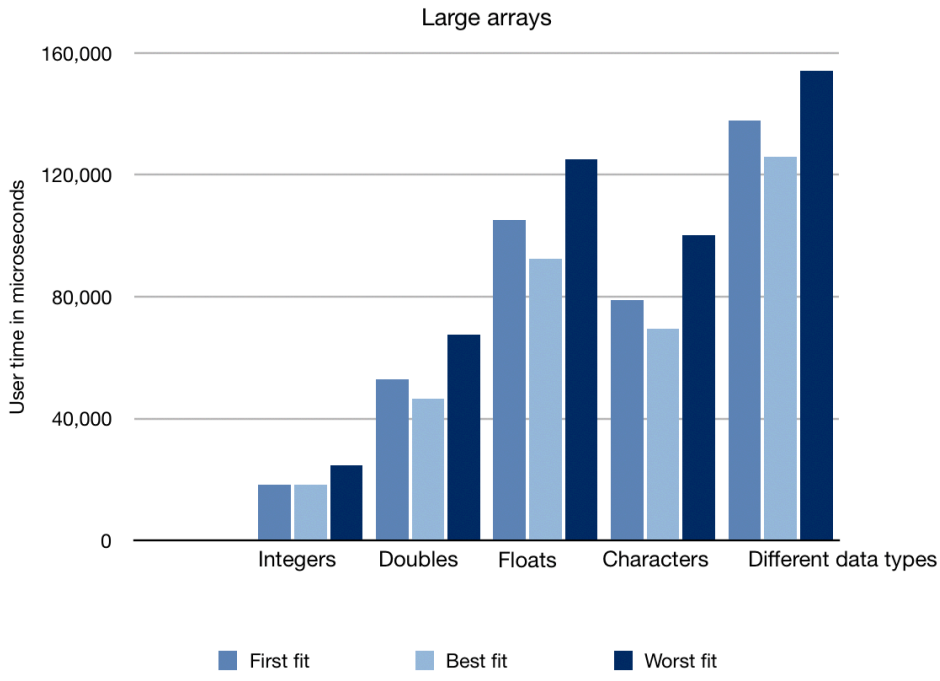

Large arrays
400,000
300,000
200,000
100,000
0
System time in microseconds
Integers
Doubles
Floats
Characters
Different data types
First fit
Best fit
Worst fit

From the above experimental data, best fit is a better allocation strategy in terms of system time when the array size is medium, while worst fit seems to be a better policy for all data types except integers when the array size is small. For large array sizes, system time for best and first fit are quite similar, while worst fit takes significantly longer.

The next three charts will depict the **average** user time for small, medium and large arrays.

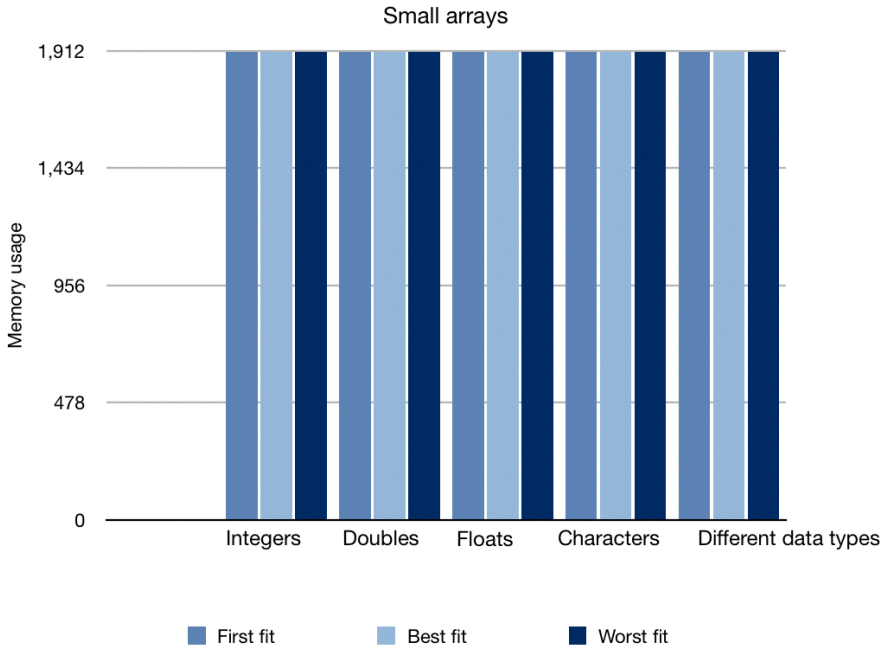


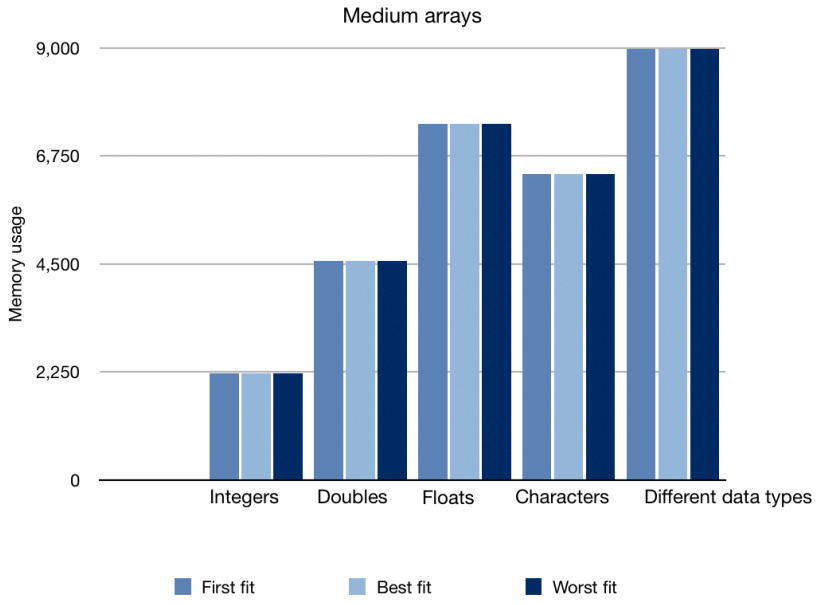

Medium arrays
9,000
6,750
4,500
2,250
0
User time in microseconds
Integers
Doubles
Floats
Characters
Different data types
First fit
Best fit
Worst fit

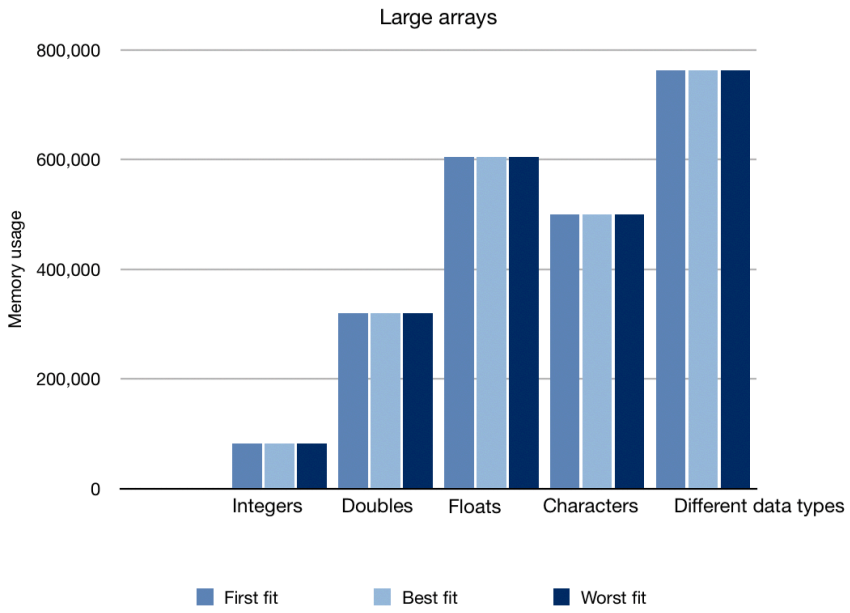

Large arrays
160,000
120,000
80,000
40,000
0
User time in microseconds
Integers
Doubles
Floats
Characters
Different data types
First fit
Best fit
Worst fit

In case of user time, first fit takes the least time for small and medium arrays, but for large arrays, first fit takes a bit longer than best fit. Worst fit takes the longest in most cases however, when different types of data are allocated and deallocated, best fit takes the longest when the array sizes are medium.

The following three bar charts will illustrate the **average** memory usage in kilobytes for small, medium and large arrays.







For my experiment, the memory consumption appears to be similar for all the allocation strategies, depending on their data type and data size. The small amount of difference that they have is not enough to show a significant difference in the bar chart. For small arrays, all data types have the same memory usage but for medium and large array sizes, integers have the least memory usage, followed by double, characters, floats and then all different types of data allocated and deallocated together.

3. Fragmentation

Processes are continuously loaded and unloaded from the main memory by a user. These processes are stored in the main memory in blocks. When we allocate the RAM to the processes dynamically, some free memory blocks remain available which are not enough to load the process on RAM. Such a condition is called fragmentation. There are two types of fragmentations: external and internal.

External fragmentation occurs when memory blocks of different sizes are allocated to the processes. Free space is created in the RAM during this type of fragmentation which cannot be reused since the size is too small. Internal

fragmentation occurs when fixed size memory is allocated to the processes. This happens when the memory requested by the process is smaller than the memory assigned to the process. *([4]T4Tutorials, n.d.)*

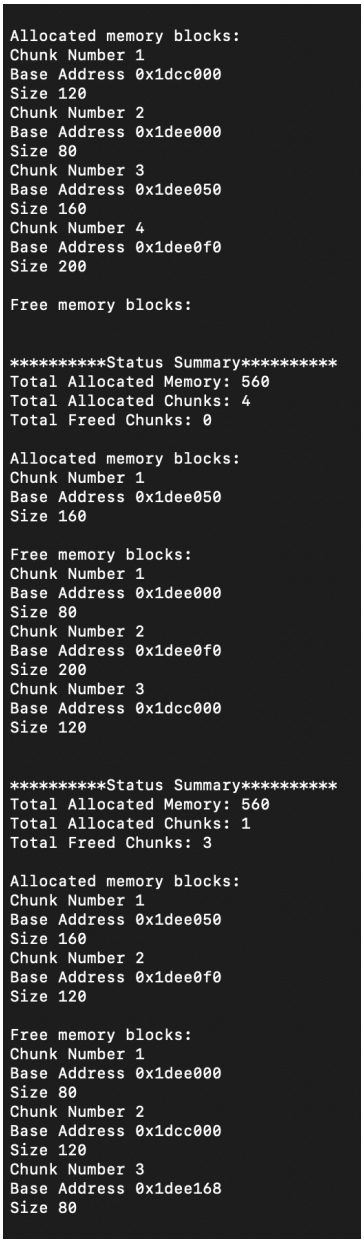
3.1 Solution to fragmentation

Compaction or shuffling of memory contents can be used to deal with external fragmentation. All the free memory chunks can be moved together in one large contiguous block. Thus, it can process the load on the RAM and reduce wastage. *([5]Stack Overflow, 2014)* Another way external fragmentation can be reduced is paging. "Paging is a memory management technique in which the process address space is broken into blocks of the same size called pages." The number of pages is a measure of the size of the process. The main memory is broken down into frames, small sized block of the physical memory, which are kept the same size of a page so that the main memory is utilised optimally. *([6]Tutorialspoint.com, n.d.)*

3.2 Fragmentation in experiment

The fragmentation in my experiment is external fragmentation. Evidence of fragmentation is given below. The fragmentation can be reduced by compaction or shuffling of memory contents as described in section 3.1.

For first fit, the fragmentation is shown in the image below:

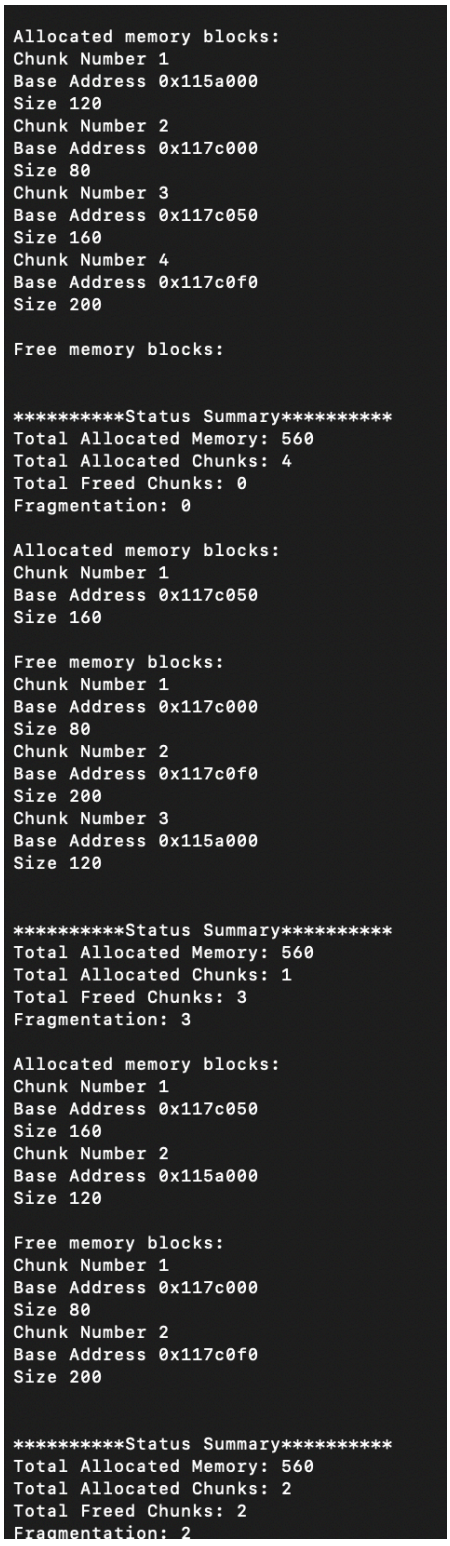


```
Allocated memory blocks:
Chunk Number 1
Base Address 0x1dcc000
Size 120
Chunk Number 2
Base Address 0x1dee000
Size 80
Chunk Number 3
Base Address 0x1dee050
Size 160
Chunk Number 4
Base Address 0x1dee0f0
Size 200

Free memory blocks:


**********Status Summary**********
Total Allocated Memory: 560
Total Allocated Chunks: 4
Total Freed Chunks: 0

Allocated memory blocks:
Chunk Number 1
Base Address 0x1dee050
Size 160

Free memory blocks:
Chunk Number 1
Base Address 0x1dee000
Size 80
Chunk Number 2
Base Address 0x1dee0f0
Size 200
Chunk Number 3
Base Address 0x1dcc000
Size 120


**********Status Summary**********
Total Allocated Memory: 560
Total Allocated Chunks: 1
Total Freed Chunks: 3

Allocated memory blocks:
Chunk Number 1
Base Address 0x1dee050
Size 160
Chunk Number 2
Base Address 0x1dee0f0
Size 120

Free memory blocks:
Chunk Number 1
Base Address 0x1dee000
Size 80
Chunk Number 2
Base Address 0x1dcc000
Size 120
Chunk Number 3
Base Address 0x1dee168
Size 80
```

After a second memory of 120 bytes is allocated, the 200-byte chunk is split to 80 and 120 under the list of free memory blocks. The chunk with 200 is selected because the 200-byte chunk is the first chunk available which is large enough to fit 120 bytes. So, in the end, I am left with three chunks in my free memory list.

Average memory chunk is = (80+120+80)/3 = 93.3 bytes

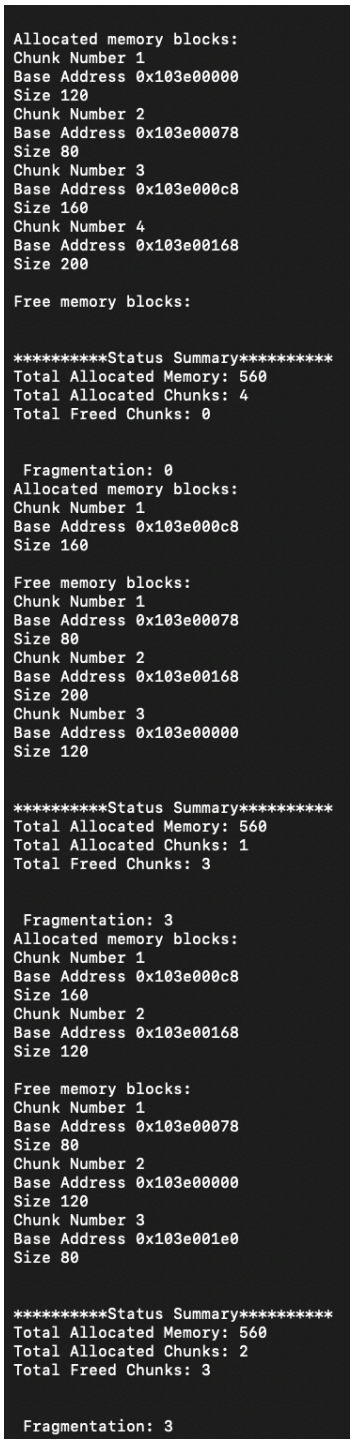
For best fit, the fragmentation is shown below:



After the second allocation of 120 bytes, since a 120-byte chunk is available, that chunk is selected because it is just enough to fit 120 bytes. So, in the end, there is only two chunks in my free memory list. Fragmentation for best fit is less than that of first and worst fit.

Average memory chunk = (80+200)/2 = 140 bytes. This is larger than first and worst fit.

For worst fit, fragmentation is shown below:



Here, after the second allocation of 120 bytes, the 200 bytes chunk in the free list is split into 120 bytes and 8- bytes. The 200-byte chunk is selected because it is the largest chunk available to fit the 120-byte chunk. Therefore, at the end, there are three chunks in my free memory list.

Average memory chunk = (80+120+80)/3 = 93.3 bytes

4. Wastage

All the three allocation strategies cause external fragmentation. Best fit tends to create many useless tiny fragments that might be not large enough to fit any other chunk. However, from the experimental data above, first fit and worst fit creates more wastages while best fit causes the least.

Wastage can be reduced by block coalescing, which has a similar concept as compaction, that is, joining neighbouring free blocks to make a large one. This way, fragmentation is reduced, and this also increases the reuse of memory as coalesced blocks can fit bigger future allocations. *([7]Heikkilä, 2012)*

5. Conclusion

From my experimental data, the statistics show that different strategies are better for different sizes of data and different types of data. However, from my findings, best fit is the best allocation strategy because it never takes the longest system or user time with different sizes and different types of data, as opposed to first fit and worst fit. Best fit also creates the least external fragmentation and wastage, thus making it the best allocation strategy in my opinion.

6. References

1. Gibilisco, S. (2012). *What is memory management? - Definition from WhatIs.com*. [online] WhatIs.com. Available at: https://whatis.techtarget.com/definition/memory-management [Accessed 5 Sep. 2019].

2. Memorymanagement.org. (2018). *2. Allocation techniques — Memory Management Reference 4.0 documentation*. [online] Available at: https://www.memorymanagement.org/mmref/alloc.html [Accessed 30 Aug. 2019].

3. Gokey, C. (2019). *Online CS Modules: Memory Allocation*. [online] Courses.cs.vt.edu. Available at: https://courses.cs.vt.edu/csonline/OS/Lessons/MemoryAllocation/index.html [Accessed 30 Aug. 2019].

4. T4Tutorials. (n.d.). *Fragmentation, External Fragmentation, Internal Fragmentation, first fit, best fit, worst fit in operating systems (OS) | T4Tutorials*. [online] Available at: https://t4tutorials.com/fragmentation-external-fragementation-internal-fragmentation-in-operating-systems-os/ [Accessed 4 Sep. 2019].

5. Stack Overflow. (2014). *How does external fragmentation happen?*. [online] Available at: https://stackoverflow.com/questions/13658791/how-does-external-fragmentation-happen/38560736#38560736 [Accessed 4 Sep. 2019].

6. Tutorialspoint.com. (n.d.). *Operating System - Memory Management - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/operating_system/os_memory_management.htm [Accessed 4 Sep. 2019].

7. Heikkilä, V. (2012). *https://pdfs.semanticscholar.org/90de/1d1e92d2c2966e87730c12d0c754990cab82.pdf*. [ebook] p.19. Available at: https://pdfs.semanticscholar.org/90de/1d1e92d2c2966e87730c12d0c754990cab82.pdf [Accessed 5 Sep. 2019].

7. Appendix

After the submission of my code, I created loops around my experimental code to run the code 30 times. I have also coded to store all the experimental data, that is, the memory usage, system time and user time in an array and then calculate and display the average data.