Operating Systems Principles

ASSIGNMENT 2 REPORT

Labiba Islam

Student ID: s3694372

Contents Page

1. Introduction

Multitasking allows the computer to run two or more programs simultaneously. There are two forms of multitasking- process-based and thread-based. Process-based multitasking is when multiple programs are run concurrently whereas thread-based multitasking is when pieces of the same program runs concurrently. A multithreaded program has two or more programs that run concurrently, and each part of the program is called a thread. “Each thread defines a separate path of execution”. *(1, Tutorialspoint.com, 2019).* Thread-safe code ensures that all of the threads are working properly without unintended interaction. *(2, En.wikipedia.org, 2019)*

2. Solution design

For my assignment, I used the reader-writer problem. The reader-writer problem is used to assist synchronisation so that there are no issues with the object data. There can be multiple readers at the same time, however, there cannot be more than one writer at the same time. To solve this, the writer needs an exclusive access to an object. If a process has locked a list for writing, only that process can either read or write the list since it has exclusive access to that list. However, if a process locks a list for reading purpose, then it is a shared lock, i.e. multiple processes can read the list at the same time but none of those can write it.

The code for putting and releasing the locks is added in the same file since the file contains the operations to be performed on the data structures for maintaining allocated and freed list. So, before starting an operation, the required lock is requested and after completing the operation, the lock is released, making it available for other processors to use. The statements to put locks and release the locks are modular in nature, thus could be called directly from the current file without increasing the length of code too much.

In my assignment, for maximum concurrency, I have two read locks for allocated list and free list so that multiple processes can use them simultaneously. For example, if there is one reader reading the free list, another reader can read the allocated list. One lock will lock both the lists together and decrease concurrency. My code is deadlock free because none of the processes wait to acquire lock which is acquired by some other process while holding lock on some of the code which might be required by the other process. *(3, Tutorialspoint.com, 2019)* I have made sure that my code doesn’t allow two processes to select the same chunk as best suited chunk for the memory allocation. This is done by putting exclusive locks even while reading from the free list to choose the best chunk possible according to the allocation strategies. There are no race conditions in my code since my program never gives unexpected outcomes for similar inputs.

3. Experiment

3.1 Experimental setup

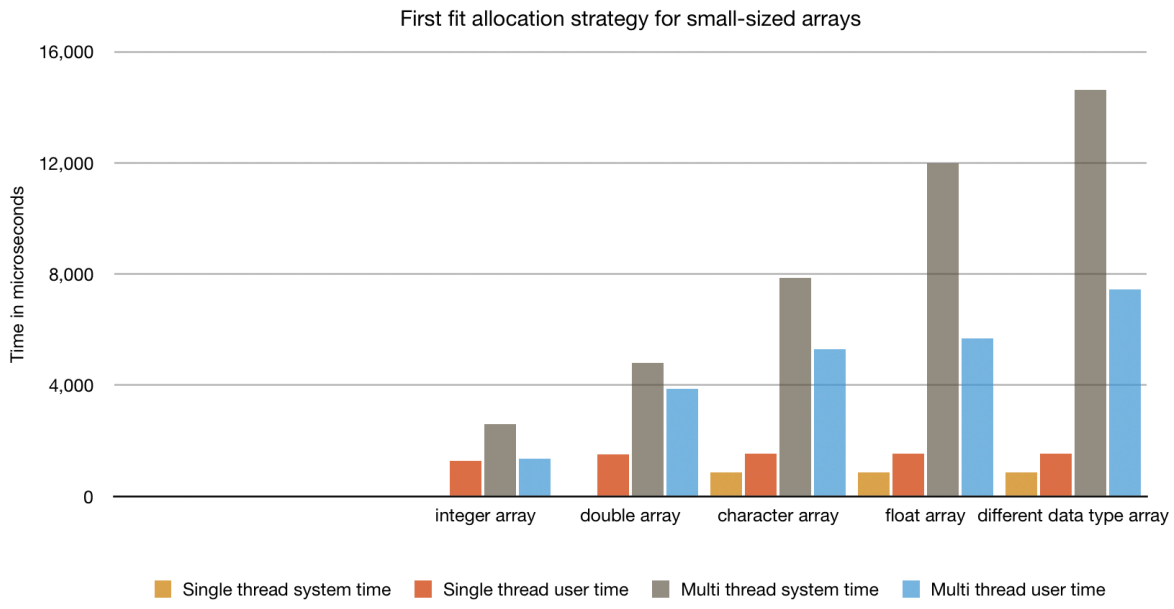
In my experiment, I have integer, double, float, and character arrays that are categorised into three sizes- small, medium and large. The small arrays are within the range of 20 to 50 indices long, while the size of medium ranges from 2000 to 5000 and the large ones from 200,000 to 500,000. My experiment is such that I allocate four integer arrays within the specified range of small arrays, then deallocate three and allocate one more. Similar to the integer arrays, I have arrays of double, characters and floats as mentioned earlier. This will help me compare the system and user time 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.

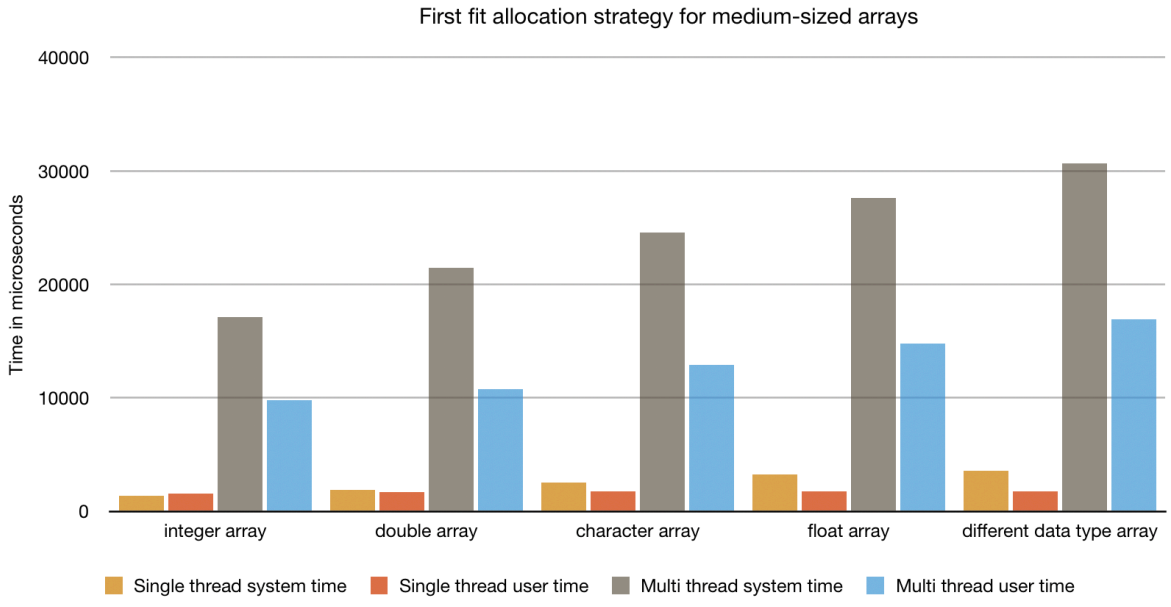
This is the same experiment as assignment 1. I have only created threads for assignment 2. I have loop to iterate the experimental data. First experiment is with 20 iterations and second experiment is with 60 iterations. My experiment will compare the system and user time for single threaded and multithreaded program, i.e. assignment 1 and assignment 2.

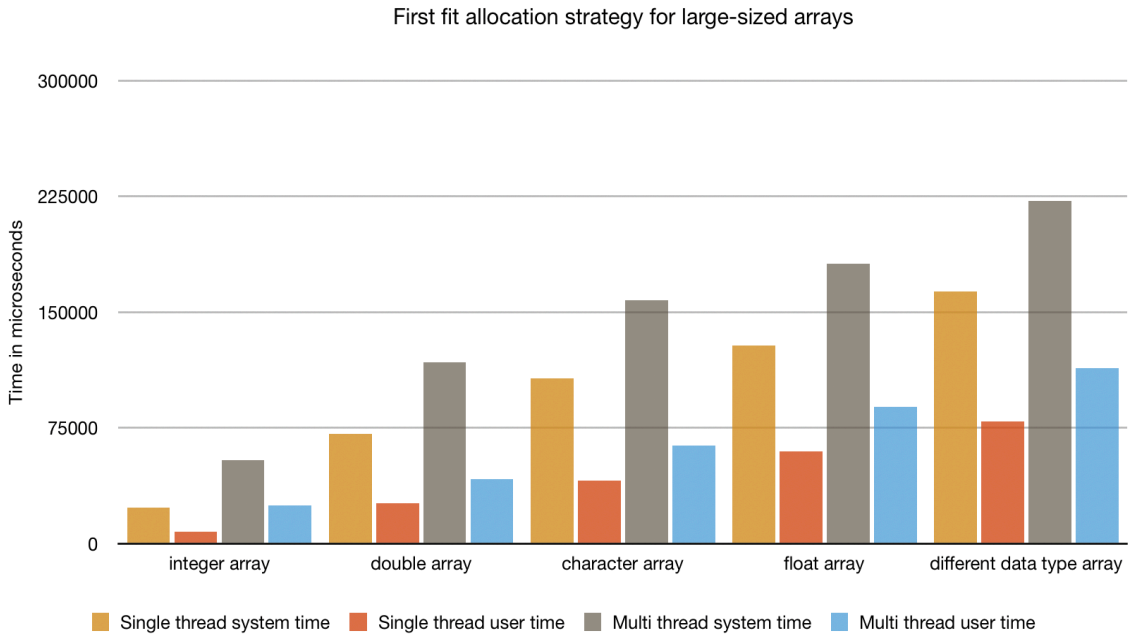
3.2 Experimental data

When number of iterations is 20:

The following three graphs show allocation and deallocation for first fit allocation strategy.

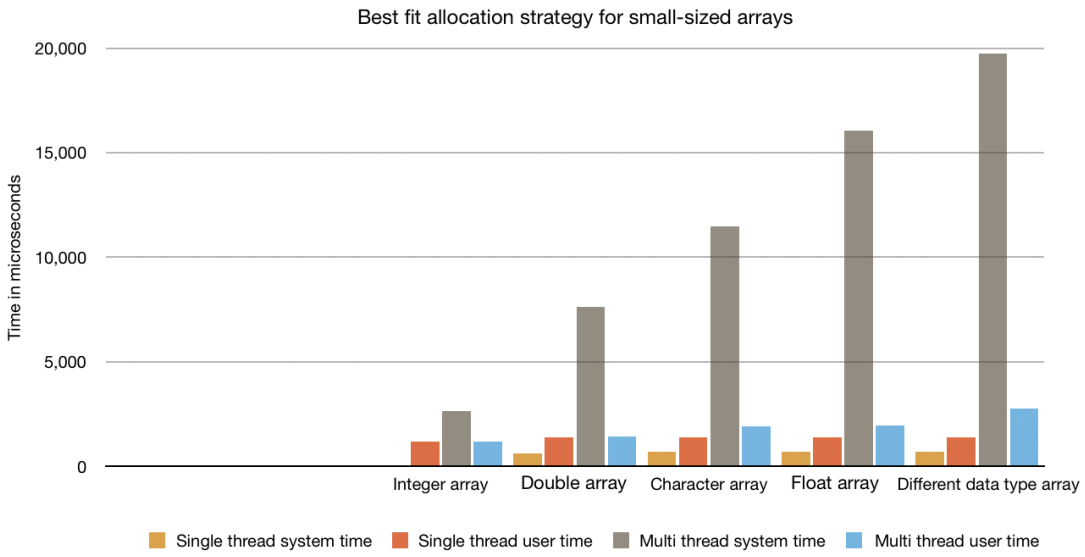


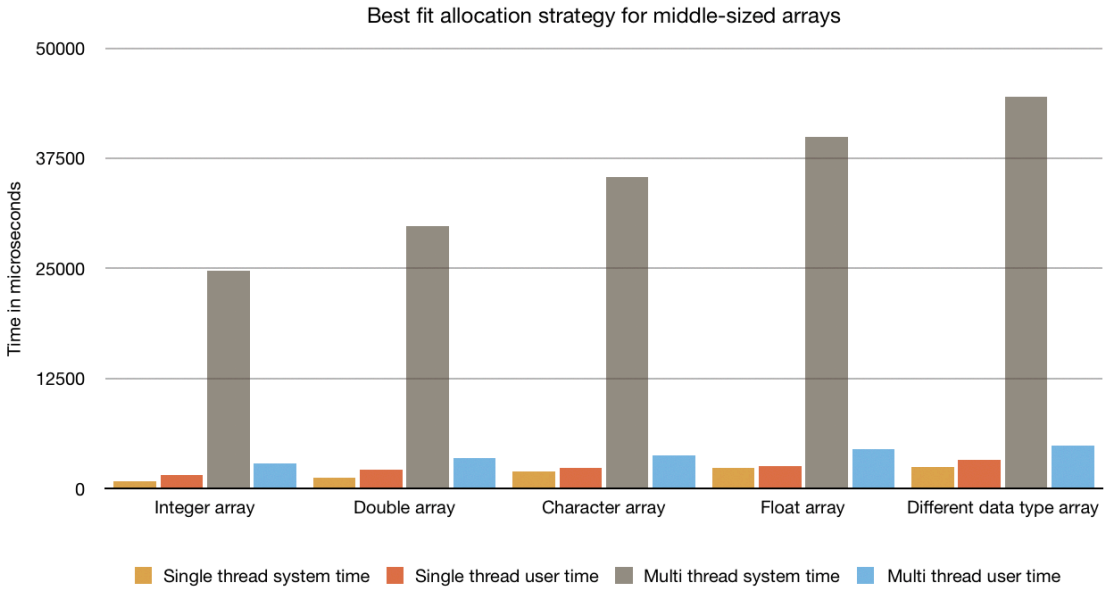


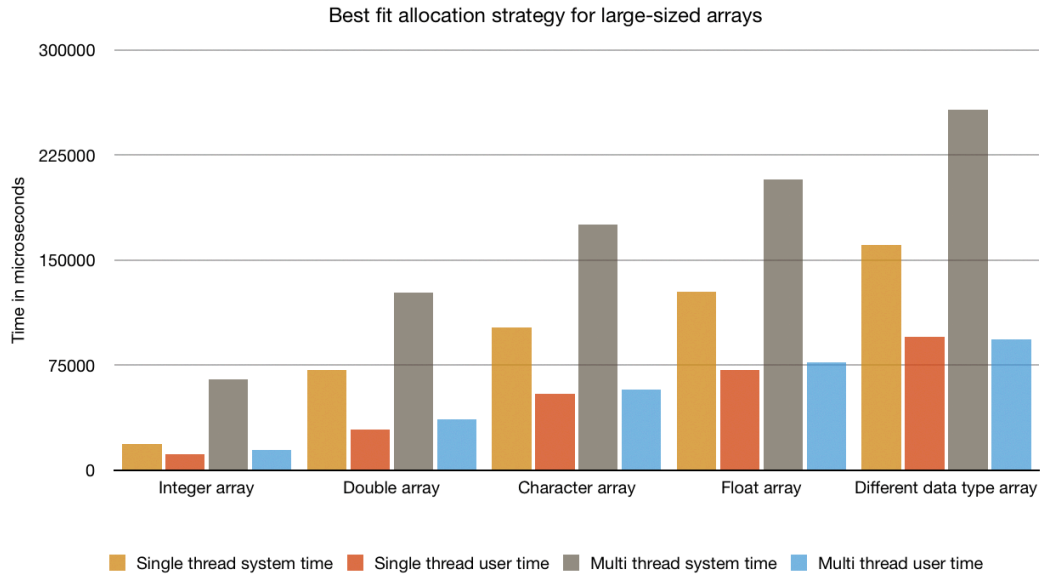


From the above three graphs, it can be seen that for small and medium-sized arrays, the multi- threaded program in assignment 2 is much slower than single threaded program in assignment 1. In case of large arrays, multi thread is still slower, but the difference is less than that of small and medium-sized arrays.

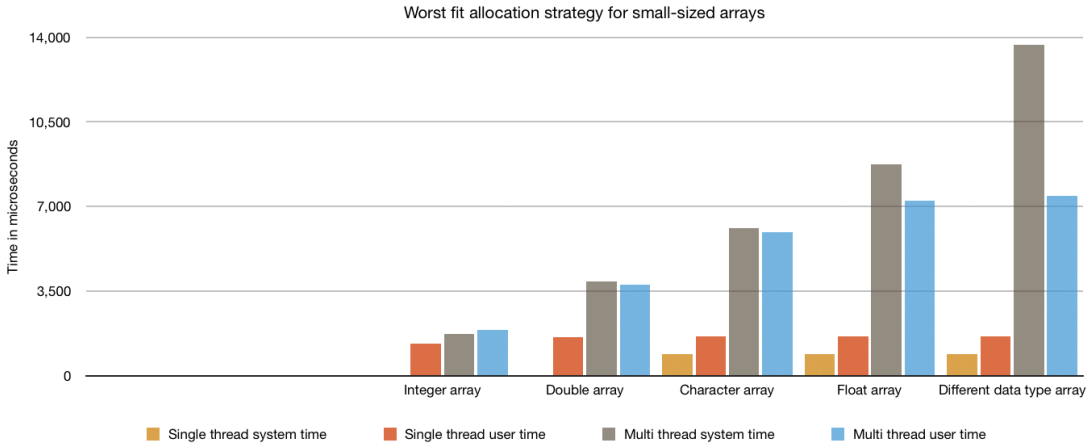
The next three graphs will depict the allocation and deallocation time for best fit allocation strategy.

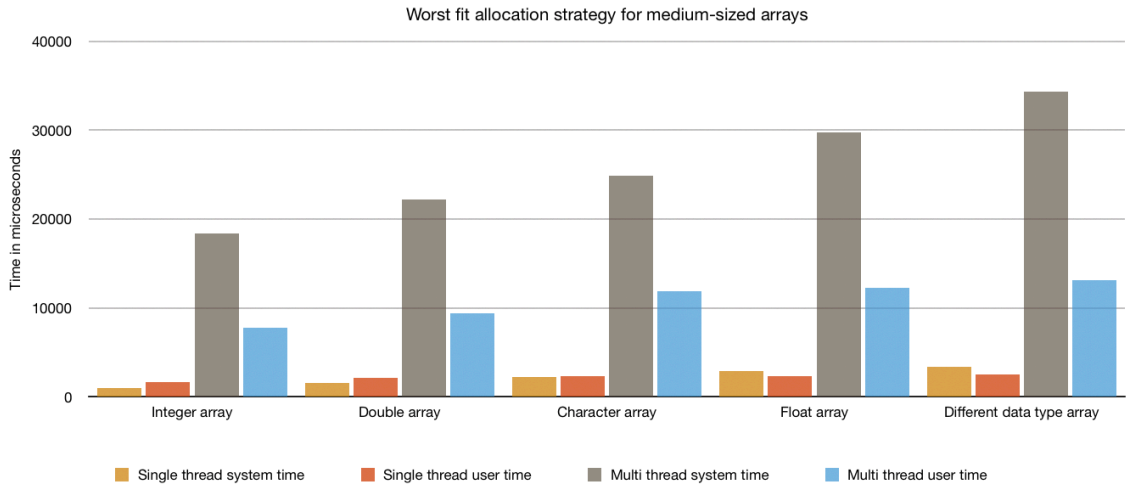

Best fit allocation strategy for small-sized arrays
Time in microseconds
20,000
15,000
10,000
5,000
0
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time

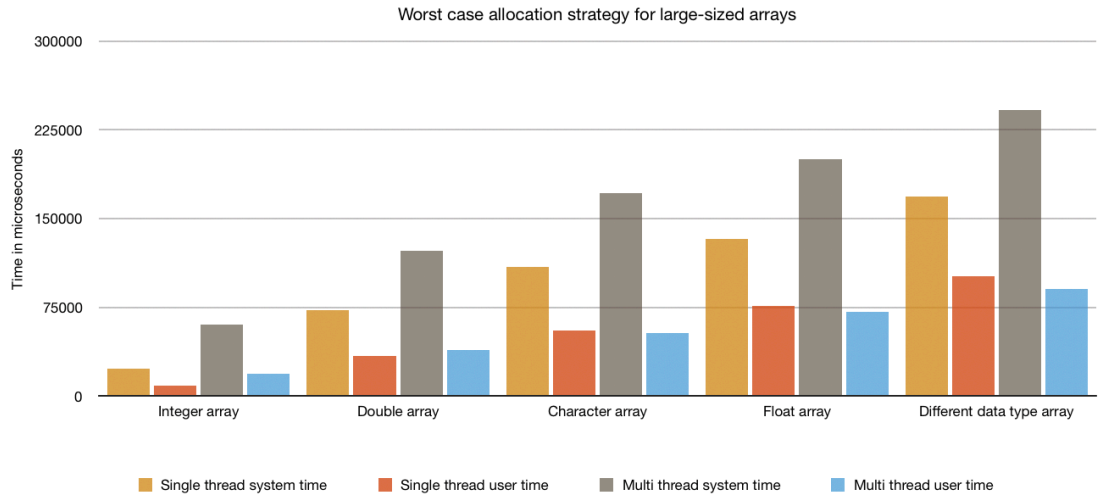

Best fit allocation strategy for middle-sized arrays
Time in microseconds
50000
37500
25000
12500
0
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time



The next three bar charts will show the allocation and deallocation time for worst fit allocation strategy.

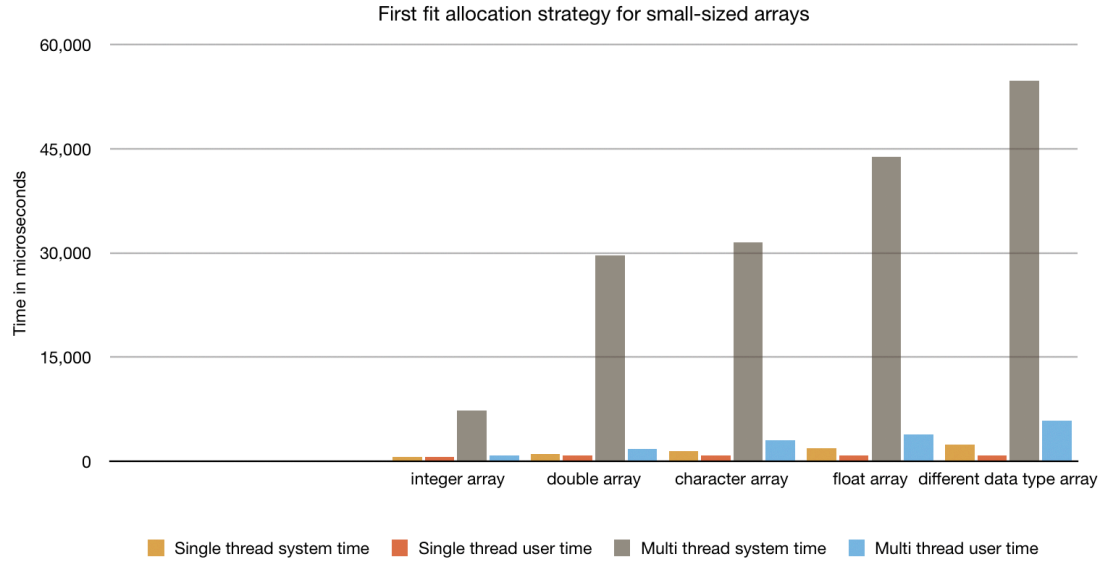


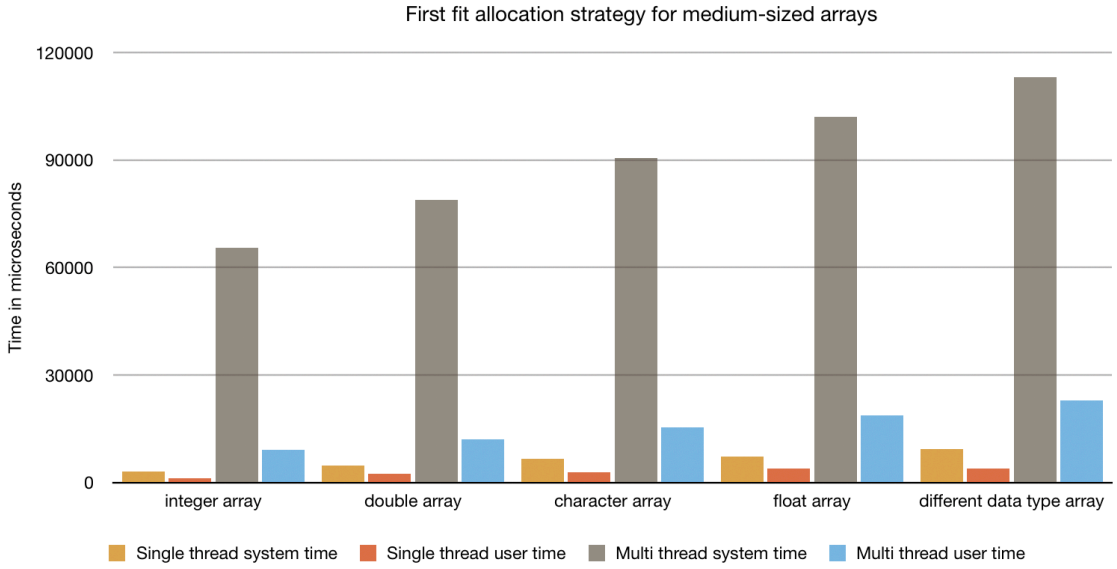


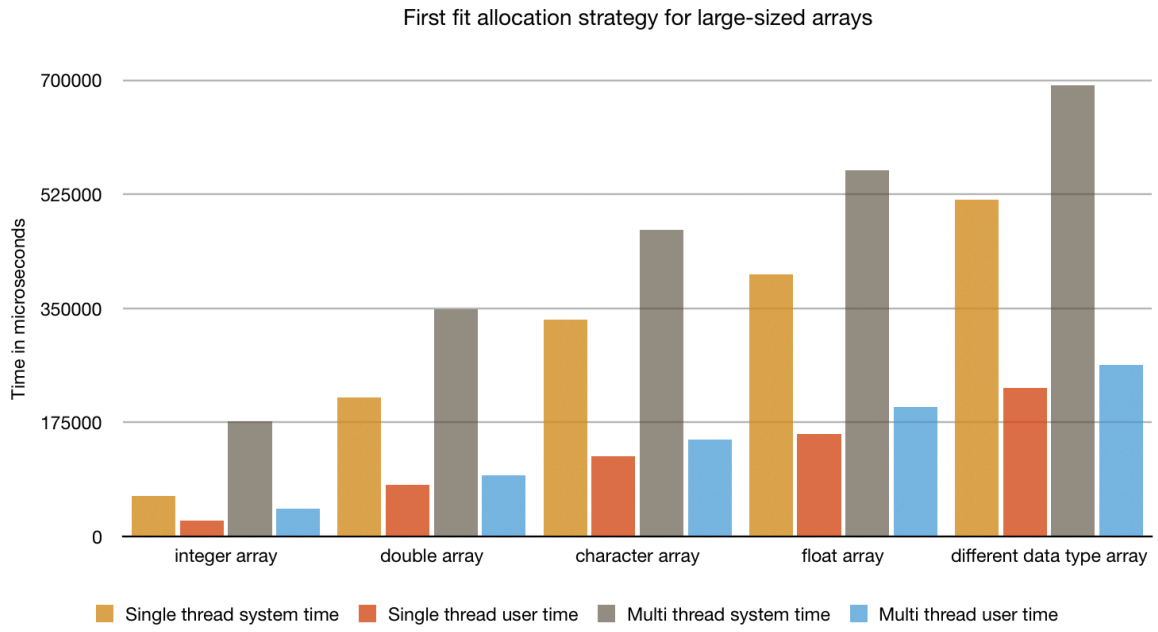


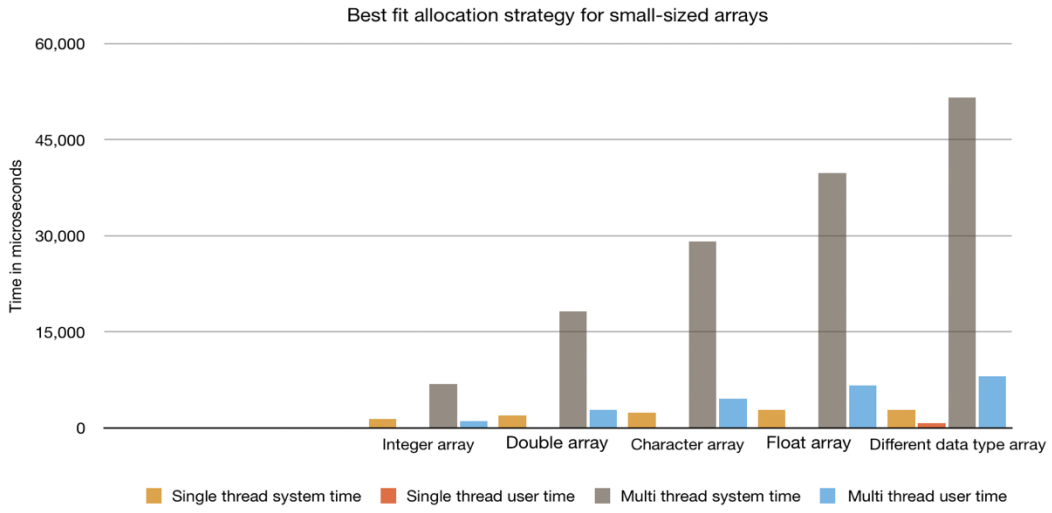
The comparison of system and user time depending on the allocation strategies for best fit and worst fit follows the same pattern as first fit.

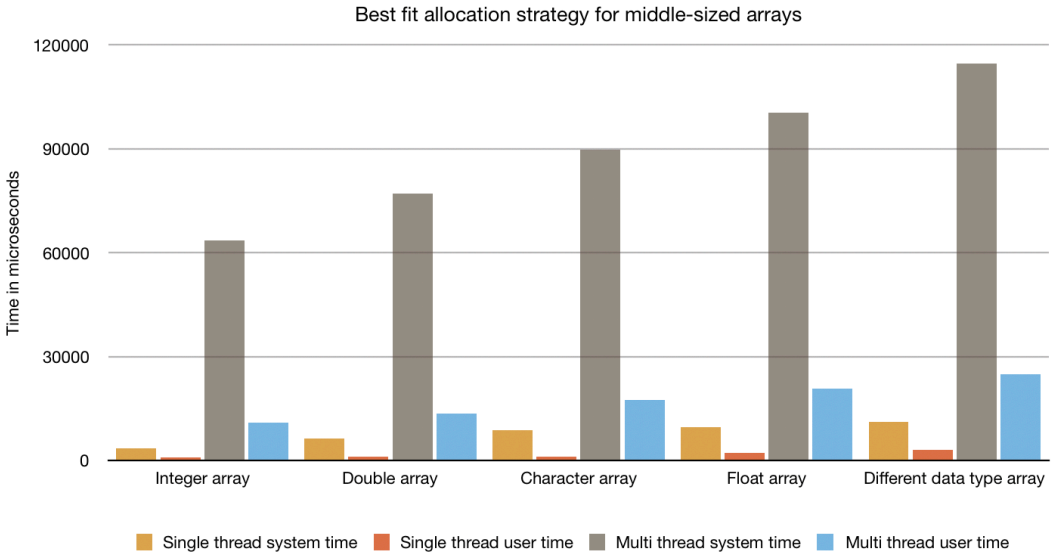
Now, changing the number of iterations to 60 yields the following results:

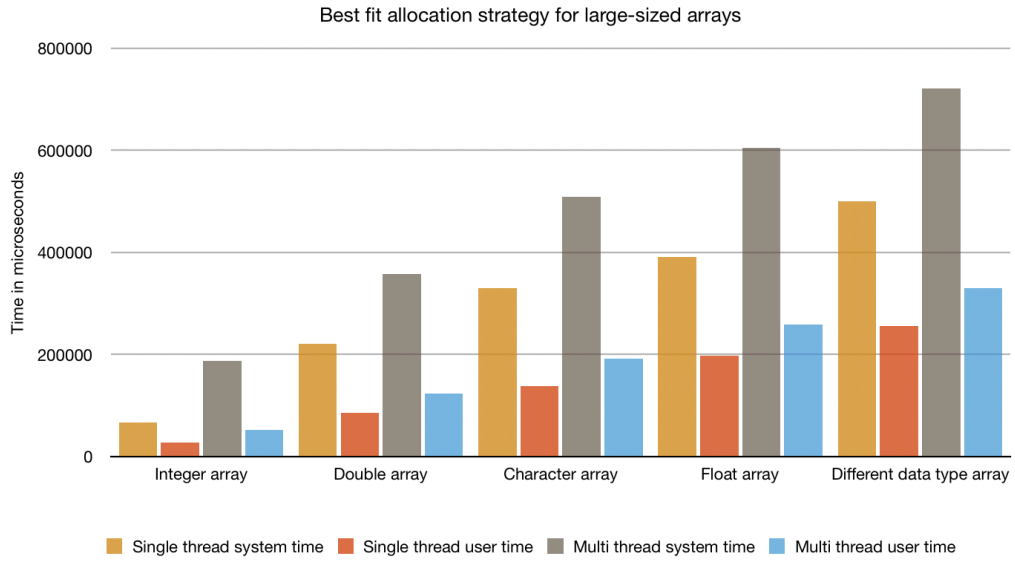

First fit allocation strategy for small-sized arrays
60,000
45,000
30,000
15,000
0
Time in microseconds
integer array
double array
character array
float array
different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time

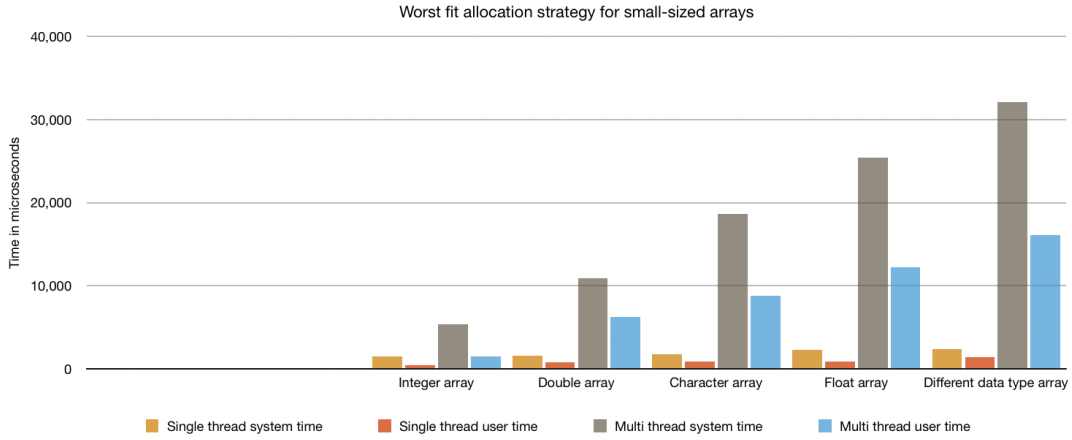

First fit allocation strategy for medium-sized arrays
120000
90000
60000
30000
0
Time in microseconds
integer array
double array
character array
float array
different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time

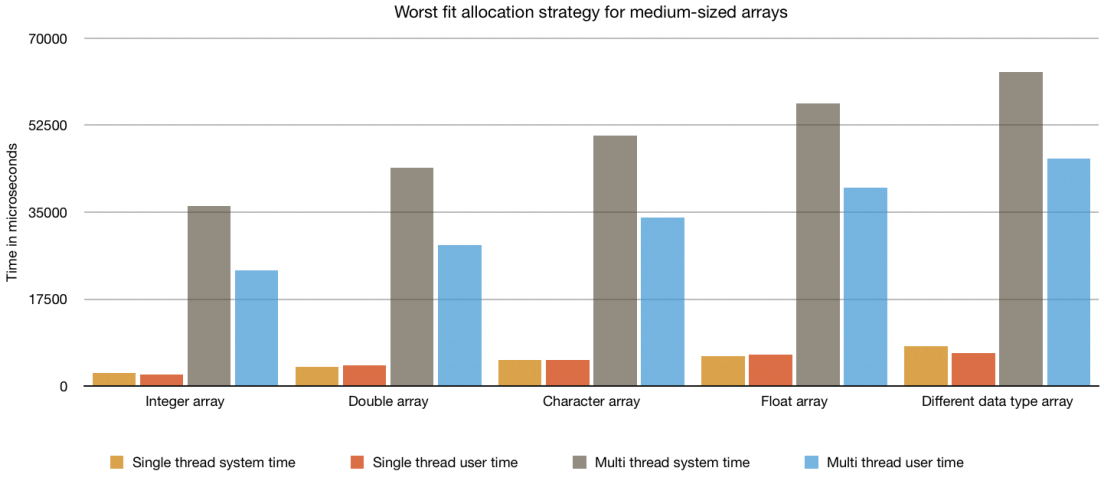

First fit allocation strategy for large-sized arrays
700000
525000
350000
175000
0
Time in microseconds
integer array
double array
character array
float array
different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time

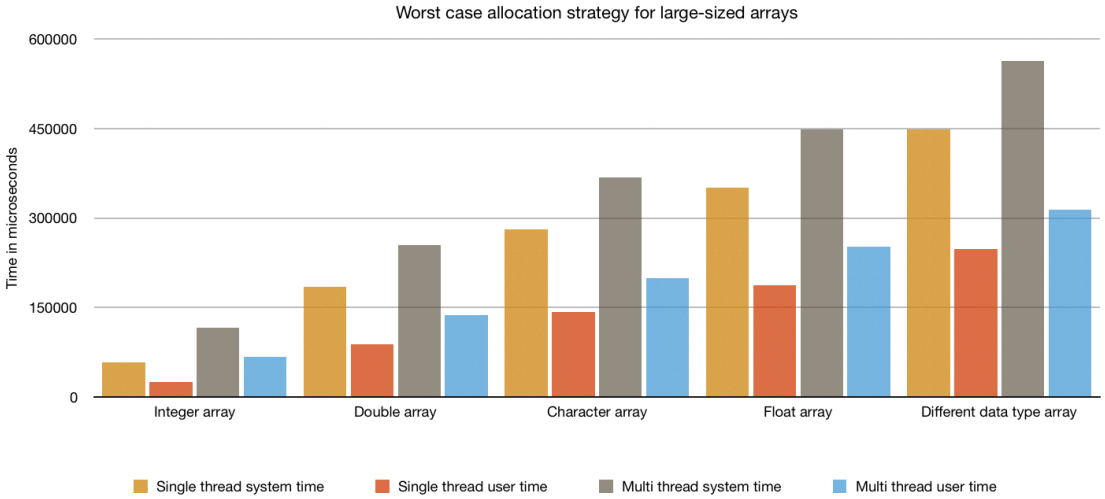

Best fit allocation strategy for small-sized arrays
60,000
45,000
30,000
15,000
0
Time in microseconds
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time


Best fit allocation strategy for middle-sized arrays
120000
90000
60000
30000
0
Time in microseconds
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time


Best fit allocation strategy for large-sized arrays
800000
600000
400000
200000
0
Time in microseconds
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time


Worst fit allocation strategy for small-sized arrays
40,000
30,000
20,000
10,000
0
Time in microseconds
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time


Worst fit allocation strategy for medium-sized arrays
70000
52500
35000
17500
0
Time in microseconds
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time


Worst case allocation strategy for large-sized arrays
600000
450000
300000
150000
0
Time in microseconds
Integer array
Double array
Character array
Float array
Different data type array
Single thread system time
Single thread user time
Multi thread system time
Multi thread user time

4. Assignment 1 vs assignment 2

Increasing the number of threads by increasing the number of iterations has shown that the user and system time also increases, thus there is a positive correlation between them. Analysing all of the graphs, I can conclude that with increasing size of arrays, the percentage difference in the time between single-threaded and multi-threaded program decreases. For smaller arrays, single-threaded program is much faster because there is less overhead. However, for large-sized arrays, multi-thread works better. one significant fact to notice is that the system time has always been much higher than user time for all types of data.

For my second assignment, the allocation strategy that is better is best fit according to all the graphs I have presented above. For most of the cases, time for best fit is lower than first fit and worst fit. In assignment 1, this was the case as well. Therefore, concurrency has not changed which allocation strategy is the best.

5. References

1. Tutorialspoint.com. (2019). *Multithreading in C*. [online] Available at: https://www.tutorialspoint.com/multithreading-in-c [Accessed 16 Oct. 2019].
2. En.wikipedia.org. (2019). *Thread safety*. [online] Available at: https://en.wikipedia.org/wiki/Thread_safety [Accessed 16 Oct. 2019].
3. Tutorialspoint.com. (2019). *Readers-Writers Problem*. [online] Available at: https://www.tutorialspoint.com/readers-writers-problem [Accessed 16 Oct. 2019].