RMIT UNIVERSITY

COSC1112/1114 Operating Systems Principles Assignment 2 REPORT

Due Date: 9 PM 18th of October 2019

Student Number: s3701799

Student Name: Syed Hariz bin Syed Azmi

Table of Contents

Evidence presentation

Experiments that are being carried out uses a separate source file from the program. The program reads a file containing a large chunk of data and stores them in a list before allocating, deallocating and reallocating them in the memory. When the program starts user can choose running the First fit, Best fit or Worst fit algorithm to allocate the data. To run the program, read READ-ME text file that was included in the code submission folder. After the program runs the chosen algorithm will display meaningful statistics such as below;

- PROCESS
    - Allocations – data that were allocated
    - Deallocations – data that were deallocated
    - Chunks allocated – chunks that were allocated in a list
    - Chunks deallocated – chunks that were deallocated from a list
    - Bytes allocated – bytes of allocation
    - Bytes deallocated – bytes of deallocation

- CHUNK STATISTICS
    - Average chunk size allocated – average of chunk size being allocated in bytes
    - Average chunk size freed – average of chunk size being freed in bytes
    - Percentage of Fragmentation – fragmentation percentage of data being processed

- TIME
    - Allocation time – time of allocation being performed
    - Deallocation time – time of deallocation being performed
    - Reallocation time – time of reallocation being performed

- PROGRAM EDGE
    - New program edge – the new program edge *after* process is executed
    - Old program edge – the original program edge *before* process is executed

Experiment file

Name	**names**
Type	.txt
Description	A file that consists a large chunk of random names.
File size	179349 bytes
Data volume	21991 words

Program evidence

Algorithm – First fit

```
[s3701799@csitprdap03 TEST5]$ ./allocate firstfit
*****PROGRAM BEGIN*****
Algorithm choice: firstfit
Old program edge: 0x249c000

*****PROGRAM STATUS*****
Allocating data...
Deallocating data...
Reallocating random data from text file...

*****PROGRAM SUMMARY*****
FILE PROPERTIES
Text file: names.txt
Total words: 21991 words
File size: 179349 bytes

PROCESS
Allocations: 3200 Deallocations: 1600
Chunks Allocated: 1600 Chunks Freed: 267
Bytes Allocated: 13246 Bytes Freed: 766

CHUNK STATISTIC
Average chunk size Allocated: 8 bytes
Average chunk size Freed: 2 bytes
Percentage of Fragmentation: 5.46674 %

TIME
Allocation time: 1550 ms
Deallocation time: 630 ms
Reallocation time: 1560 ms

PROGRAM EDGE
New program edge: 0x249fc73
Old program edge: 0x249c000
```

Algorithm – Best Fit

```
[s3701799@csitprdap03 TEST5]$ ./allocate bestfit
*****PROGRAM BEGIN*****
Algorithm choice: bestfit
Old program edge: 0x1300000

*****PROGRAM STATUS*****
Allocating data...
Deallocating data...
Reallocating random data from text file...

*****PROGRAM SUMMARY*****
FILE PROPERTIES
Text file: names.txt
Total words: 21991 words
File size: 179349 bytes

PROCESS
Allocations: 3200 Deallocations: 1600
Chunks Allocated: 1600 Chunks Freed: 80
Bytes Allocated: 13056 Bytes Freed: 295

CHUNK STATISTIC
Average chunk size Allocated: 8 bytes
Average chunk size Freed: 3 bytes
Percentage of Fragmentation: 2.20957 %

TIME
Allocation time: 1420 ms
Deallocation time: 610 ms
Reallocation time: 1550 ms

PROGRAM EDGE
New program edge: 0x1303a4f
Old program edge: 0x1300000
```

Algorithm – Worst Fit

```
[s3701799@csitprdap03 TEST5]$ ./allocate worstfit
*****PROGRAM BEGIN*****
Algorithm choice: worstfit
Old program edge: 0x2408000

*****PROGRAM STATUS*****
Allocating data...
Deallocating data...
Reallocating random data from text file...

*****PROGRAM SUMMARY*****
FILE PROPERTIES
Text file: names.txt
Total words: 21991 words
File size: 179349 bytes

PROCESS
Allocations: 3200 Deallocations: 1600
Chunks Allocated: 1600 Chunks Freed: 1233
Bytes Allocated: 14558 Bytes Freed: 4735

CHUNK STATISTIC
Average chunk size Allocated: 9 bytes
Average chunk size Freed: 3 bytes
Percentage of Fragmentation: 24.5426 %

TIME
Allocation time: 1410 ms
Deallocation time: 620 ms
Reallocation time: 1590 ms

PROGRAM EDGE
New program edge: 0x240cd66
Old program edge: 0x2408000
```

Discussion of Results

In this section of the report, graphs are being represented to illustrate time performance, fragmentation percentage and bytes allocated and freed for all 3 strategies after processing its algorithm. In the program, a simple `std::clock()` function was added to clock the starting time of each algorithm and once the algorithm completes its process, we calculate the final time in milliseconds with this simple line of computation;

```
finalTime = (std::clock() - clockStart) / (double)(CLOCKS_PER_SEC / 1000).
```

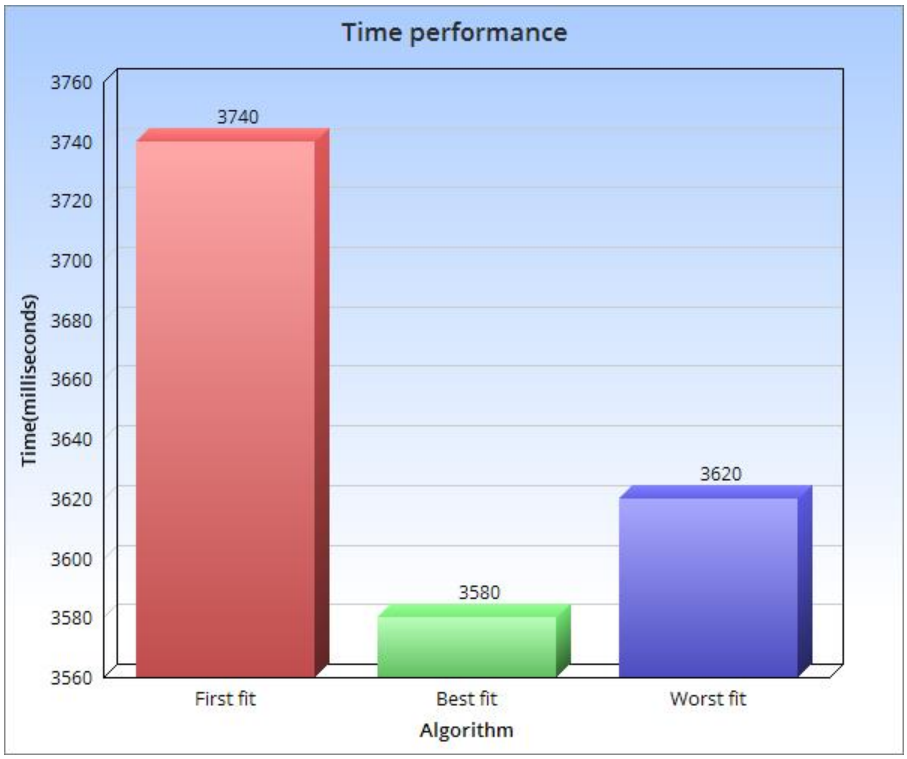
Fragmentation percentage is calculated with a simple division of; 1 - the size allocated in bytes / size allocated in bytes + size freed in bytes

```
fragmentationPerCent = 1 - ((float)sizeAllocated / ((float)sizeAllocated +
(float)sizeFreed));
```

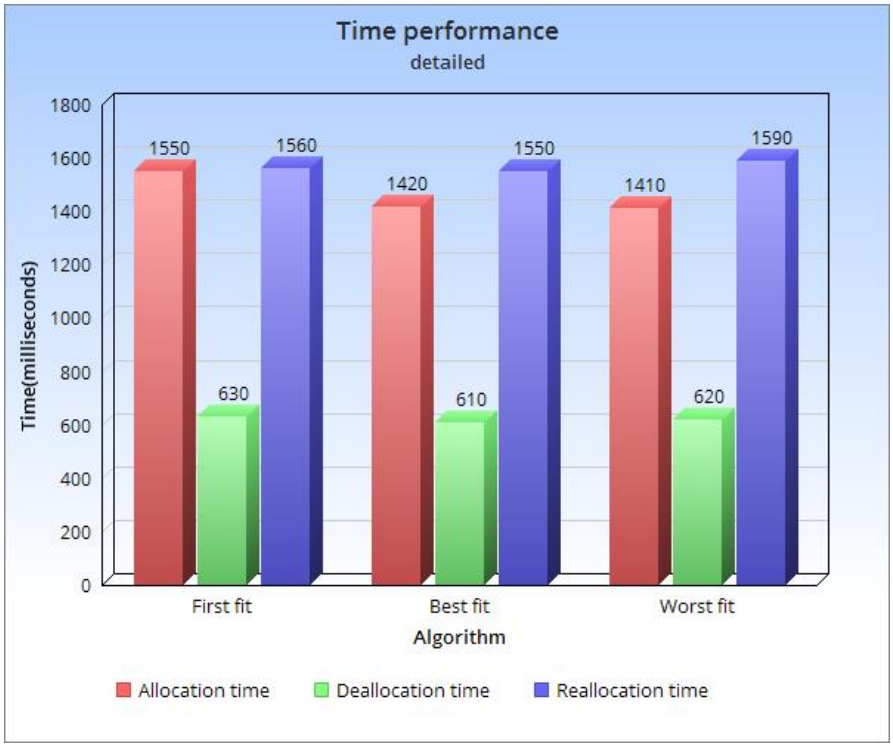
Bytes allocated and bytes freed are the values that chunks being allocated in the 2nd list and chunks that are being freed from the 1st list respectively.

The graphs are being utilized to analyse the performance of each 3 described algorithms. Further discussion will be clarified based on the results that were prompted on the application to oversee detailed comparison between them.

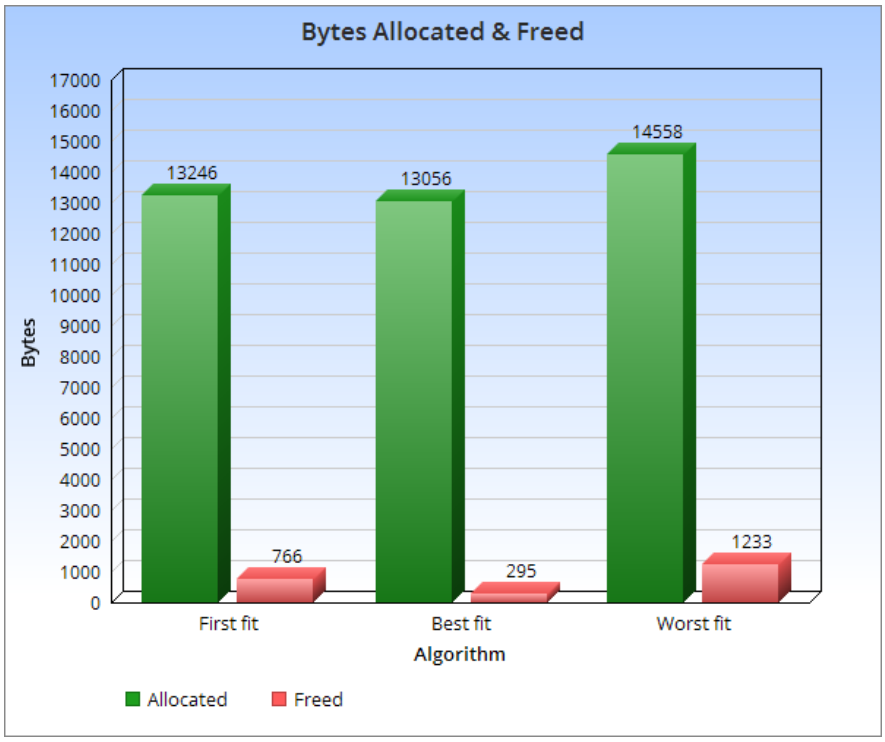
Time performance – total



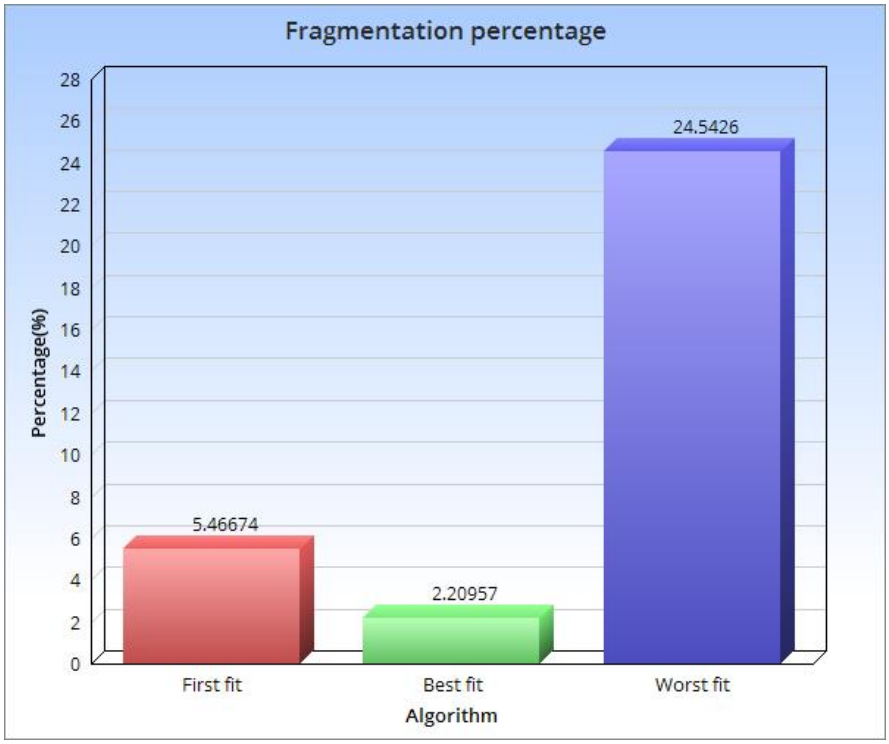
Time performance – detailed



Bytes allocated and freed



Fragmentation percentage



Discussion

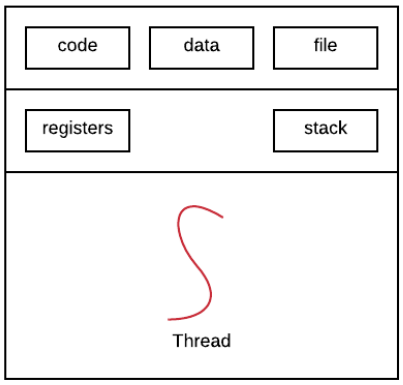
First fit	Best fit	Worst fit
Bytes Allocated: 13246 Bytes Freed: 766	Bytes Allocated: 13056 Bytes Freed: 295	Bytes Allocated: 14558 Bytes Freed: 1233
Fragmentation percentage: 5.46674 %	Fragmentation percentage: 2.20957 %	Fragmentation percentage: 24.5426 %
Allocation time: 1550 ms Deallocation time: 630 ms Reallocation time: 1560 ms Total time: 3740 ms	Allocation time: 1420 ms Deallocation time: 610 ms Reallocation time: 1550 ms Total time: 3580 ms	Allocation time: 1410 ms Deallocation time: 620 ms Reallocation time: 159 ms Total time: 3620 ms

Based on the information and statistics that were collected, we can easily compare the 3 algorithms. Focusing on Bytes Allocated and Freed between the 3 algorithms, we can conclude that Worst fit algorithm needs a bigger section of memory when allocating data than the other 2 algorithms. Even though Worst fit algorithm freed a fairly amount of bytes from the memory for other processes to use, Worst fit algorithm has an astounding chance of getting a fragmentation comparing to the other 2 algorithms with 25% (after rounding). The efficiency of real-time systems relies on the availability of this memory blocks with minimum fragmentation in a timely manner. The Best fit algorithm has the fastest total time of allocating, de-allocating and reallocating data between First fit and Worst fit algorithms with 3580 milliseconds. As mentioned, the importance of memory management efficiency has been broached many times but the difference between the 3 algorithms are in milliseconds where their differences merely affect the judgement of their time performance to determine the fastest one. Therefore, from the statistics that were collected, we can prove

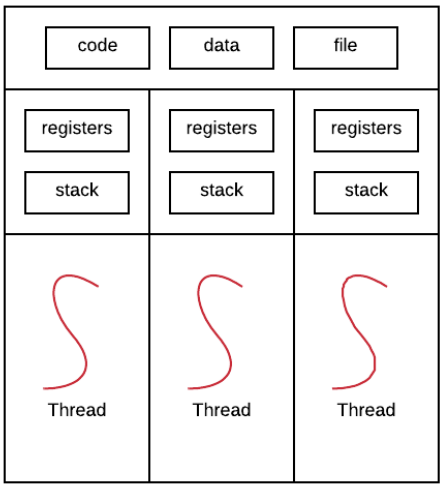
that Best fit algorithm has the best statistics between those 3 algorithms. Although Best fit has lesser bytes freed than First fit, it allocates a lesser amount of bytes in the memory with a minimum chance of getting fragmentation. We have to take this under consideration because fragmentation causes a memory block to be unused which can affect the performance. Memory allocation algorithm must efficiently allocate requested memory blocks with minimum memory loss.

Concurrency Discussion

In operating system, processes might look like they are running at the same time but it might not be. Concurrency can be viewed as multiple processes of the same resource. It is the interleaving of processes in time to give the appearance of simultaneous execution. In a multi-processor or core system, concurrency can greatly speed up some process. If we want a process to be able to execute on multiple CPUs at a time and to take advantage of the multi-core systems, the process must have several execution-context called threads. A thread is a lightweight process, it is a basic CPU utilization. A thread is an active entity which executes a part of a process. Multiple threads execute simultaneously with each other which results in the execution of a single whole process. Since several threads of a process are being executed at the same time, it is important that they maintain coordination between each other. Coordination is required between threads in order to share a system resource such as; CPU, memory. The threads of a process are a part of virtual address space, they share all the code, file and data. However, each threads has different instruction when accessing the address space. Each thread will access a different portion of the space. To visualize threads, different thread has a different stack, stack pointer, program counter and other registers. Multi-threaded process is much more complex than a single-threaded process where it contains all the information that are being shared among all the threads with separate execution of all threads. The diagram below is a visualization of single-threaded process and multi-threaded process.



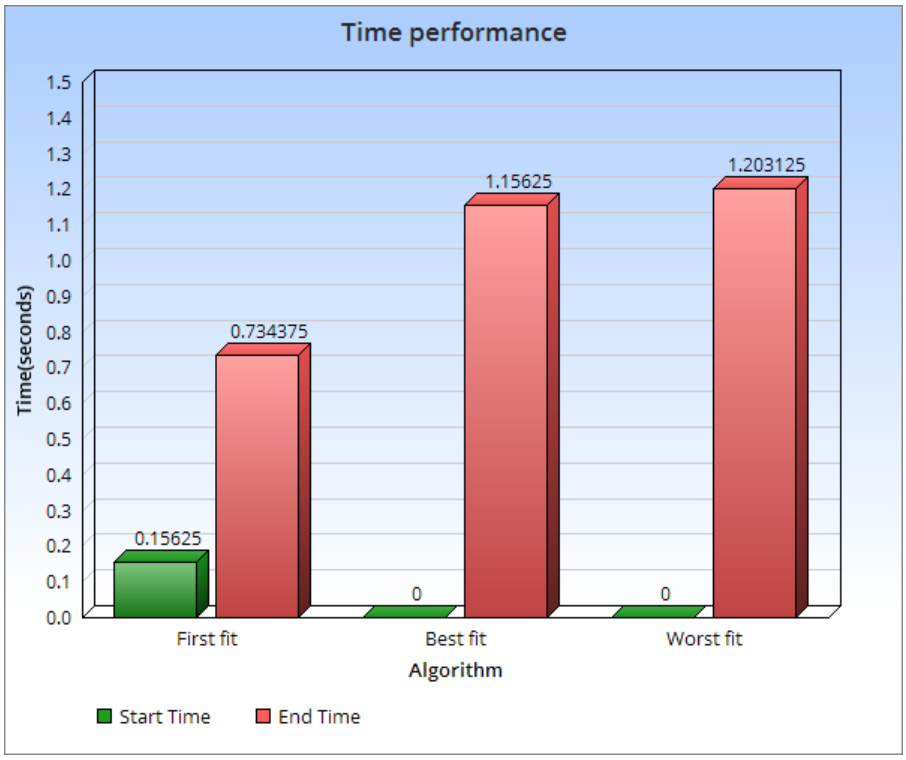
Single-threaded process



Multi-threaded process

Comparison with Assignment 1 results

Assignment 1 – Time performance



Assignment 2 – Time performance

Analysis

In Assignment 1, we were to conduct findings between those 3 described strategies based on its time performance and allocation of data. However, in Assignment 2 we were instructed to implement and investigate locking functions on top of Assignment 1 code.

Read Locks

When accessing data structures, we have to lock the whole data structure for reading and enable new readers to read it. Once a writer is requested, current readers must be allowed to complete their process and new readers must wait.

Write Locks

When a thread wants to modify the allocation or freed list, it must gain exclusive access to the list by using semaphore or mutex to lock it and no other threads can modify the list. Mutex was used for this implementation.

Acquired from the result of Assignment 2 and Assignment 1, we can agree using multi-threading is beneficial particularly in time performance of each algorithm to complete its process. As being shown in enhanced Assignment 2 code where threads are included, all algorithms took much lesser time to complete its process. In fact, by a great extent. Time performance in Assignment 1 is being measured in seconds, wherein Assignment 2 to display an accurate value of time, it has to be in milliseconds. Comparing all the algorithms, time in First fit doubled, Best fit tripled and Worst fit almost quadrupled in Assignment 1 comparing to Assignment 2. This experiment manifests a true benefit of how threads can affect the performance of a process.

Conclusion

Traditionally, multiple single-threaded processes have been used to achieve parallelism but some programs can benefit from a finer level of parallelism. Multi-threaded processes offer parallelism within a process. Multi-threaded process also shares many concepts involved in programming multiple single-threaded processes.

In conclusion, there are benefits of using multi-threaded process;

Parallelization

In multi-processor architecture, different threads can execute different instructions at a time which speeds up the execution of the process.

Specialization

We can manage the threads by specializing different threads to perform a different task. For example, giving higher priority to threads that are executing an important task.

Efficiency

Multi-threaded programs are more efficient as threads share address space. Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.