Advanced Journal of Science, Technology and Engineering
ISSN: 2997-5972
Volume 4, Issue 3, 2024 (pp. 1-13)
www.abjournals.org

USING A MATHEMATICAL MODEL TO EVALUATE THE EFFICIENCY OF MEMORY ALLOCATION ALGORITHM

Ogba P. O.[1] and Bello M.[2]

[1]Computer Science Department, Kogi State Polytechnic, Lokoja.
Email: ogbapaul@gmail.com

[2]Information Technology and Resource Center, Prince Abubakar Audu University, Anyigba.
Email: bmuriana685@gmail.com

Cite this article:

Ogba P. O., Bello M. (2024), Using a Mathematical Model to Evaluate the Efficiency of Memory Allocation Algorithm. Advanced Journal of Science, Technology and Engineering 4(3), 1-13. DOI: 10.52589/AJSTE-JIHPGG70

Manuscript History

Received: 23 May 2024

Accepted: 31 Jul 2024

Published: 13 Aug 2024

ABSTRACT: *One of the key research areas in operating systems is memory allocation and process management by the operating system. Memory allocation is the process of allocating blocks of memory to different executing processes in order to improve overall system performance. This paper analyses the efficiency of memory allocation algorithms (first fit, best fit and worst fit) in the multiple partition and contiguous memory allocation scheme. A Mathematical model was used on process sizes in terms of percentage internal fragmentation (IF%), percentage external fragmentation (EF%), and percentage total utilization (TU%), this was done to compare the performance of the memory allocation algorithm. Results from the analysis show that the best fit made more efficient use of the available memory space than that of first-fit and worst fit.*

KEYWORDS: Memory allocation, Operating system, Mathematical model, Internal fragmentation, External fragmentation, Memory utilization, Total Utilization.

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

INTRODUCTION

The act of controlling computer memory at the system level is known as memory management and it is the function responsible for assigning and managing the primary memory of the computer (Abraham, Peter, and Greg, 2010). Memory management must be able to dynamically assign sections of memory to applications as they request it, and then free it for reuse when it is no longer required. This is crucial in any complex computer system where multiple processes may be running at the same time. Memory management is the function in operating systems that manages the computer's primary memory. The function keeps track of each memory location's status, whether it's allocated or unallocated. It decides how memory is distributed across multiple processes, determining who gets memory, when they get it, and how much they get. It is decided which memory regions will be allotted when memory is allocated. It monitors and reports when memory is freed or unallocated.

Several approaches for improving memory management effectiveness have been devised. To efficiently organize memory, various memory allocation techniques have been proposed. Allocation algorithms are used to decide which of the available slots in the partitioned memory block can be assigned to a process. The most basic form of memory management, known as partitioning, involves allocating a single contiguous portion of memory to each process. The simplest form of partitioning is to divide memory into numerous fixed-size sections in advance, which is known as fixed partitioning. Fixed partitioning is the simplest and oldest method of putting multiple processes in the main memory. There are two types of fixed partitioning: equal-sized partitioning and unequal-sized partitioning. Any process with a size less than or equal to the partitioning size can be loaded into any of the partitions that are available. Overlays and internal fragmentation are two types of problems that plague fixed-size partitions. When a process's size exceeds the partition's size, it suffers from an overlaying problem, in which just the information that is required is maintained in memory. Overlays are a difficult and time-consuming process. When the memory allotted to a process is slightly greater than the amount requested by the process, free space in the allocated memory is created, resulting in internal fragmentation. When processes are given fixed-sized memory blocks, this happens. Each process is assigned to the smallest division in which it fits in many queues, minimizing the internal fragmentation problem.

RELATED WORK

Modern operating systems offer efficient memory management, and research is still ongoing to improve how memory is allocated for applications. The main problem that memory allocation algorithms face is efficiently allocating demanded memory blocks to demanding applications with the shortest response time and the least amount of memory loss known as fragmentation of memory (Soto and Sevaux, 2011). Finding a block of unused memory of a suitable size is the task of completing an allocation request. Memory demands are met by allocating chunks of memory from the stack or free store, which is a vast pool of memory. Some parts of the stack are in use at any given time, while others are "free" (unused) and hence accessible for future allocations. External fragmentation, which occurs when there are many small gaps between allocated memory blocks, and invalidates their use for an allocation request, is one issue that complicates the implementation (William, 2017). The information of the allocator can bloat the size of (individually) minor allocations as well. The dynamic

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

memory allocation mechanism used can have a substantial impact on performance. The overheads involved with a variety of allocators are illustrated in research completed by Digital Equipment Corporation in 1994. The shortest average instruction route length for allocating a single memory slot was 52 (as assessed by an instruction level profiler on a variety of software).

When a system has memory that is nominally free but that the computer can't use, it's called memory fragmentation. When data is removed, the memory allocator splits and allocates memory blocks as needed by programs; when data is deleted, more memory blocks are freed up in the system and re-added to the available memory pool. Fragmentation occurs when the operations of the allocator or the recovery of recently occupied memory segments result in chunks or even bytes of memory that are too small or segregated to be utilized by the memory pool. Fragmentation can eat up a lot of free memory on a computer, and it's a common cause of unpleasant out-of-memory error messages.

Memory Management Approaches

There are many memory management approaches to choose from such as partitioned allocation, paged memory management, single contiguous allocation, etc.

Partitioned Allocation: Primary memory is partitioned into numerous memory partitions, which are normally contiguous portions of memory. Each partition could hold all of the data needed for a single job or task. Memory management involves allocating a partition to a work when it begins and unallocating it when the job is completed. To prevent jobs from interacting with one another or with the operating system, partitioned allocation usually necessitates certain hardware support. A lock-and-key system was utilized on the IBM System/360. Other systems employed base and bounds registers to store the partition's limits and to signal inappropriate accesses (Samanta, 2004).

Paged Memory Management: The basic memory of the computer is divided into fixed-size units called page frames, and the virtual address space of the software is divided into pages of the same size. Pages are mapped to frames by the hardware memory management unit. While the address space appears to be contiguous, physical memory can be provisioned on a page basis. Each job usually operates in its own address area when using paged memory management. Meanwhile, certain single address space operating systems, such as IBM OS/VS2 SVS, which ran all jobs in a single 16MiB virtual address space, run all processes within a single address space.

Single Contiguous Allocation: The simplest memory management strategy is single contiguous allocation. The single application has access to the whole computer's memory, with the exception of a small fraction allocated for the operating system. MS-DOS is an example of a system that uses this method of memory allocation. This strategy could also be used by an embedded system running a single application. By exchanging the contents of memory to switch between users, a system with a single contiguous allocation can nonetheless multitask.

Memory Fragmentation

When a system has memory that is nominally free but that the computer can't use, it's called memory fragmentation. When data is removed, the memory allocator splits and allocates memory blocks as needed by programs; when data is removed, more memory space is released in the system and added back to the memory pool. Fragmentation occurs when the

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

operations of the allocator or the recovery of recently occupied memory segments result in blocks or even bytes of memory that are small or fragmented to be utilized by the memory pool. Fragmentation can eat up a lot of free memory on a computer, and it's a common cause of annoying out-of-memory error messages. There are two major types of memory fragmentation;

Internal Fragmentation: This happens when the memory allocator keeps excess space unfilled inside a block of memory assigned to a process, this error occurs. This frequently arises because the processor's design requires memory to be partitioned into blocks of specific sizes, such as four, eight, or sixteen bytes. When this happens, a process that requires 85 bytes of Memory space, for example, may be given a block with 90 or even 92 bytes. The extra bytes that the client doesn't require are wasted, and over time, these small chunks of unused memory might add up to significant amounts of memory that the allocator can't use.

External Fragmentation: When the memory allocator leaves areas of unused memory blocks between allotted memory blocks, this occurs. The free block is fractured if many memory blocks are created in a continuous line but one of the intermediate blocks is freed. The allocator can still utilize the block in the future if it needs memory that fits in that block, but it's no longer usable for bigger memory needs. It can't be regrouped with the system's total free memory since total memory must be contiguous in order to be usable for larger processes.

Memory Allocation Algorithms

In this paper, we looked at processes and their sizes on a memory partition. The algorithms' efficiency is determined by their ability to use as much memory as possible. Since the holes aren't ordered in any particular sequence, such as from smallest to largest or largest to smallest, the worst case of binary search is always used. Where O(log(n)) is always the time complexity. The algorithms used are first fit, best fit and worst fit.

First Fit: This works by scanning the memory from the beginning and allocating the first available block that is large enough. It is one of the quickest algorithms since it searches as little as possible. However, if the remaining unused memory spaces after allocations are too small, they become waste. As a result, requests for significant amounts of memory cannot be fulfilled. We will show which approach makes the most efficient use of memory in the following problem.

Best Fit: Best fit requires searching the entire list of blocks for the one that is closest in size to the request and allocating that block. Because it examines the smallest free partition first, memory consumption is substantially better than First-fit. However, it is slower and may even cause memory to fill up with tiny worthless holes.

Worst Fit: Worst fit requires searching the full list of blocks for the largest block and allocating that block which reduces the rate at which tiny gaps are produced. This technique, on the other hand, results in the largest residual block, which may be large enough to hold another process. However, if a process that requires more memory arrives later, it will be unable to fit because the largest hole has already been split and occupied.

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

Advanced Journal of Science, Technology and Engineering
ISSN: 2997-5972
Volume 4, Issue 3, 2024 (pp. 1-13)
www.abjournals.org

Table 1: Processes and their sizes

Processes (P)	Sizes (K)
P1	115
P2	250
P3	400
P4	300
P5	100

The memory is divided into five partitions as shown in Table 2. It has a total memory size of 1550 KB where the operating system (OS) takes 100k and the processes take 1450 KB. This is indicated in the memory partition diagram given in Figure 1.

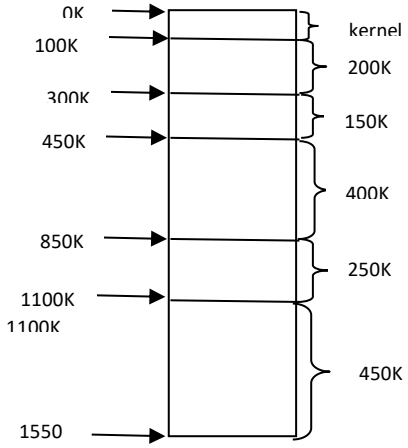

Figure 1: Pictorial Representation of the Memory Partitions

Table 2: Size of the memory partitions

Partitions/Holes	Sizes (K)
1	200
2	150
3	400
4	250
5	450

Mathematical Model of the Algorithm

First Fit Analysis

Using the Table 1 data set, the First fit algorithm will search the memory partitions (holes) from the upper level of the memory. We have partitions 200K, 150K, 400K, 250K, 450K. Process P1 is first taken and the algorithm checks if the first partition is equal to or greater than the size of the process. P1 = 115K and partition1 = 200K so P1 is allocated to the first partition. i.e. P1 with the size 115K goes into partition 200K,

IF = (200 – 115) K = 85K(unused)

Next, take process P2 which is 250K, the First partition has been allocated to process P1, and

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

the algorithm starts checking with the second partition to know where the process can fit in. The second partition is less than P2 and the algorithm proceeds to check the next partition which is partition 3, the third partition is greater than P2. So, P2 is allocated to the third partition. i.e. P2 with the size 250K goes into partition 400K,

IF = (400-250)K, = 150K(unused)

Next, take process P3 with size 400K, the First and third partitions have been allocated to processes. The second partition is less than P3, the fourth partition is less than P4 but the fifth partition is greater than P3 so P3 (400K) is allocated to the fifth partition (450K).

IF = (450-400)K = 50K(unused)

Next, take process P4 with the size 300K, P4 > IF1, P4 > EF2, P4> IF3, P4> EF4, P4> IF5. Hence, P4 cannot be allocated.

Lastly, Process P5 with the size 100k is allocated to the second partition (150k) because the second partition is greater than P5 and it is the first partition to consider.

IF = (150-100)K = 50K(unused)

Table 3: First Fit IF and EF

Processes (P)	Allocated Partition No	IF (K)	EF(K)
P1	1	85	
P2	3	150	
P3	5	50	
P4	Not allocated		250
P5	2	50	
Total		335	250

Total Memory Used = 115K + 250K + 400K + 100K= 865K

Memory Not used = 85K + 150K + 50K + 50K = 335K

% Total memory utilization = 1115/1450 *100 = 76.9%

% Internal fragmentation = 335/1450 * 100 = 23.10%

% External fragmentation = 250/1450 * 100 = 17.24%

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

Advanced Journal of Science, Technology and Engineering
ISSN: 2997-5972
Volume 4, Issue 3, 2024 (pp. 1-13)

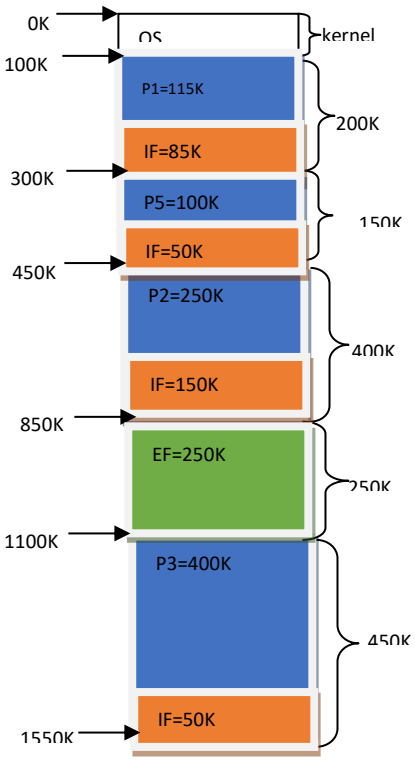

Figure 2: First fit memory allocation Algorithm

Best Fit Analysis

Using table 1 and 2 data sets, Process P1 with size 115K (P1 = 115K), available partitions are (200, 150, 400, 250, 450)K. The best fit algorithm searches the partitions and compares the sizes then allocates the smallest partition that is big enough to accommodate the incoming process. So P1(115K) is allocated to the partition of size 150K,

IF = (150K – 115K) = 35K(unused)

Next, take process P2 with size 250K, remaining partitions to search through are: (200, 400, 250, 450)K partitions. The smallest partition that is big enough to accommodate P2 is the fourth partition. Hence, process P2 is allocated into partition 250K,

IF = (250K – 250K) = 0K.

Next is process P3 with size 400K and available partitions to search through are: (200, 400, 450)K partitions. So P3 is allocated to partition 400K,

IF = (400K – 400K) = 0K.

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

Next, take process P4 with size 300K. Available partitions to search through are: (200, 450)K. P4 is allocated to partition 450K,

IF = (450K – 300K) = 150K(used)

Lastly, take Process P5 with the size 100k, partitions to search through are: (EF1, IF5) i.e. 200K and 150K. P5 is allocated to 150K

IF = (150K – 100K) = 50K(unused)

Table 4: Best Fit IF and EF

Processes (P)	Allocated Partition No	IF (K)	EF(K)
P1	2	35	
P2	4	0	
P3	3	0	
P4	5	-	200
P5	5	50	
Total		85	200

Total Memory Used = 115K + 250K + 400K + 300K + 100K = 1165K

Memory Not used = 35K + 50K = 85K

% Total memory utilization = 1365/1450 *100 = 94.14%

% Internal fragmentation = 85/1450 * 100 = 5.86%

% External fragmentation = 200/1450 * 100 = 13.79%

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

Advanced Journal of Science, Technology and Engineering
ISSN: 2997-5972
Volume 4, Issue 3, 2024 (pp. 1-13)

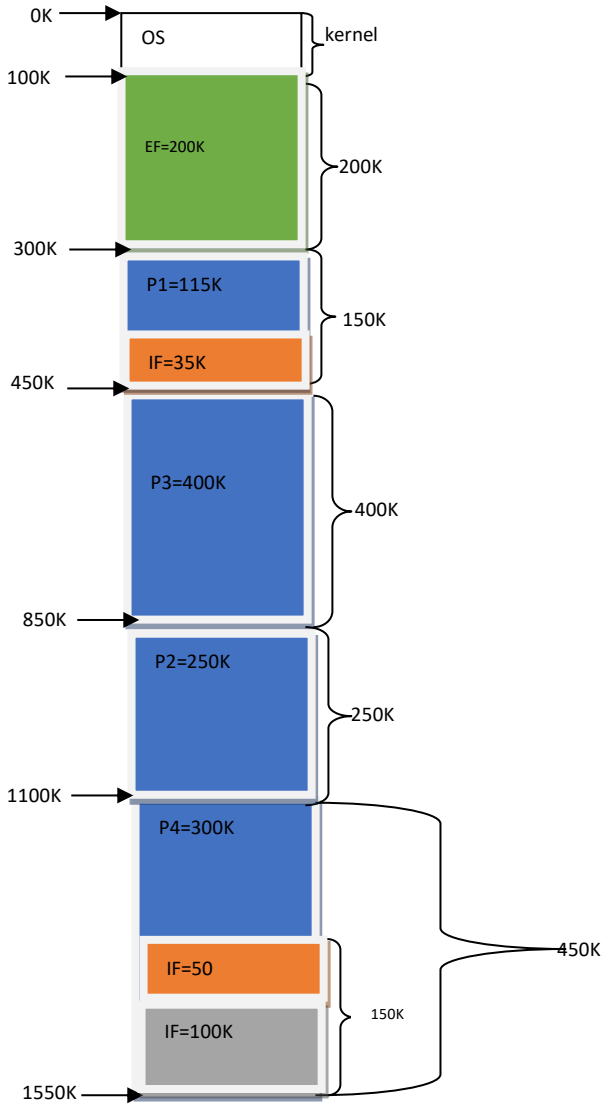

Figure 3: Best fit memory allocation Algorithm

Worst Fit Analysis

Allocate the largest partition to the first process on the queue irrespective of the position of the partition. So Taking process P1 with size 115K and available partitions to search through are (200, 150, 400, 250, 450)K. The largest partition = 450K, therefore allocate P1(115K) to 450K.

IF = 450 – 115K = 335(used).

Next, take process P2 with size 250K, available partitions to search through (200, 150, 400, 250) K and IF of 335K, largest is 400K, therefore process P2 with size 250K is allocated to 400K.

IF = 400K - 250K = 150K(unused)

Next, process P3 with the size 400K, available partitions to search through are (200K, 150K,

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

250K), IF of 335K and 150K. P3 > EF1, P3 > EF2, P3> IF3, P3> EF4, P4> IF5. Hence, P3 cannot be allocated.

Then, process P4 with the size 300K, available partitions to search from: (200K, 150K, 250K) and IF of 335K and 150K. largest = 335K

IF = 335K - 300K = 35K(unused)

And lastly P5(100K) is allocated to 250k

IF = 250K - 100K = 150K(unused)

Table 5: Worst Fit IF and EF

Processes (P)	Allocated Partition No	IF (K)	EF(K)
P1	5	-	
P2	3	150	
P3	Not allocated	0	
P4	5	35	200
P5	4	150	150
Total		335	350

Total Memory Used = 115K + 250K + 300K + 100K = 765K

Memory Not used = 35K + 150K +150K = 335K

% Total memory utilization = 1115/1450 *100 = 76.90%

% Internal fragmentation = 335/1450 * 100 = 23.10%

% External fragmentation = 350/1450 * 100 = 24.14%

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

Advanced Journal of Science, Technology and Engineering
ISSN: 2997-5972
Volume 4, Issue 3, 2024 (pp. 1-13)

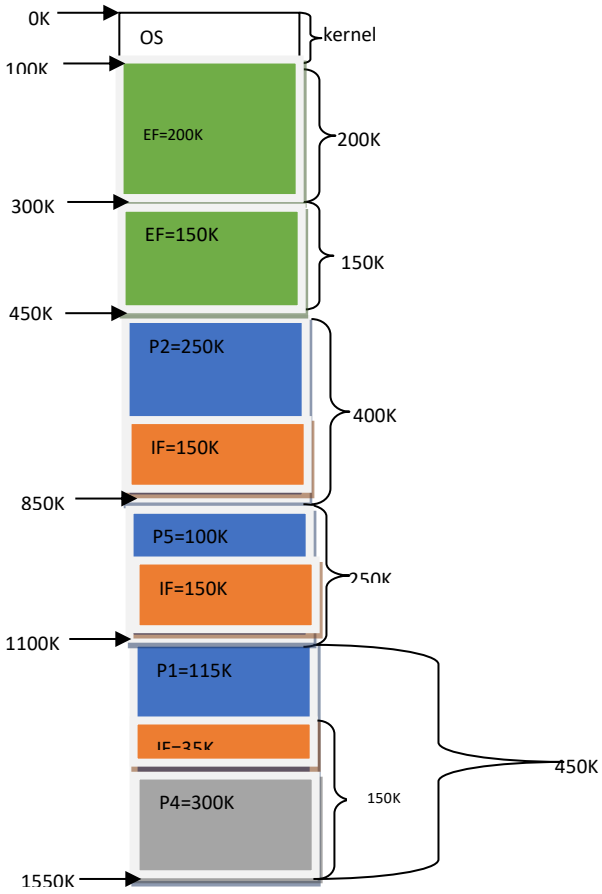

Figure 4: Worst fit memory allocation Algorithm

Table 6: TU%, EF% and IF% for the algorithms

	First Fit	Best Fit	Worst Fit
TU (%)	79.0	94.14	76.90
IF (%)	23.10	5.86	23.10
EF (%)	17.24	13.79	24.12

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

Advanced Journal of Science, Technology and Engineering
ISSN: 2997-5972
Volume 4, Issue 3, 2024 (pp. 1-13)
www.abjournals.org

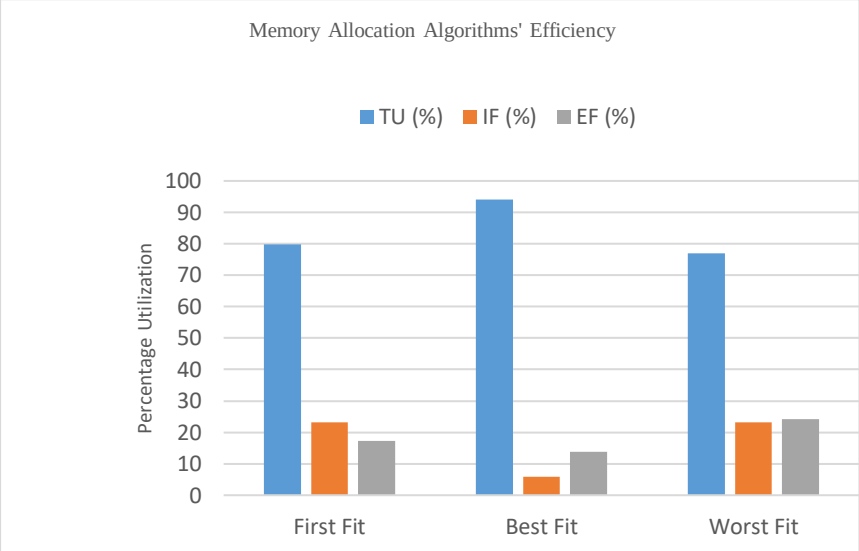

Figure 5: Bar chart showing the efficiency of the algorithms

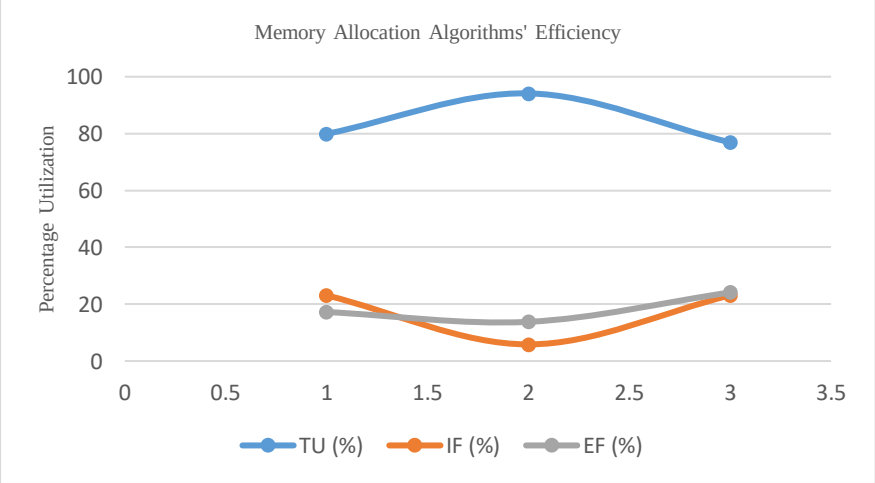

Figure 6: Line graph showing the efficiency of the algorithms

RESULTS AND DISCUSSION

The analysis findings of the article are presented in the above tables and figures. It is observed that best fit algorithms contribute less partition to external fragmentation. So it is clear that the best-fit algorithm is more efficient in memory usage, whereas the performance of the first fit and the worst fit algorithm are nearly the same.

From table 6, it is noteed that best fit algorithm has the minimum internal fragmentation of 5.86% with 94.14%. Worst fit algorithm has internal fragmentation of 23.1% with 79.72% of total memory utilization allocation. First fit algorithm also has internal fragmentation of 23.1% and its total memory utilization is 79.72%. Therefore it can be stated that best-fit used the available memory space more efficiently than first-fit and worst-fit.

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70

CONCLUSION

This research provided a detailed description and analysis of the three major memory allocation methods used in multiple partition contiguous memory allocation schemes. While comparable results were obtained using the same partitioning configuration on the data set of processes, the mathematical model can be used to argue that the best fit method is generally the best in terms of memory use.

REFERENCES

Abraham S., Peter B.G., and Greg G. "Operating System Concepts", John Wiley & Sons, INC., January 1, 2002.

Erica K. "What Is Internal & External Memory Fragmentation?" [Online] (http:// everyday life. globalpost.com/internal-external-memory- fragmentation-28851.html) 2015

Ledisi G. k, Tamunoomie S. G. "Efficiency of Memory Allocation Algorithms Using Mathematical Model." International Journal of Emerging Engineering Research and Technology Volume 3, Issue 9, September, 2015, PP 55-67 ISSN 2349-4395.

Muhammad A.A. "Challenges and Techniques for Algorithms in Relation with Today's Real Time Needs." International journal of Multi-Disciplinary Sciences and Engineering, vol.7, No.3, March 2016.

Paul G. "Multics Virtual Memory." Tutorial and reflection. Retrieved May 9, 2012.

Rachael C., Okuthe P., Kogeda M.L. "An optimized main memory management partitioning placement algorithm." Pan African Conference on science, Computing and Telecommunications (PACT), Kampala, Uganda, July 27-29, 2015.

Samanta D. "Classic Data Structures." PHI Learning Pvt. Ltd. ISBN 8120318749, P. 94, 2004.

Soto A.R.M and Sevaux M. "A mathematical model and meta-heuristic approach for a memory allocation problem." Springer science and Business media, 2011.

William S. "Operating Sytstem Internals and Design Principles." March 20, 2017.

Article DOI: 10.52589/AJSTE-JIHPGG70
DOI URL: https://doi.org/10.52589/AJSTE-JIHPGG70