CRITICAL SCRUTINYOF MEMORY ALLOCATION ALGORITHMS: FIRST FIT, BEST FIT AND WORST FIT

Amit Kumar Mandal[1],
Department of Computer Science, Moridhal College, Dhemaji, Assam, India

Dilip Kumar Baruah[2],
Department of Computer Science, Moridhal College, Dhemaji, Assam, India

Jogamohan Medak[3],
ASRLM, Majuli, Assam, India

Neelutpol Gogoi[4],
Department of Computer Science, Moridhal College, Dhemaji, Assam, India

ParthaPratim Gogoi[5],
Assam Secretariat, Assam, India

ABSTRACT
Memory allocation is the process of assigning a portion of memory to a process for execution. Partitioning makes easier to organise memory requirement of different processes. It is accomplished through a procedure called memory management. Memory requirement of process may be either contiguous allocation or non-contiguous allocation. If process requirement is contiguous, then consecutive blocks of memory have to be allocated to the process. If the requirement is non-contiguous, then process may be allocated the blocks of memory scattered all around the memory space. The efficiency of memory management depends on the utilization of memory by the process. If the process is allotted exactly the amount of memory requested by it, then it results in the maximum utilization. On the other hand, if the process is allotted more memory than requested, then excess memory is wasted as it cannot be used by another process. Such wastage of memory is referred to as internal fragmentation that has to be minimized. If the process does not get its required contiguous memory due to total available free memory is in non-contiguous form, then, it is referred to as external fragmentation. This incident should also be avoided. All these aspects should be considered while computing the memory utilization. In this paper, we will discuss about some of the popular memory allocation algorithms that are, first fit, best fit and worst fit for fixed sized and variable sized partition of contiguous nature, their performance will be analysed along with internal and external fragmentation and the whole procedure of an algorithm will be illustrated with proper diagrams. The main objective of this paper is to determine the most efficient algorithm in contiguous memory allocation that as minimum fragmentation.

KEYWORDS: best fit, first fit, fixed sixed partition, variable size partition, worst fit.

INTRODUCTION
Memory of a computer system can be considered as a storehouse of data and information to be needed by various applications at any time. Memory can be divided into two types: primary or main memory and secondary memory. Primary memory is mostly used by operating system and user defined processes for execution. Secondary memory is used for

storage of data and information [1]. In multiprogramming environment, primary memory is occupied by several processes at a time. Therefore, allocation of available memory to the processes must be managed properly for efficient functioning of the computer system. The Memory Management module of an operating system takes care of allocation and de-allocation of memory [1].

There are two types of memory management technique: Contiguous memory allocation and Non-contiguous memory allocation [2].

When a process is brought from secondary memory to main memory for execution, memory allocation for the process may be contiguous in nature, i.e., the whole process is positioned in a single segment in the main memory. This type of allocation is referred to as contiguous memory allocation. However, if the process is divided into pieces and arranged them in main memory at separate location, it is referred to as non-contiguous memory allocation [3-4]. In this paper, we will focus on contiguous memory allocation and its various methodologies.

MEMORY ALLOCATION

An easy way for allocation of memory is to divide the contiguous memory space into fixed sized partitions, which may or may not be of equal size. In this case, each process will get exactly one partition of contiguous memory for execution. When a partition is free, it is allocated to a new process. After the termination of the process, the partition becomes free and is available for another process. Degree of multiprogramming depends on the number of partitions. That means, number of processes cannot exceed the number of partitions. The available memory partition is referred to as a hole. In a case, if the partition size is larger than the process size, then the excess memory cannot be used for another process, as the whole partition is dedicated to the process. This type of memory wastage is known as internal fragmentation. However, if there is sufficient available memory for allocation to the process, but in the form of smaller non-contiguous partition, a process cannot be accommodated in it. This situation is known as external fragmentation. Fixed size partitioning suffers from both internal and external fragmentation [3-4].

Another form of allocation of memory is variable sized partitioning, where no partition is of fixed size. Operating system keeps track of the partitions which are available (set of holes) and which are employed. Initially, the entire memory space is set as available. When a process requires memory, operating system searches for a hole that is large enough to accommodate the process. If the hole is larger than the requirement of the process, the hole is then split into two parts and allocates one to the process according to its requirement and returns back the another one to the set of holes. When the process terminates, the memory is set free and it belongs to the set of holes. At any given time, the memory incorporates the set of holes sprinkled all over the memory. Two adjacent holes are merged to form a single large hole. This procedure is an analogy of dynamic memory allocation. Variable sized partitioning does not suffer from internal fragmentation. Still, it may suffer from external fragmentation. To overcome this complication, compaction can be done to rearrange the memory space so that all free memory partitions come together to form a large hole. However, this procedure is time consuming and needs dynamic reallocation ability [3-4].

MEMORY ALLOCATION ALGORITHMS

Memory allocation strategy has an objective to have an efficient memory management in order to minimize any kind of memory wastages in the form of fragmentation. Some of the most commonly used strategies are first fit, next fit, best bit and worst fit. Each strategy works distinctly and has its own advantages and disadvantages.

FIRST FIT

In this strategy, memory is scanned from the beginning and searched for the hole that is large enough to accommodate the process. As soon as the first hole is identified, it is allocated to the process. This strategy is less time consuming as it searches only for the first hole that has the size greater than or equal to the requirement of the process, not for the entire set of holes. The degree of internal fragmentation depends on the difference of the size between the hole and the process. If the process size is much smaller than the hole, then due to internal fragmentation, considerable amount of memory will be wasted [5]. An algorithm for first fit strategy is given below.

```
1. Start sequentially selecting each unvisited page until all are visited.
2. Search for a memory hole from the beginning.
3. i. If size-of-current-process < = size-of-hole
   Then, allocate the current hole to the current process.
   ii. Else
a. If there are unvisited holes, search for further memory holes and go to step 3.
b. Else mark the process as not satisfied and go to step 1 [6].
```

BEST FIT

In this strategy, memory is scanned from the beginning and searched for the set of holes that have the size equal to or greater than the size of process. The smallest hole that is big enough to accommodate the process is allocated. This strategy results in much better memory utilization than first fit. However, due to searching for required set of holes, it becomes a time consuming procedure. But, this effort pays in lesser internal fragmentation than first fit [7].

An algorithm for best fit strategy is given below.

```
1. Find the entire set of holes from the beginning of the memory.
2. Start sequentially selecting each unvisited process until all are visited.
3. Select the hole that has minimum size among the set of holes
4. i. If  the selected hole => size-of-current-process
a. Then, allocate the current hole to the current process
b. Remove the current hole from the set of holes and go to step 2.
   ii. Else
a. If set of holes is not empty, select the next hole of minimum size and go to step 4.
b. Else mark the process as not satisfied and go to step 2 [6].
```

WORST FIT

In this strategy, memory is scanned from the beginning and searched for the entire set of holes. Now, the hole that has the largest size among them is allocated to the process. This strategy generates the largest leftover hole, which may be sufficient enough to accommodate another process. Due to this, degree of internal fragmentation is much higher than first fit and best fit strategies that result in very poor memory management. Also, this procedure becomes time consuming as it has to search the largest hole among the set of holes [8].

An algorithm for worst fit strategy is given below.

```
1. Find the entire set of holes from the beginning of the memory.
2. Start sequentially selecting each unvisited process until all are visited.
```

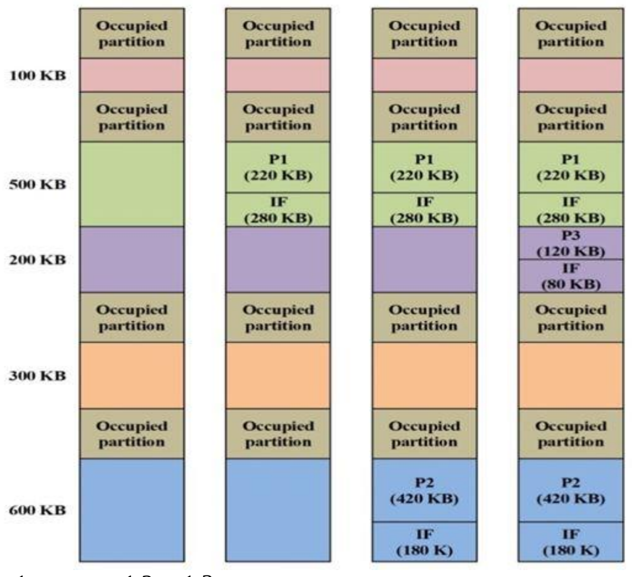
3. Select the hole that has maximum size among the set of holes
4. i. If the selected hole => size-of-current-process
a. Then, allocate the current hole to the current process,
b. Remove the current hole from the set of holes and go to step 2.
   ii. Else, mark the process as not satisfied and go to step 2 [6].

IMPLEMENTATION

To implement the first fit, best fit and worst fit strategies I fixed sized and variable sized partition, let us consider five memory partitions or holes of 100 KB, 500KB, 200 KB, 300 KB AND 600KB (in order) and we have to place process P1 of 220 KB, P2 of 420 KB, P3 of 120 KB andP4 of 330 KB (in order).The memory partitions are shown in the Fig 1.

First Fit Algorithm in Fixed Sized Partition

Now, to illustrate the first fit algorithm, process P1, P2, P3 and P4 will be considered sequentially. Initially, process P1 is considered. It has the size of 220 KB. First memory partition has 100 KB. So, it cannot be allocated to P1.Then, second partition is considered that has the capacity of 500 KB. Now, this partition can be allocated to P1. However, an internal fragmentation (IF) of 280 KB occurs due to this allocation. It is shown in Fig 1.1. For process P2 of 420 KB, first partition of 100 KB cannot be considered. Second partition is already occupied by P1. Third and fourth partition of 200 KB and 300 KB respectively are smaller than P2. Fifth and last partition of 600 KB can be allocated to P2. That results in an internal fragmentation (IF) of 180 KB. It is shown in the Fig 1.2 below. For process P3 of size 120 KB, first partition of 100 KB cannot be considered. Second partition is already occupied by P1. Third partition of 200 KB can be allocated to P3 with an internal fragmentation (IF) of 80 KB. It is shown in the Fig 1.3.For Process P4 of 330 KB, first partition cannot be allocated. Second and third partition is occupied by P1 and P3 respectively. Fourth partition of 300 KB cannot accommodate P4. Fifth partition is occupied by P2. In this case, combination of size of first and fourth partition becomes 400 KB, However, due to non-contiguous in nature of partitions; P4 cannot be allocated any memory partition. That means, an external fragmentation (EF) of 330 KB occurs. The whole procedure of first fit algorithm in fixed sized partition is shown in Diagram 1.



1. 1 1.2 1.3

Fig 1: Implementation of first fit algorithm in fixed sized partition

Best Fit Algorithm in Fixed Sized Partition

Initially, for the process P1 of size 220 KB, the smallest partition sufficient enough to accommodate P1 is the fourth partition of size 300 KB. After allocating P1, internal fragmentation of 80 KB occurs (vide Fig 2.1). For process P2 of size 420 KB, the smallest partition sufficient for allocation will be second partition of size 500 KB. This allocation raises an internal fragmentation of 80 KB (vide Fig 2.2). For process P3 of size 120 KB, third partition of size 200 KB can be allocated causing an internal fragmentation of 80 KB (vide Fig 2.3). For process P4 of size 330 KB, the smallest partition sufficient enough to accommodate is the second partition which is already occupied by P2. So, the next smallest but big enough to accommodate P4 is the fifth partition of 600 KB. This allocation suffers from internal fragmentation of 270 KB. It is shown in Fig 2.4Diagrammatic illustration of best fit algorithm in fixed sized partition is shown in Diagram 2.

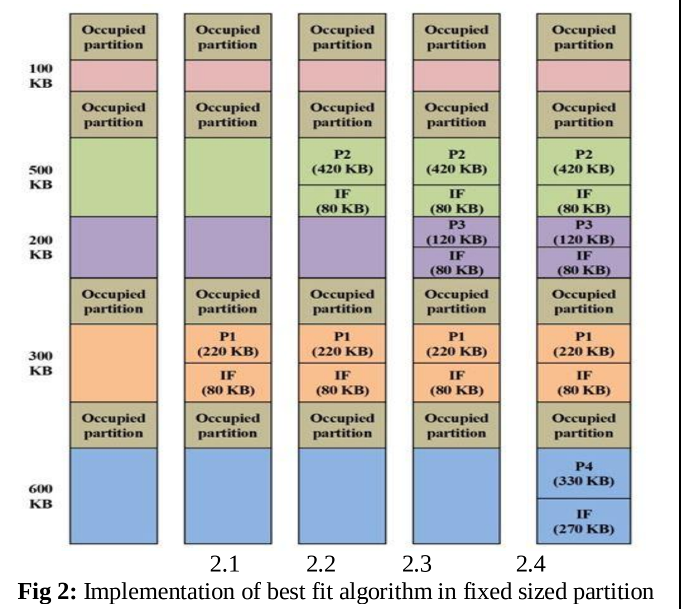


Fig 2: Implementation of best fit algorithm in fixed sized partition

***Worst Fit Algorithm in Fixed Sized Partition*:** Initially for process P1 of size 220 KB, the largest partition, i.e., the fifth partition of 600 KB is allocated. Due to this allocation, a significant amount of internal fragmentation of 380 KB occurs. It is shown in Fig 3.1. Now, for process P2 of size 420 KB, the next largest partition, that is, the second partition of 500 KB is allocated for which internal fragmentation of 80 KB occurs(vide Fig 3.2). For process P3 of 120 KB, the next largest partition that is the fourth partition of 300 KB is allocated. That result in internal fragmentation of 180 KB (vide Fig 3.3). For process P4 of size 330 KB, there is no partition available that can accommodate P4. In this case, first and third partitions are free, but not contiguous. Also their combined size does not exceed or equal the size of P4. Therefore, it cannot be considered as external fragmentation. Hence, P4 cannot be allocated. Diagrammatic illustration of best fit algorithm in fixed sized partition is shown in Diagram 3.

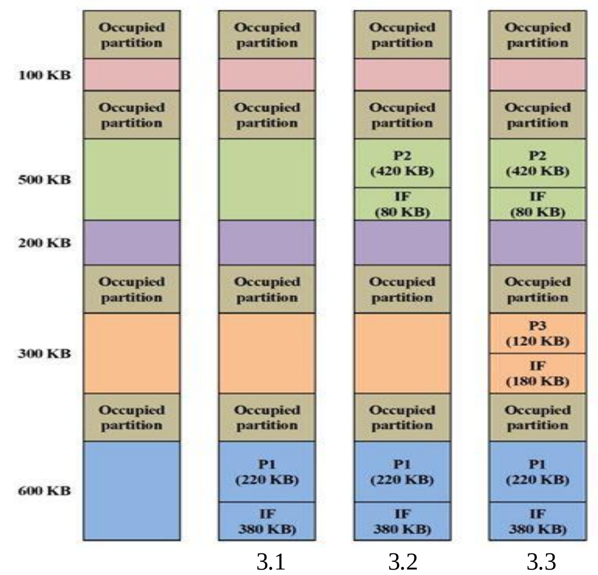


Fig 3: Implementation of worst fit algorithm in fixed sized partition

First Fit Algorithm in Variable Sized Partition

We know that variable sized partition has no definite partition size. Its diagrammatic representation is illustrated in Fig 2. At first, for process P1 of size 220 KB, first unoccupied partition of size 100 KB cannot accommodate P1. Second unoccupied partition of size 700 KB can be allocated to P1 and remaining 480 KB is treated as unoccupied partition (vide Fig 4.1). Now, for process P2 of size 420 KB, first partition of 100 KB cannot be allocated. However, second partition of 480 KB can be allocated to P2. Remaining 60 KB is treated as free partition (vide Fig 4.2). For process P3 of size 120 KB, first and newly created second hole of size 100 KB and 60 KB respectively cannot be allocated. Third free partition of 300 KB can be allocated to P3 that creates a new unoccupied partition of 180 KB (vide Fig 4.3). For process P4 of size 330 KB, first, second and third free partition of size 100 KB, 60 KB and 180 KB respectively cannot be allocated due to smaller size. However, last unoccupied partition of 600 KB can be allocated to P4 of size 330 KB and leftover of 270 KB is treated as unoccupied partition (vide Fig 4.4). The whole procedure is illustrated in Diagram 4.

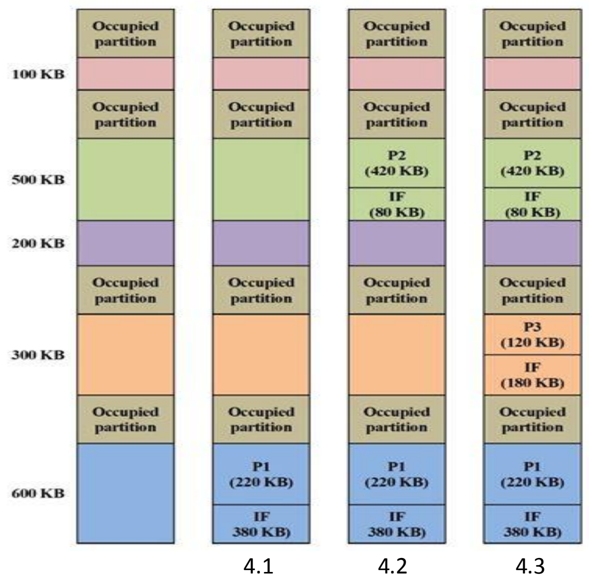


Fig 4: Implementation of first fit algorithm in variable sized partition

Best Fit Algorithm in Variable Sized Partition

Initially, for process P1 of size 220 KB, the smallest unoccupied partition big enough to accommodate which is third partition of 300 KB is allocated to P1. That result in a new unoccupied partition of 80 KB (vide Fig 5.1). For process of size 420 KB, the need for allocation in this algorithm can be satisfied by fourth partition of 600 KB. Remaining 180 KB is treated as new unoccupied partition (vide Fig 5.2). Now, for process P3 of size 120 KB, newly created partition of 180 KB can be allocated and 60 KB new free partition is created (vide Fig 5.3). For process P4 of size 330 KB, second unoccupied partition of 700 KB can be allocated and remaining 370 KB is treated as new unoccupied partition(vide Fig 5.4). This whole procedure is shown in Diagram 5.

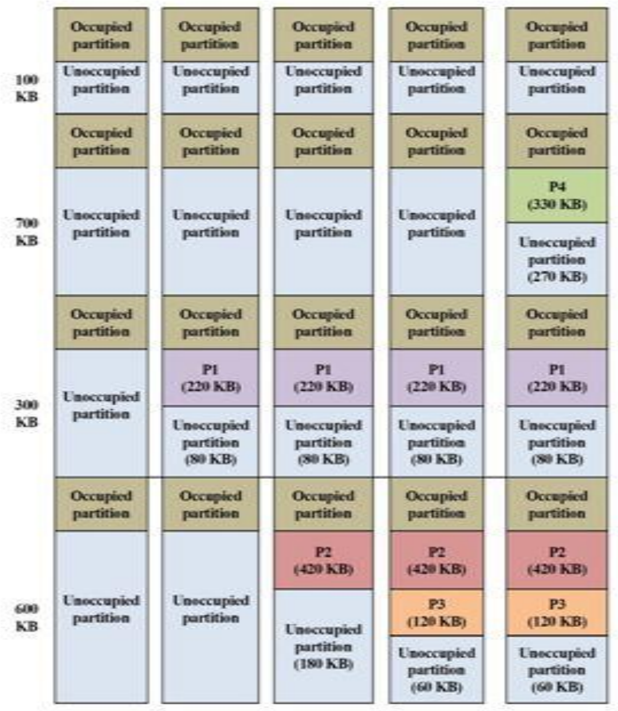
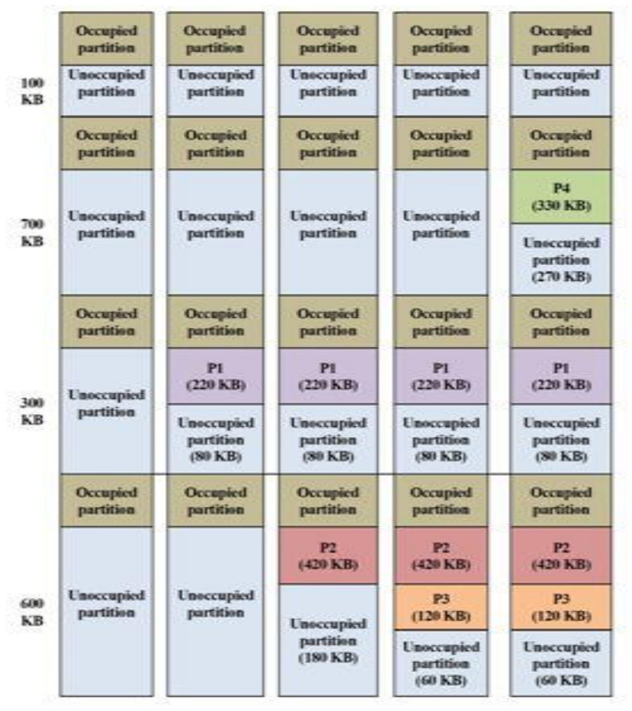


Fig 5: Implementation of best fit algorithm in variable sized partition

Worst Fit Algorithm In Variable Sized Partition

In this algorithm, for process P1 of size 220 KB, the largest unoccupied partition, i.e. , second partition of 700 KB is allocated. Remaining 480 KB is treated as new unoccupied partition (vide Fig 6.1). For process P2 of size 420 KB, the largest free partition will be the fourth partition of 600 KB. After allocation, new unoccupied partition of 180 KB is created (vide Fig 6.2). For process P3 of size 120 KB, the largest available partition is second partition of 480 KB and after allocation, remaining 360 KB is treated as free partition(vide Fig 6.3).For process P4 of size 330 KB, the largest free partition is newly created second partition of 360 KB. After allocation, unoccupied 30 KB is treated as new free partition (vide Fig 6.4). The whole procedure is shown in Diagram 6.


100 KB
700 KB
300 KB
600 KB
Occupied partition
Unoccupied partition
P1 (220 KB)
Unoccupied partition (80 KB)
P2 (420 KB)
Unoccupied partition (180 KB)
P3 (120 KB)
Unoccupied partition (60 KB)
P4 (330 KB)
Unoccupied partition (270 KB)

6. 1 6.2 6.3 6.4

Fig 6: Implementation of worst fit algorithm in variable sized partition

RESULT ANALYSIS

From the above diagrams and figures, it is observed that in case of variable sized partition, the entire three algorithms do not contribute to external fragmentation. Hence, its memory utilization for all three algorithms = $\frac{1090}{1700} \times 100\ \% = 64.12\ \%$ and process allocation is 100%.

Table 1. Internal fragmentation and memory utilization in fixed sized partition

Algorithms	Total memory allocated (In KB)	Total size of Process allocation(In KB)	% of process allocation	Internal fragmentation(In KB)	% of internal fragmentation
First fit	500 + 600 + 200 = 1300	220 + 420 + 120 = 760	$\frac{760}{1090} \times 100\ \%$ = 69.72 %	280 + 180 + 80 = 540	$\frac{540}{1700} \times 100\ \%$ = 31.76 %.
Best fit	300 + 500 + 200 + 600 = 1600	220 + 420 + 120 + 330 = 1090	$\frac{1090}{1090} \times 100\ \%$ = 100%	80 + 80 + 80 + 270 = 510	$\frac{510}{1700} \times 100\ \%$ = 30.00 %.
Worst fit	500 + 600 + 200 = 1300	220 + 420 + 120 = 760	$\frac{760}{1090} \times 100\ \%$ = 69.72 %	380 + 80 + 180 = 640	$\frac{640}{1700} \times 100\ \%$ =37.65 %

Now, a table has been constructed to present the internal fragmentation and memory utilization in fixed sized partition for all three algorithms considering initial unoccupied memory size is 1700 KB.From the Table 1, it is observed that best fit algorithm has the minimum internal fragmentation of 30 % with 100% process allocation. Its memory utilization is $\frac{760}{1300}$ = 58.46%. Worst fit algorithm has the maximum internal fragmentation of 37.65% with only 69.72% of process allocation. Its memory utilization is $\frac{1090}{1600}$ = 68.13%.First fit algorithm has the internal fragmentation of 31.76% with only 69.72 % of process allocation. Its memory utilization is $\frac{760}{1300}$ = 58.46%.

CONCLUSION

In fixed sized partition, internal fragmentation may or may not occur, but external fragmentation does not occur, because leftover of a partition cannot be allocated to other process. However, in case of variable sized partition, internal fragmentation does not takes place, because leftover of a partition can be allocated to other process, but external fragmentation may have a chance to happen, which may be solved using compaction. Analysis of the three algorithms: first fit, best fit and worst fit shows that best fit appears to be the most efficient algorithm among three algorithms in fixed sized partition with minimum internal fragmentation. However, worst fit may not be the worst performer in variable sized partition. An improvement of first fit algorithm may be the next fit algorithm, where instead of searching the first unoccupied partition from the beginning; it searches in the set of holes to find the first unoccupied partition. That results in less amount of time consumption. However, its performance is equivalent to first fit algorithm. In future, there may be dynamic memory allocation with combination of fixed sized and variable sized partition according to problem domain to improve the efficiency of the memory allocation.

REFERENCES

[1] P. K. S, P. Sinha., “ Foundation Of Computing” Third Edition, September-2015.
[2] “Implementation Of Contiguous Memory Management Techniques” , Available At https://www.geeksforgeeks.org/implementation-of-contiguous-memory-management-techniques/
[3] A. Silberschatz, P.B. Galvin, G. Gange, Operating System Concepts. Wiley India Edition, 8th Edition, 2010.
[4] W. Stallings, Operating System Internals Design And Principles, Prentice Hall, 7th Edition, 2012.
[5] “First-Fit Allocation In Operating System” Available At https://www.geeksforgeeks.org/first-fit-allocation-in-operating-systems/
[6] L. W. Htun, M. M. M. Kay, A. A. Cho.,“Analysis Of Allocation Algorithms In Memory Management” IJTSRD, Vol. 3, No. 5, August 2019.
[7] “Best-Fit Allocation In Operating System” Available At https://www.geeksforgeekss.org/best-fit-allocation-in-operating-system/?ref=rp
[8] “Worst-Fit Allocation In Operating System” Available At https://www.geeksforgeeks.org/worst-fit-allocation-in-operating-system/?ref=rp
[9] M. A. Awais, “Memory Management: Challenges And Techniques For Traditional Memory Allocation Algorithms In Relation With Today's Real Time Needs”, IJMST. Vol. 7, No. 3, March 2016 Bpb Publication, 1982.
[10] L. G. Kabari, T. S. Gogo, “Efficiency Of Memory Allocation Algorithms Using Mathematical Model”, Ijeert, Valume 3, Issue 9.
[11] “Fragmentation (Computing)”, 2020. Available At https://en.wikipedia.org/wiki/ fragmentation_(computing)

[12] “Contiguous Memory Allocation | Static Partitioning” Available At https://www.gatevidyalay .com /contiguous-memory-allocation-static-partitioning/