International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 3 Issue 5, August 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

# Analysis of Allocation Algorithms in Memory Management

**Lae Wah Htun[1], Moh Moh Myint Kay[1], Aye Aye Cho[2]**

[1]Assistant Lecturer, [2]Associate Professor
[1,2]University of Computer Studies, Hinthada, Myanmar

***How to cite this paper:*** Lae Wah Htun | Moh Moh Myint Kay | Aye Aye Cho "Analysis of Allocation Algorithms in Memory Management" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-5, August 2019, pp.1985-1987, **https://doi.org/10.31142/ijtsrd26731**

IJTSRD26731



**ABSTRACT**

Memory management is the process of controlling and coordinating computer memory, assigning portions called blocks to various running programs to optimize overall system performance and also known as memory allocation. Placement algorithms are implemented to determine the slot that can be allocated process amongst the available ones in the partitioned memory. Memory slots allocated to processes might be too big when using the existing placement algorithms hence losing a lot of space due to internal fragmentation. In dynamic partitioning, external fragmentation occurs when there is a sufficient amount of space in the memory to satisfy the memory request of a process but the process's memory request cannot be satisfied as the memory available is in a non-contiguous manner. This paper describes how to resolve external fragmentation using three allocation algorithms. These algorithms are First-fit, Best-fit and Worst-fit. We will present the implementation of three algorithms and compare their performance on generated virtual trace.

***KEYWORDS:*** *Best-fit, First-fit, Worst-fit, Performance, Memory Management*

## 1. Introduction

Memory Management is the function responsible for allocating and managing computer's main Memory. Different memory allocation algorithms have been devised to organize memory efficiently. Allocation algorithms are implemented to determine the slot that can be allocated a process amongst the available ones in the partitioned memory block. Allocating a single contiguous section of memory to each process is the most primitive method of memory management, usually called partitioning.

In partitioning, the simplest partitioning method is dividing memory into several fixed-sized partitions in advance, called fixed partitioning. Fixed partitioning is the oldest and simplest technique used to put more than one processes in the main memory. In fixed partitioning, there are two methods: equal-sized partitioning and unequal-sized partitioning. In equal-sized partitioning, any process whose size is less than or equal to the partition size can be loaded into any available partition. Fixed size partitions suffer from two types of problems; they are overlays and internal fragmentation.

If a process is larger than the size of the partition then it suffers from overlaying problem in which only required information will be kept in memory. Overlays are extremely complex and time consuming task. When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated causing internal fragmentation. It occurs when fixed sized memory blocks are allocated to the processes.

In multiple queues, each process is assigned to the smallest partition in which it fits and minimizes the internal fragmentation problem. In single queue, the process is assigned to the smallest available partition and the level of multiprogramming is increased the size of each block in fixed partition is varied where processes are assigned to the blocks where it fits exactly: in other words, processes may be queued to use the best available partition. In the unequal size partition compared to equal size partition, memory wastage is minimized, and may not give best throughput as some partitions may be unused. The unequal size partitions use two types of queues where processes are assigned to memory blocks. They are multiple queue and single queue. In multiple queues, each process is assigned to the smallest partition in which it fits and minimizes the internal fragmentation problem. In single queue, the process is assigned to the smallest available partition and the level of multiprogramming is increased. If process loaded is much smaller than any partition either equal or unequal, then it suffers from internal fragmentation in which memory is not used efficiently. To overcome this problem, an approach known as dynamic partitioning was developed. Partitioning may be done dynamically, called dynamic partitioning. In dynamic partitioning, external fragmentation occurs when there is a sufficient amount of space in the memory to satisfy the memory request of a process but the process's memory request cannot be satisfied as the memory available is in a non-contiguous manner. A method for overcoming external fragmentation is compaction. The difficulty with compaction is that it is a time-consuming and requires relocation capability. Therefore, different strategies may be taken as to how space is allocated to processes: the common placement algorithms are Firs-fit, Best-fit and Worst-fit.

## 2. Background of Theory

Modern operating systems provide efficient memory management and still research is being conduct to improve

the way the memory is allocated for applications because the main problem faces by memory allocation algorithms is to efficiently allocating the demanded memory blocks to the demanding applications with the minimum response time along with minimum memory loss in the shape of traditional memory loss problem called the fragmentation of memory. We will use the placement algorithm to efficiently allocate the processes in the main memory.

### A. First-fit

In First-fit, scan the memory from the beginning and allocate the first available block that is large enough. It is one of the fastest algorithms as it would search only as little as possible. But, the remaining unused memory areas left after allocations become waste if it is too smaller. Thus request for large memory requirement cannot be accomplished. We will prove that the following problem, which algorithm makes the most efficient use of memory.

Implementation in First-fit:

1. Input memory blocks with size and processes with size.
2. Initialize all memory blocks as free.
3. Start by picking each process and check if it can be assigned to current block.
4. If size-of-process <= size-of-block if yes then assign and check for next process
5. If not then keep checking the further blocks.

For example: Given ten memory partitions of 500KB, 200KB, 800KB, 400KB, 100KB, 700KB, 300KB, 600KB, 1000KB, 900KB (in order), How would the First-fit, Best-fit, Worst- fit algorithms place processes of 212KB, 150KB, 375KB, 950KB, 350KB, 632KB, 400KB, 717KB, 811KB (in order)? Which algorithm makes the most efficient use of memory?

In First-fit:
212KB is put in 500KB partition
150KB is put in 288KB (new partition 288KB=500KB-212KB)
375KB is put in 800KB partition
950KB is put in 1000KB partition
350KB is put in 425KB (new partition 425KB=800KB-375KB
632KB is put in 700KB partition
400KB is put in 400KB partition
717KB is put in 900KB partition
811 must wait

| Process No. | Process Size | Block no. |
|---|---|---|
| 1 | 212 | 1 |
| 2 | 150 | 1 |
| 3 | 375 | 3 |
| 4 | 950 | 9 |
| 5 | 350 | 3 |
| 6 | 632 | 6 |
| 7 | 400 | 4 |
| 8 | 717 | 10 |
| 9 | 811 | Not Allocated |

Figure1. First-fit Output results

### B. Best-fit

In Best-fit, the entire list of blocks must be searched the closest in size to the request and allocate this block. Memory utilization is much better than First-fit as it searches the smallest free partition first available. But, it is slower and may even tend to fill up memory with tiny useless holes.

Implementation in Best-fit:

1. Input memory blocks with size and processes with size.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(blockSize[1]),blockSize[2],……..,blockSize[n] > processSize[current] it found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

Calculation with Best-fit, above the problem describes in First-fit
In Best-fit:
212KB is put in 300KB partition
150KB is put in 200KB partition
375KB is put in 400KB partition
950KB is put in 1000KB partition
350KB is put in 500KB partition
632KB is put in 700KB partition
400KB is put in 600KB partition
717KB is put in 800KB partition
811KB is put in 900KB partition

| Process No. | Process Size | Block no. |
|---|---|---|
| 1 | 212 | 7 |
| 2 | 150 | 2 |
| 3 | 375 | 4 |
| 4 | 950 | 9 |
| 5 | 350 | 1 |
| 6 | 632 | 6 |
| 7 | 400 | 8 |
| 8 | 717 | 3 |
| 9 | 811 | 10 |

Figure2. Best-fit Output results

### C. Worst-fit

In Worst-fit, the entire list of blocks must be searched the largest block and allocate this block. Reduce the rate of production of small gaps. In contrast, this strategy produces the largest leftover block, which may be big enough to hold another process. But, if a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

Implementation in Worst-fit:

1. Input memory blocks with size and processes with size.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max (blockSize[1]), blockSize[2],……., blockSize[n] > processSize[current] it found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

Calculation with Worst-fit, above the problem describes in First-fit
In Worst-fit:
212KB is put in 1000KB partition
150KB is put in 900KB partition
375KB is put in 800KB partition
950KB must wait
350KB is put in 788KB (new partition 788KB=1000KB-212KB)

632KB is put in 750KB (new partition 750KB=900KB-150KB)
400KB is put in 700KB partition
717KB must wait
811KB must wait

| Process No. | Process Size | Block no. |
|---|---|---|
| 1 | 212 | 9 |
| 2 | 150 | 10 |
| 3 | 375 | 3 |
| 4 | 950 | Not Allocated |
| 5 | 350 | 9 |
| 6 | 632 | 10 |
| 7 | 400 | 6 |
| 8 | 717 | Not Allocated |
| 9 | 811 | Not Allocated |

Figure3. Worst-fit Output results

## 3. Experimental Results

In this paper, we have been seen that Best-fit algorithm is the best among three placement algorithms. We are explained with a problem, how to calculate this algorithm. These algorithms cannot eliminate external fragmentation. To overcome this problem, we must use compaction.

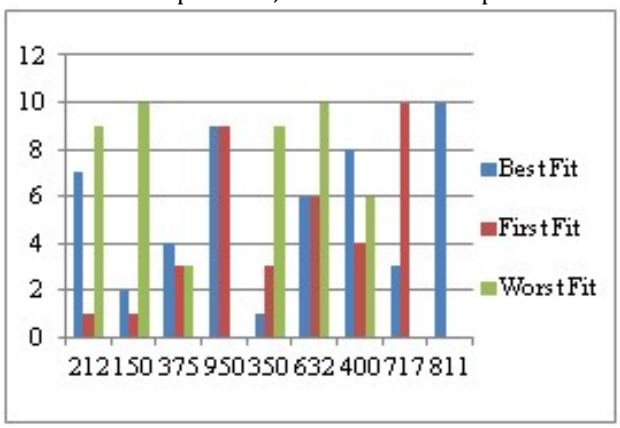


Figure4. Comparison of Three Allocation Algorithms for Performance

## 4. Conclusion

Main memory management is very important in operating system. Simulations have shown that both First-fit and Best-fit are better than Worst-fit in terms of decreasing both time and storage utilization. Neither First-fit nor Best-fit is clearly better in terms of storage utilization but First-fit is generally faster. Among three algorithms, Best-fit algorithm makes the most efficient use of memory. In this paper, we also proved that Best-fit algorithm is the best in memory utilization.

## References

[1] Rachael Chikorrie, Okuthe P. Kogeda, Manoj Lall: "An Optimized Main Memory Management Partitioning Placement Algorithm", Pan African conference on Science, computing and Telecommunications (PACT) Publisher, July 27-29, Kampala Uganda 2015.

[2] Abraham Silerschatz, Peter Beer Galvin, and Greg Gange, "Operating System Concepts", John Wiley & Sons, INC., January 1, 2002.

[3] William Stallings, "Operating System Internals and Design Principles", March 20, 2017.

[4] Ledisi G. Kabari, TamunoomieS. Gogo, "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

[5] Muhammand Abfullh Awais, "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.