Simulating and Analyzing Deadlock and Memory Allocation Strategies in Operating Systems

Alentajan, Jervie D.

Florida, Emanuel C.

Platon, Laiza Marie O.

Sison, Mark Anthony R.

Serapion, Pauline L.

CS110 & CS202

May 2025

https://github.com/paulnsrpn/OS_Final_Deadlock_MemorySim.git

1. Introduction

A deadlock, in the context of computer programming, is the situation wherein a number of processes cannot proceed, waiting for another process to release the resources. A set of processes is blocked because of this situation. It mainly focuses on the conditions namely mutual exclusion, resource holding, circular wait, and no preemption. To prevent deadlock situations from happening, memory allocation is important to improve the performance, stability, and scalability of the system being developed. It mainly affects and improves the performance of the system by reducing the time of memory allocation. Fast, dependable, and resource-conscious systems are made possible by memory allocation efficiency, which primarily serves to advance industry system development.

2. Methodology

2.1 Simulation process of Banker’s Algorithm

Simulating deadlock involves setting up a system where multiple processes compete for limited resources, potentially leading to a state where no process can proceed. This happens when four conditions: mutual exclusion, hold and wait, no preemption, and circular wait are all present. The simulation uses defined processes, resources, and matrices to track allocations and needs. Algorithms like the Banker's Algorithm help determine if the system remains in a safe state during execution. If all processes become blocked, the simulation identifies a deadlock and reports which processes are affected.

2.2 Memory Allocation Strategy Implementation

Memory allocation strategies like First Fit, Best Fit, and Worst Fit simulate how processes are assigned to memory blocks based on different selection rules. First Fit assigns a process to the first sufficiently large block it encounters, offering speed but potentially causing fragmentation. Best Fit chooses the smallest suitable block, maximizing space efficiency initially but often leaving behind unusable fragments. Worst Fit assigns processes to the largest available block to reduce fragmentation, though it can lead to inefficient use of memory. In your simulation, Best Fit achieved a 100%

success rate, while First Fit and Worst Fit each allocated memory to three out of four processes, resulting in a 75% success rate.

3. Results & Analysis

3.1 Memory Strategies

--- Memory Allocation Table ---

Process	Size	First Fit	Best Fit	Worst Fit
P0	212 KB	Block 1	Block 3	Block 4
P1	417 KB	Block 4	Block 1	Block 1
P2	112 KB	Block 1	Block 2	Block 4
P3	426 KB	Not Allocated	Block 4	Not Allocated

Figure 1. Memory Allocation Table

Figure 1 illustrates how First Fit, Best Fit, and Worst Fit allocation strategies assign memory blocks to processes based on their size requirements. The memory blocks available are: B0 = 100 KB, B1 = 500 KB, B2 = 200 KB, B3 = 300 KB, and B4 = 600 KB. The processes include P0 (212 KB), P1 (417 KB), P2 (112 KB), and P3 (426 KB). In First Fit, the algorithm allocates the first block large enough for each process. P0 is placed in B1, P1 in B4, and P2 also in B1 (assuming independent evaluation). P3 is not allocated due to insufficient space. Best Fit allocates each process to the smallest suitable block, resulting in P0 in B3, P1 in B1, P2 in B2, and P3 in B4—maximizing space efficiency. Worst Fit uses the largest block for each process, placing P0 in B4, P1 in B1, P2 again in B4, and leaving P3 unallocated due to lack of fitting blocks.

These results highlight trade-offs: First Fit is faster but less space-efficient, Best Fit minimizes waste, and Worst Fit attempts to preserve smaller blocks but leads to higher fragmentation.

3.2 Discuss time efficiency and fragmentation

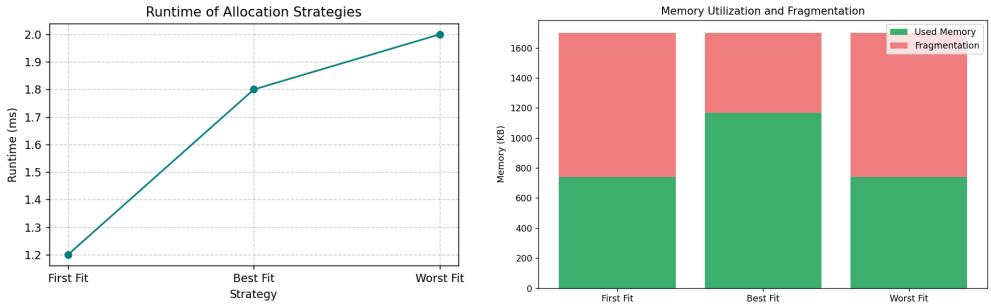

Figure 2. Runtime of Allocation Strategies, Memory Utilization and Fragmentation

Figure 2 compares memory allocation strategies based on runtime efficiency and fragmentation. The line graph shows that First Fit is the fastest, with a runtime of around 1.2 ms, due to its straightforward approach of allocating the first suitable block. Best Fit and Worst Fit take longer—1.8 ms and 2.0 ms respectively—because they scan all blocks to find the most suitable match. The bar graph illustrates memory utilization. Best Fit achieves the highest memory efficiency with minimal fragmentation, as it allocates the smallest fitting block to each process. In contrast, First Fit and especially Worst Fit lead to more fragmentation. Worst Fit leaves large unused memory gaps by prioritizing the largest blocks, making reuse less likely.

In summary, First Fit is the most time-efficient but has moderate fragmentation, Best Fit offers optimal memory use with a slight performance cost, and Worst Fit performs worst in both runtime and memory utilization. These trade-offs are critical when choosing an allocation strategy in practice.

4. Conclusion

4.1 Insights gained

Through the simulation and analysis of deadlock scenarios and memory allocation strategies, several key insights were revealed. First, understanding the conditions that lead to deadlocks—mutual exclusion, hold and wait, no preemption, and circular wait—provides a foundational framework for designing safer and more efficient operating systems. The implementation of the Banker's Algorithm effectively

demonstrated how to maintain system safety by pre-checking for potential deadlocks before granting resource requests.

On the memory allocation front, it became clear that the choice of strategy significantly impacts both system performance and memory utilization. First Fit demonstrated speed and simplicity but suffered from moderate fragmentation. Best Fit offered the most efficient memory use with minimal wasted space, though at the cost of slightly higher runtime. Worst Fit, while attempting to reduce future fragmentation, resulted in poor memory utilization and longer execution time.

4.2 Best-performing strategy

Among the three memory allocation strategies—First Fit, Best Fit, and Worst Fit—Best Fit emerged as the most effective overall. It successfully allocated memory to all four processes, achieving a 100% success rate, and demonstrated the highest memory utilization with minimal fragmentation. Although it required more time than First Fit, its ability to maximize memory efficiency makes it the best choice when space optimization is a priority.

Ultimately, this study emphasizes the importance of choosing appropriate memory management techniques based on the specific goals of the system, whether that be speed, memory efficiency, or a balance of both.

5. Contribution Table

Member Names	Contribution	% of Total Work
Alentajan, Jervie D.	Visualization Developer	20%
Florida, Emanuel C.	Tester and Analyst	20%
Platon, Laiza Marie O.	Documentation Specialist	20%
Serapion, Pauline L.	Lead Programmer	20%
Sison, Mark Anthony R.	Document Finalization	20%