Simulating and Analyzing Deadlock and Memory Allocation Strategies in Operating Systems

In partial fulfillment of the requirements for the courses

CS202: Parallel and Distributed Programming

CS101: Operating System

Submitted by:

Azarcon, Christian Jull G.

Diano, Maxzine Arly V.

Perin, Mariel B.

Rafer, Britney P.

Sotelo, Shantelle C.

Mr. Ronnie Chu

Professor

May 2025

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

INTRODUCTION

System performance and stability have never been more important in today's rapidly changing digital environment. Some of the critical variables that influence how a system functions smoothly are how deadlocks are handled and how the memory allocation is managed. Deadlocks occur when two or more processes get stuck because each is waiting for the other to release and provide a needed resource causing problems in multitasking environments like operating systems or the cloud platforms, where the smooth tasks execution is critical. To prevent deadlock, developers use different strategies such as resource ordering, timeouts, or detection algorithms that allow the system to recover before things could go awry.

On the other hand, efficient memory allocation is essential for keeping systems reliable and efficient, especially with dealing with the modern applications that utilize large numbers of data or serve many users at once. Slowdowns, crashes, and wasted system resources are all possible consequences of poor memory management. Systems that use techniques such as dynamic memory allocation and memory recycling can perform much better, conserve energy, and handle more jobs seamlessly.

METHODOLOGY

To simulate deadlock, we created a simple system with several processes and limited resources. Each of the processes had specific resource needs and could request more while maintaining some. If the resources were available, they were provided; otherwise, the process waited. We monitored the simulation to check if the necessary conditions for deadlock were all present, namely the mutual exclusion, hold and wait, no preemption and circular wait. To detect circular wait, we used a resource allocation graph and basic cycle detection method. In other tests, we attempted simple solutions, like terminating a process or forcing it to release resources.

For memory allocation, we implemented First Fit, Best Fit, and Worst Fit strategies using a fixed memory block. Each time a process requested a memory, the strategy in utilizing determined how the memory was assigned. When the overall process is finished, memory was released and adjacent free blocks were merged to reduce fragmentation. Us researchers tracked things like memory usage, response time, and wasted space to compare how effective each strategy was.

RESULTS AND ANALYSIS

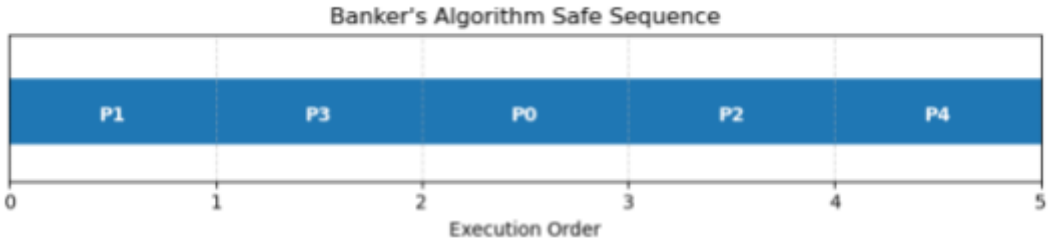

Figure 1. Bank Algorithm Sequence Chart

The chart shows the safe sequence of processes using the Banker's Algorithm, which makes sure each process gets the resources it needs without blocking others. The order P1, P3, P0, P2, then P4 is like a line where they wait their turn safely. As each process finishes, it gives back resources so the next one can run.

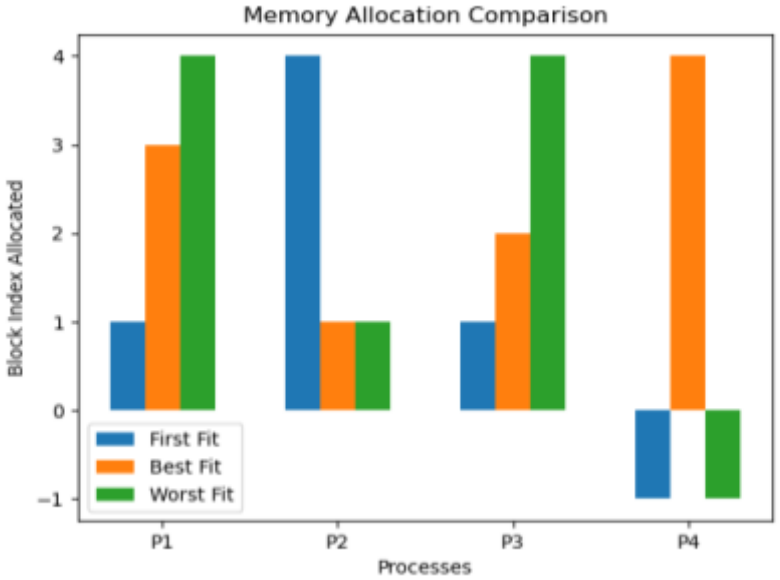

Figure 2. Process Allocation Chart

The chart shows that all the three strategies failed to allocate memory to the last process, **First Fit** is the best for its speed but suffers from higher fragmentation. **Best Fit** is quite slower but generally manages memory more efficiently, though it can still create unusable small gaps. **Worst Fit** selects the largest blocks in order to delay fragmentation, but this can result in inefficient use of large memory blocks.

CONCLUSION

To sum up, the results from the Banker's Algorithm show with full clarity that the system and all maintained processes are in a safe state as the processes can be executed in a bound sequential manner of: P1, P3, P0, P2, P4 without a deadlock situation occurring. The existence of this guaranteed order further denotes that the system is under optimal control in terms of resources and that every process has a definite way to finish as long as this sequence is followed. The results illustrate how useful resource management policies are in terms of avoiding system failures and enabling execution. The algorithm is useful in systems where several processes contend for a limited set of resources, thus becoming essential to the preservation of system stability and deadlock free operation.

The review of memory allocation techniques indicates that Best Fit offers the optimal efficiency for this case. Unlike First Fit and Worst Fit which, for some reason, left one of the processes (P4) without allocated memory, Best Fit completed all processes by appropriately using all available memory blocks. Thus, Best Fit achieved better outcomes by reducing allocation waste and improving system performance. Although First Fit provides faster execution time and Worst Fit appears to reduce fragmentation, they both failed in this instance. As a result, for systems that strive to maximize aligned resource utilization and successful memory allocation, based on data collected, Best Fit stands as the clearest and most accurate decision.

Contribution Table

NAME	CONTRIBUTION	% OF TOTAL WORK
AZARCON, Christian Jull B.	Researcher Programming & Algorithm Implementation Testing Support Final Polishing	20%
DIANO, Maxzine Arly V.	Researcher Documentation & Report Writing Video Editing Final Polishing	20%
PERIN, Mariel B.	Researcher Programming & Algorithm Implementation Testing Support Final Polishing	20%
RAFER, Britney P.	Researcher Programming & algorithm implementation Testing Support Final Polishing	20%
SOTELO, Shantelle C.	Researcher Documentation & Report Writing Video Editing Final Polishing	20%
	TOTAL:	**100%**