Grade: 65/65 A+

Deadlock: Characteristics and Handling Strategies

Alejandro Ricciardi

Colorado State University Global

CSC300: Operating Systems and Architecture

Joe Rangitsch

July 2, 2023

Deadlock: Characteristics and Handling Strategies

In the context of operating systems, deadlock is a critical issue that can occur in multiprocessing systems when two or more processes or threads are running simultaneously, and each process needs access to a resource that another process is using (Zimmer, 2016). Consequently, these processes block each other from accessing their needed resources thereby hindering each other's completion indefinitely or until one or more processes are terminated by the operating system or a user. Deadlocks have four characteristics or necessary conditions for the deadlock to occur. The necessary conditions are mutual exclusion, hold and wait, no preemption, and circular wait. Furthermore, deadlocks can have severe consequences on the system's performance, and the availability and use of resources, and potentially lead to system crashes. (Prepbytes, 2023). Therefore, it is crucial for operating systems to implement strategies to prevent or avoid the occurrence of deadlocks. Two of those strategies are deadlock prevention and deadlock avoidance. Finally, eliminating the possibility of deadlock is challenging, and comes with the overhead of managing the possibility of it, but it is crucial for the performance and stability of multiprocessing systems. Therefore, it is essential to strike a balance between the implementation of deadlock prevention and avoidance strategies and the overhead cost associated with these strategies.

Deadlock Characteristics

In 1971, computer scientist Edward G. Coffman described the four necessary conditions for a deadlock to occur, now known as the Coffman conditions or the deadlock characteristics (Zimmer, 2016). The four necessary conditions are mutual exclusion, hold and wait, no preemption, and circular wait. First is the mutual exclusion condition, which is an essential tool in multiprocessing systems, it impedes a resource from being accessed by more than one process

at a time. In a uniprocessor, processes running concurrently cannot overlap execution, they must be interleaved (Stallings, 2018). In other words, they cannot interrupt each other's execution. Therefore, mutual exclusion is implemented to prevent such interruptions. In multiprocessor systems, the principal purpose of mutual exclusion is to prevent race condition. "Race condition is a situation in which multiple threads or processes read and write a shared data item, and the final result depends on the relative timing of their execution" (Stallings, 2018, Chapter 5). If it is not prevented, it can lead to data corruption. Hence, mutual exclusion by preventing race condition is critical in maintaining data integrity. Nonetheless, mutual exclusion can result in some resources not being shared among processes.

Second is the hold and wait condition, which is the condition in which a process is holding a resource, that may or may not be needed by another process, and simultaneously waiting for another resource to be released by another process. Hold and wait may be implemented where processes require multiple resources to complete their tasks, it is also used to manage resources allocation between multiple concurrent processes. However, this approach may lead to starvation, a situation where a process or processes are perpetually denied access to a resource causing them to not complete and waste system resources.

Third is the no preemption condition, which prevents resources held by a process from being forcefully removed from its control. Instead, the process must voluntarily relinquish the resource (Silberschatz et al., 2018). The no preemption condition is useful in a read-writer situation where the reader process cannot be preempted from a resource by another read or writer process. Therefore, maintaining data integrity, but this approach is also problematic. The no preemption condition can lead to low-priority processes starvation and not completion, this can be caused by higher-priority processes perpetually holding resources.

The fourth is the circular wait condition, which occurs in a circular chain of processes where each process in the chain is waiting for a resource held by another process in the chain (Prepbytes, 2023). The circular wait is not something that is intentionally used or implemented. Instead, it can be described as a possible consequence of implementing the other three Coffman conditions. Furthermore, if all the other three conditions are also present, a deadlock can occur, resulting in an endless waiting loop. Figure 1 illustrates a two-process circular wait deadlock. Process 1 and Process 2 each hold resources B and A, respectively, and each requires the other's resources to complete their task. As a result, the two processes are preventing each other from completing and creating an infinite circular loop. Moreover, a deadlock critically affects the performance and stability of systems and potentially leads to system crashes.

Thus, in multiprocessing systems, the Coffman conditions are both necessary and useful for maintaining data integrity, preventing race condition, and where processes require multiple resources to complete their tasks. However, they have a major drawback: they can lead to a deadlock situation. Deadlocks can cause a system to become unresponsive, waste system resources as processes are stuck waiting for resources that they can never acquire, and potentially result in system crashes (Prepbytes, 2023). Thus, it is crucial to implement strategies to prevent or avoid the occurrence of deadlocks. Two of those strategies are deadlock prevention and deadlock avoidance.

Deadlock Prevention and Avoidance

There are many strategies for handling a deadlock situation, depending on the system's handling needs, some methods are better suited than others. Examples of deadlock handling strategies are deadlock prevention, deadlock avoidance, deadlock detection, and deadlock ignorance. In this essay, I will discuss two of the most commonly used strategies which are deadlock prevention and deadlock avoidance.

First, deadlock prevention prevents deadlock using the indirect and direct methods. The indirect method prevents deadlock by ensuring that one of the first three Coffman conditions does not occur. The direct method, on the other hand, prevents only the occurrents of the circular wait condition (Stallings, 2018). Both deadlock prevention methods guarantee the non-occurrence of deadlocks, but they come with trait-offs. The implementations of deadlock prevention methods are complex and come with significant computing overhead that may decrease system performance. Furthermore, the indirect method, by limiting access to resources may lead to underutilization of the system resources.

Second, the deadlock avoidance strategy is more flexible and allows more concurrency than the deadlock prevention strategy. Deadlock avoidance allows the first three Coffman conditions, but it also ensures that the point of deadlock is never reached by carefully managing resource allocation. This requires knowledge of future process requests, process execution must be unconstrained by any synchronization requirements, and a fixed number of resources to allocate (Stallings, 2018). These increase the system's overhead and complexity, which can negatively affect the system's overall performance and waste system resources.

Hence, both strategies are useful to handle the deadlock issue. However, they come with disadvantages, they increase systems overhead and complexity, and as a consequence, they may

reduce system performance and waste resources. Therefore, when implementing those strategies is crucial to strike a balance between the implementation of deadlock prevention and avoidance strategies and the overhead cost associated with these strategies.

Conclusion

In conclusion, deadlock is a critical issue in multiprocessing systems. The four necessary conditions for a deadlock to occur are mutual exclusion, hold and wait, no preemption, and circular wait. Furthermore, they can be necessary and useful for maintaining data integrity and for the optimal functioning of multiprocessing systems. However, they can lead to deadlock situations which critically affect the performance and stability of systems, and potentially lead to system crashes. Therefore, it is crucial to implement strategies to prevent or avoid the occurrence of deadlocks. Two commonly used strategies are deadlock prevention and deadlock avoidance. In simple terms, deadlock prevention limits access to resources or directly prevents the circular wait condition, to impede the occurrence of a deadlock. Deadlock avoidance, on the other hand, carefully manages resource allocation by having knowledge of future process requests, freeing process execution from any synchronization requirements, and having a fixed number of resources to allocate. However, those strategies have trait-offs, they increase systems overhead and complexity, and they may reduce system performance and waste resources. Finally, while it is critical to negate deadlocks, it is equally important to strike a balance between the implementation of strategies to handle the deadlock issue and the overhead and complexity associated with these strategies.

References

Prepbytes (2023, January 31). Deadlock in OS. *PrepBytes Blog*. https://www.prepbytes.com/blog/operating-system/deadlock-in-os/#:~:text=Deadlock%20can%20cause%20a%20system,cause%20the%20system%20to%20crash

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* [PDF]. Wiley. Retrieved from: https://os.ecci.ucr.ac.cr/slides/Abraham-Silberschatz-Operating-System-Concepts-10th-2018.pdf

Stallings, W. (2018). *Operating systems: Internals and design principles*. Pearson.

Zimmer, S. (2016, May 1). *Applied science: computer science.* Salem Press.