

**جــامـعـة الإسراء الخــاصــة**

**كـليـة تكـنولوجيـا المعلومـات**

**Isra University**

**Faculty of IT**

**First Semester 2021/2022**

|  |  |  |
| --- | --- | --- |
| Course Title: DBMS | Course No.: | HW No :2 |
| Submitted to: Dr. Maher Abuhamdeh | | |
| Student Number:  AD0039 | Student Name:  Feras Sameer Saleem | Date: |

**Wound Wait vs. Wait Die Algorithms**

**1. Introduction**

Database Management Systems (DBMS) play a crucial role in managing and organizing data in various applications. In a multi-user environment, concurrency control algorithms are essential to ensure that transactions can execute concurrently without causing data inconsistencies. However, certain concurrency control algorithms may exhibit starvation, a situation where a transaction is unable to make progress due to the continuous prioritization of other transactions. This report focuses on the analysis of starvation in the context of two common concurrency control algorithms: Wound Wait and Wait Die.

**2. Background**

**2.1 Concurrency Control**

Concurrency control in DBMS involves managing access to the database by multiple transactions simultaneously. This is achieved through the use of locking mechanisms and algorithms that ensure consistency and isolation among transactions.

**2.2 Starvation**

Starvation occurs when a transaction is continually denied access to a resource it needs to proceed, while other transactions are granted access. This can lead to a situation where a transaction is never able to complete, hindering the overall progress of the system.

**3. Wound Wait Algorithm**

The Wound Wait algorithm is a type of optimistic concurrency control. It allows a transaction to proceed with its execution, but if it detects a potential conflict with another transaction, it may wound (roll back) the conflicting transaction. This ensures that a higher-priority transaction is given precedence.

**3.1 Starvation in Wound Wait**

Wound Wait can lead to starvation when lower-priority transactions repeatedly wound higher-priority transactions. If a transaction with lower priority continually interferes with transactions of higher priority, those high-priority transactions may never be able to complete successfully, resulting in starvation.

**4. Wait Die Algorithm**

The Wait Die algorithm is a pessimistic concurrency control algorithm. It grants access to a transaction based on its priority, and if a lower-priority transaction requests a resource held by a higher-priority transaction, the lower-priority transaction is made to wait (if the higher-priority transaction is active) or be aborted (if the higher-priority transaction is waiting).

**4.1 Starvation in Wait Die**

Starvation in Wait Die occurs when a high-priority transaction is repeatedly aborted due to conflicts with low-priority transactions. If a high-priority transaction keeps getting aborted, it may be unable to make progress, resulting in starvation.

**5. Comparative Analysis**

Both Wound Wait and Wait Die algorithms aim to prioritize higher-priority transactions over lower-priority ones to ensure fairness and efficiency. However, the difference in their approach to resolving conflicts leads to variations in the occurrence of starvation.

**5.1 Factors Influencing Starvation**

The occurrence of starvation is influenced by factors such as transaction arrival rates, the nature of conflicts, and the distribution of transaction priorities. Analyzing these factors can provide insights into the conditions under which each algorithm is more prone to causing starvation.

**6. Conclusion**

In conclusion, Wound Wait and Wait Die are two widely used concurrency control algorithms in DBMS that exhibit distinct behaviors concerning starvation. Understanding the factors contributing to starvation in these algorithms is crucial for designing systems that balance fairness and efficiency in a multi-user environment. Further research and experimentation are recommended to explore strategies for mitigating starvation and improving the overall performance of database management systems.