**Concurrency Control Protocol**

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

* Lock based protocols
* Time stamp based protocols

Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

* **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
* **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

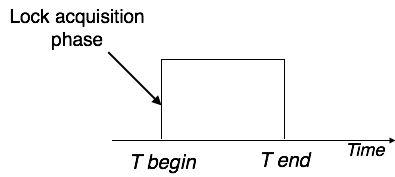
There are four types of lock protocols available −

Simplistic Lock Protocol

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

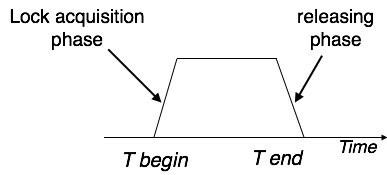
Pre-claiming Lock Protocol

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



Two-Phase Locking 2PL

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

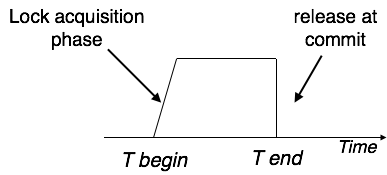


Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

**Timestamp Ordering Protocol**

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

* The timestamp of transaction Ti is denoted as TS(Ti).
* Read time-stamp of data-item X is denoted by R-timestamp(X).
* Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows −

* **If a transaction Ti issues a read(X) operation −**
  + If TS(Ti) < W-timestamp(X)
    - Operation rejected.
  + If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  + All data-item timestamps updated.
* **If a transaction Ti issues a write(X) operation −**
  + If TS(Ti) < R-timestamp(X)
    - Operation rejected.
  + If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  + Otherwise, operation executed.

Thomas' Write Rule

This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making Ti rolled back, the 'write' operation itself is ignored.

**In optimistic concurrency control techniques**, also known as validation or certification techniques, no checking is done while the transaction is executing. Several theoretical concurrency control methods are based on the validation technique. We will describe only one scheme here. In this scheme, updates in the trans-action are not applied directly to the database items until the transaction reaches its end. During transaction execution, all updates are applied to local copies of the data items that are kept for the transaction. At the end of transaction execution, a validation phase checks whether any of the transaction’s updates violate serializability. Certain information needed by the validation phase must be kept by the system. If serializability is not violated, the transaction is committed and the database is updated from the local copies; otherwise, the transaction is aborted and then restarted later.

There are three phases for this concurrency control protocol:

        Read phase. A transaction can read values of committed data items from the database. However, updates are applied only to local copies (versions) of the data items kept in the transaction workspace.

        Validation phase. Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.

        Write phase. If the validation phase is successful, the transaction updates are applied to the database; otherwise, the updates are discarded and the trans-action is restarted.

The idea behind optimistic concurrency control is to do all the checks at once; hence, transaction execution proceeds with a minimum of overhead until the validation phase is reached. If there is little interference among transactions, most will be validated successfully. However, if there is much interference, many transactions that execute to completion will have their results discarded and must be restarted later. Under these circumstances, optimistic techniques do not work well. The techniques are called optimistic because they assume that little interference will occur and hence that there is no need to do checking during transaction execution