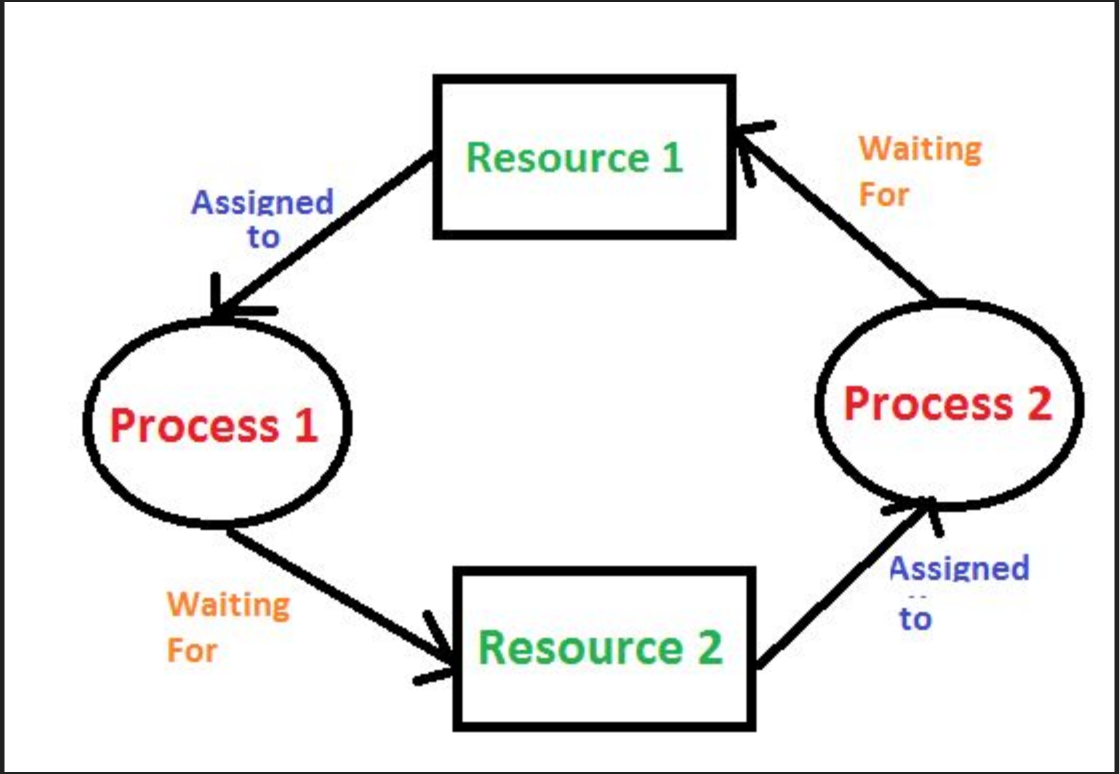
TWO PHASE LOCKING NON RESOURCE DEADLOCK & STARVATION

SONY K MARTIN
SREEDEV T
SREELAKSHMI S
SREENANDAN N S
STEVE GEORGE
SUHAIL SUBAIR

DEADLOCK

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.


Resource 1
Assigned to
Waiting For
Process 1
Process 2
Waiting For
Assigned to
Resource 2

REAL LIFE EXAMPLE OF DEADLOCK

“**No, you hang up first**” problem

Imagine a couple having a romantic phone conversation and when it times to hang up either of them wants to hang up first and they keep saying “No, you hang up first!”

CONDITIONS FOR DEADLOCK

->Mutual Exclusion

Two or more resources are non-shareable (Only one process can use at a time)

->Hold and Wait

A process is holding at least one resource and waiting for resources.

->No Preemption

A resource cannot be taken from a process unless the process releases the resource.

->Circular wait

A set of processes are waiting for each other in circular form.

DEALING WITH DEADLOCK

There are three ways to handle deadlock:

1. Deadlock prevention or avoidance
2. Deadlock detection and recovery
3. Ignore the problem all together

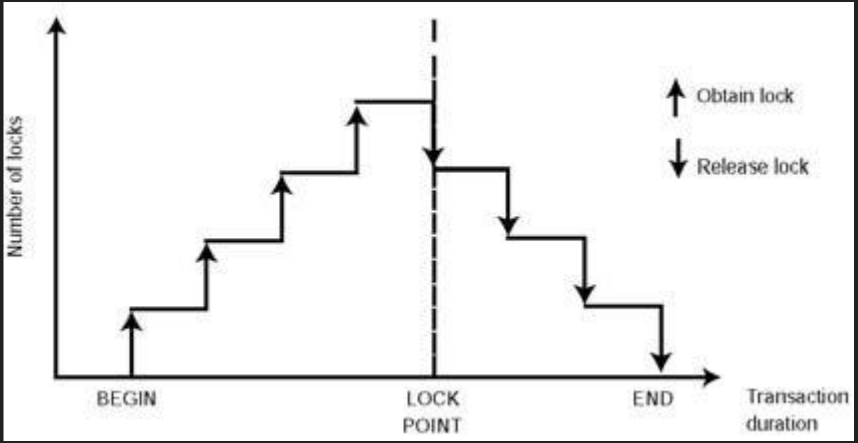
TWO PHASE LOCKING

A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.

1. **Growing Phase:** New locks on data items may be acquired but none can be released.
2. **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

Types of Locks

Shared lock(S): Read only access
Exclusive lock(X): Read & Write access



STATIC / CONSERVATIVE 2-PL

- Static 2-PL, this protocol requires the transaction to lock all the items it access before the Transaction begins execution by pre declaring its read-set and write-set.
- If any of the predeclared items needed cannot be locked, the transaction does not lock any of the items, instead, it waits until all the items are available for locking.
- Static 2-PL is deadlock free.
- However, it is difficult to use in practice because of the need to predeclare the read-set and the write-set which is not possible in many situations.

Types of 2-PL

Basic	The basic two phase locking allows release of lock at any time after all the locks have been obtained.
Strict	This requires that in addition to the lock being 2-Phase all Exclusive(X) locks held by the transaction be released until after the Transaction Commits.
Rigorous	This requires that in addition to the lock being 2-Phase all Exclusive(X) and Shared(S) locks held by the transaction be released until after the Transaction Commits.
Conservative	This protocol requires the transaction to lock all the items it access before the Transaction begins execution.

DEADLOCK IN 2-PL

	T1	T2
1	X(A)	
2		X(B)
3	X(B) {T1 will wait for T2 to unlock B}	
4		X(A) {T2 will wait for T1 to unlock A}

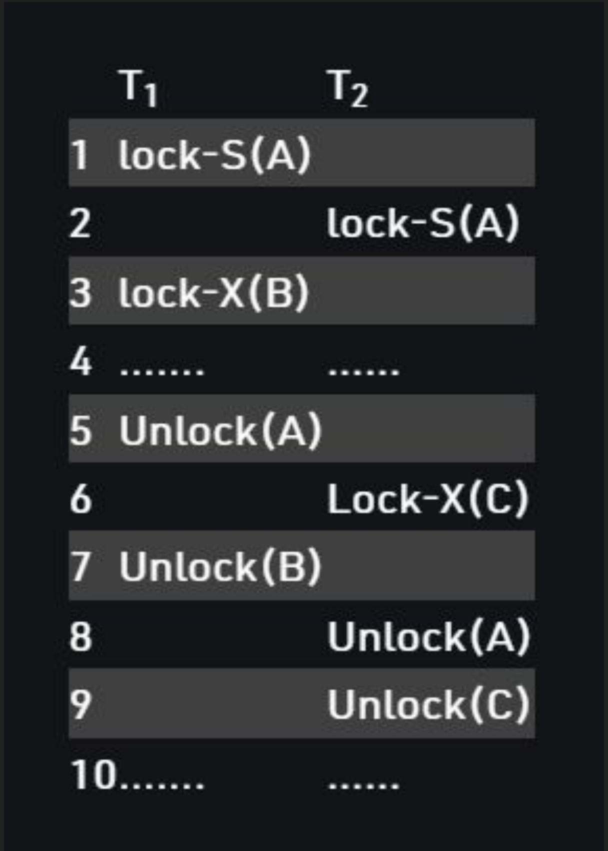
STARVATION IN 2-PL

	T1	T2	T3	T4
1		S(A)		
2	X(A) {T1 will wait for other process to complete}	..		
3		..	S(A)	
4		U(A)	..	
5			..	S(A)
6			U(A)	..
7				..
8				U(A)

SCHEDULING ALGORITHMS IN 2-PL

Scheduling Policy	Info
TMAXF	The transaction, which has locked the most data items among blocked transactions, is scheduled first.
OTF(Oldest Transaction First)	The oldest transaction is scheduled first. This is achieved by assigning each transaction a timestamp value, which is the time at which the transaction was generated. The transaction with the lowest timestamp value is scheduled first.

Transaction implementing 2 PL



	T_1	T_2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	....	......
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	....	......

This is just a skeleton transaction that shows how unlocking and locking work with 2-PL. Note for:

Transaction T1:

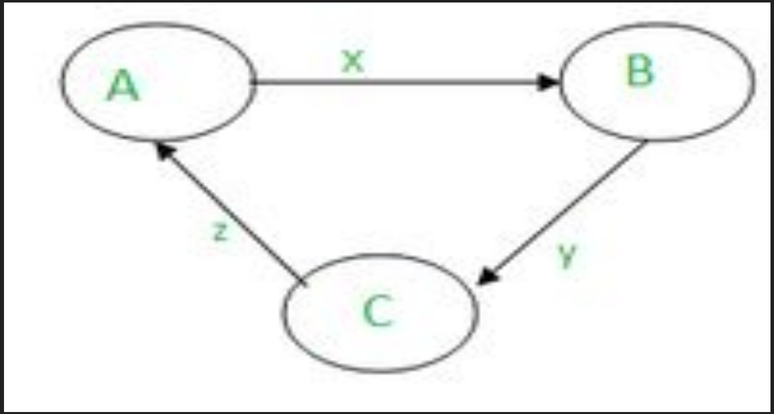
- The growing Phase is from steps 1-3.
- The shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T2:

- The growing Phase is from steps 2-6.
- The shrinking Phase is from steps 8-9.
- Lock Point at 6

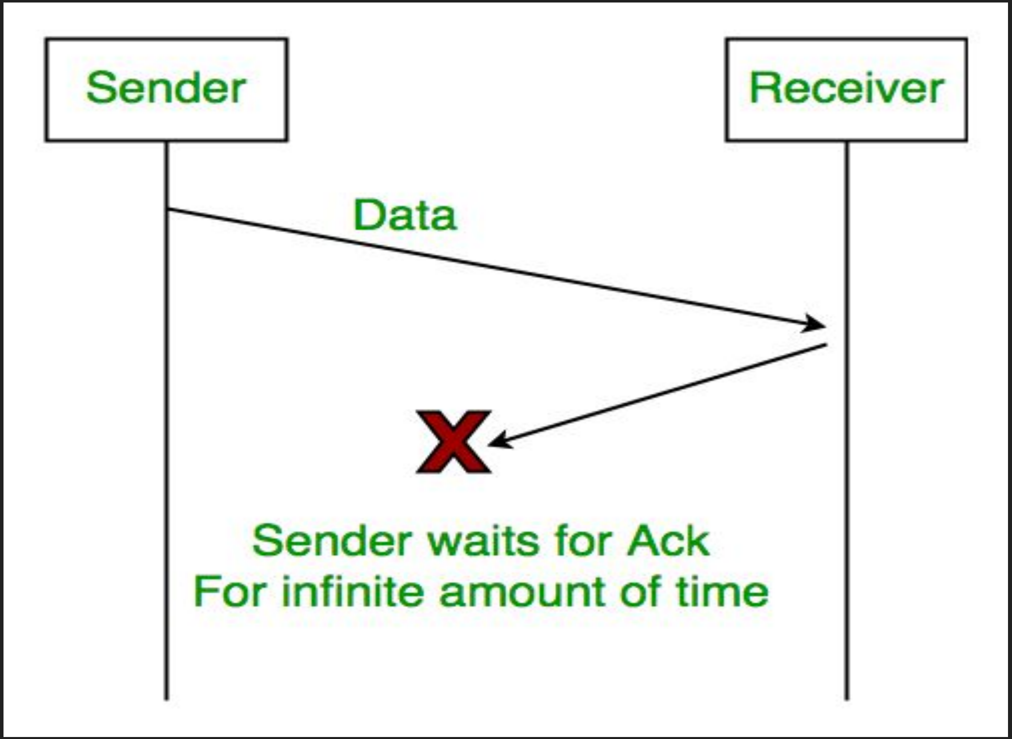
COMMUNICATION / NON-RESOURCE DEADLOCK

- Deadlocks can also occur in processes not involving resources at all.
- Processes wait to communicate with other process in a set of processes.
- On receiving communication from any of these processes, a waiting process can unblock. If each process in the set is waiting to communicate with another process in the set, and no process in the set ever begins any additional communication until it receives the communication for which it is waiting, the set is communication-Deadlocked.



- In the communication model, before a process can continue, it may know the identification of the processes from which it must receive a message to communicate.
- a process cannot proceed with the execution until it can communicate with one of the processes for which it is waiting.
- Resource deadlock can be prevented by a safe state, but communication deadlock cannot be prevented by a safe state.
- Waiting (by processes) is done for messages.

EXAMPLE:


Sender
Receiver
Data
Sender waits for Ack
For infinite amount of time

STARVATION

- **Starvation** or indefinite blocking is a phenomenon associated with the Priority scheduling algorithms, in which a process ready for the CPU (resources) can wait to run indefinitely because of low priority.

- In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

CAUSE OF STARVATION

- Starvation can occur due to deadlock, livelock, or caused by another process.
- As we have seen in the event of a deadlock or a live lock a process competes with another process to acquire the desired resource to complete its task. However, due to the deadlock or livelock scenario, it failed to acquire the resource and generally starved for the resource.
- Further, it may occur that a process repeatedly gains access to a shared resource or use it for a longer duration while other processes wait for the same resource. Thus, the waiting processes are starved for the resource by the greedy process.

AVOIDING STARVATION

- One of the possible solutions to prevent starvation is to use a resource scheduling algorithm with a priority queue that also uses the aging technique. Aging is a technique that periodically increases the priority of a waiting process. With this approach, any process waiting for a resource for a longer duration eventually gains a higher priority. And as the resource sharing is driven through the priority of the process, no process starves for a resource indefinitely.

- Another solution to prevent starvation is to follow the round-robin pattern while allocating the resources to a process. In this pattern, the resource is fairly allocated to each process providing a chance to use the resource before it is allocated to another process again.

REAL LIFE EXAMPLE OF STARVATION

Airport has a single check-in counter. There are two queues, one for business and one for economy class. However, there is a business convention nearby, and the business queue is always full. The economy queue doesn’t move at all. If you happen to be in economy class, you’ll miss your plane.

REFERENCES

Tanenbaum, A. (2014). Modern operating systems (4th ed.). Pearson Education.