# OS2 project documentation

# (Readers writers problem).

## Introduction:

The readers-writers problem is used to manage synchronization so that there are no problems with the object data. For example - If two readers access the object at the same time there is no problem. However, if two writers or a reader and writer access the object at the same time, there may be problems.

### 1.solution pseudocode

Writer process:

1.Writer requests the entry to critical section.

2.If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.

3.It exits the critical section.

```
do {
    // writer requests for critical section
    wait(wrt);

    // performs the write
```

```
        // leaves the critical section
        signal(wrt);

} while(true);
```

Reader process:

1.Reader requests the entry to critical section.

2.If allowed:

.it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.

.It then, signals mutex as any other reader is allowed to enter while others are already reading.

.After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.

3.If not allowed, it keeps on waiting.

```
do {

    // Reader wants to enter the critical section
    wait(mutex);
```

```
// The number of readers has now increased by 1
readcnt++;

// there is at least one reader in the critical section
// this ensure no writer can enter if there is even one reader
// thus we give preference to readers here
if (readcnt==1)
      wait(wrt);

// other readers can enter while this current reader is inside
// the critical section
signal(mutex);

// current reader performs reading here
wait(mutex);      // a reader wants to leave

readcnt--;

// that is, no reader is left in the critical section,
if (readcnt == 0)
        signal(wrt);                // writers can enter
```

```
    signal(mutex); // reader leaves

} while(true);
```

Thus, the semaphore ‘wrt‘ is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

## 2. Examples of Deadlock

1-Perhaps the most easily recognizable form of deadlock is traffic gridlock. Multiple lines of cars are all vying for space on the road and the chance to get through an intersection, but it has become so backed up there this no free space for potentially blocks around. This causes entire intersections or even multiple intersections to go into a complete standstill. Traffic can only flow in a single direction, meaning that there is nowhere for traffic to go once traffic has stopped. However, if the car at the very end of each line of traffic decide to back up, this frees up room for other cars to do the same and therefore the gridlock is solved. Another real-world example of deadlock is the use of a single track by multiple trains. Say multiple tracks converge onto one; there is a train on each individual track, leading to the one track. All trains are stopped, waiting for another to go, though none of them move. This is an example of deadlock because the resource, the train track, is held in a state of limbo as each train waits for another to move

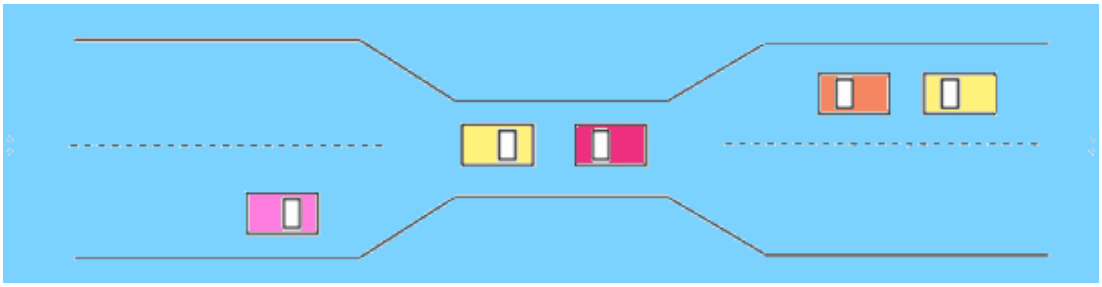
along so that they can continue.

2- A real-world example would be traffic, which is going only in one direction.

Here, a bridge is considered a resource.

So, when Deadlock happens, it can be easily resolved if one car backs up (Preempt resources and rollback).

Several cars may have to be backed up if a deadlock situation occurs.



## 3.How did solve deadlock

After diagnosing a deadlock problem, the next step is to attempt to resolve the deadlock issue resulting between two concurrently running applications each of which have locked a resource the other application needs. The guidelines provided here can help you to resolve the deadlock problem you are experiencing and help you to prevent such future incidents.

Before you begin, confirm that you are experiencing a deadlock problem by taking the necessary diagnostic steps for locking problems outlined in Diagnosing and resolving locking problems.

Theoretically, the solution to the readers-writers problem is as follows:

A writer should have exclusive access to the object in question when writing, meaning that no other readers nor writers should access the object during writing.

A reader has non-exclusive access to the object — meaning that multiple readers can operate at the same time.

Deadlock frequency can sometimes be reduced by ensuring that all applications access their common data in the same order - meaning, for example, that they access (and therefore lock) rows in Table A, followed by Table B, followed by Table C, and so on. If two applications take incompatible locks on the same objects in different order, they run a much larger risk of deadlocking.

What to do next?

Rerun the application or applications to ensure that the locking problem has been eliminated by checking the administration notification log for lock-related entries.

## 4.Examples of starvation

Starvation is a problem that is closely related to both, Livelock and Deadlock. In a dynamic system, requests for resources keep on happening. Thereby, some policy is needed to make a decision about who gets the resource and when this process, being reasonable, may lead to some processes never getting serviced even though they are not deadlocked.

Starvation is usually caused by an overly simplistic scheduling algorithm. For example, if a (poorly designed) multi-tasking system always switches between the first two tasks while a third never gets to run, then the third task is being starved of CPU time. The scheduling algorithm, which is part of the kernel, is supposed to allocate resources equitably; that is, the algorithm should allocate resources so that no process perpetually lacks necessary resources.

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. In starvation resources are continuously utilized by high priority processes. Problem of starvation can be resolved using Aging. In Aging priority of long waiting processes is gradually increased.

When the writer arrives and tries to do the same, it gets blocked . The writer must then wait until all readers have left. Once Reader B arrives, the two readers take turns leaving and re-entering. Since at least one reader is always in the system, the writer is blocked indefinitely, a situation known as starvation.

This can cause starvation if there are writers waiting to modify the resource and new readers arrive all the time , as the writers will never be granted access as long as there is at least one active reader. Give writers priority: here, readers may starve.

## 5. How did solve starvation

The writer's starvation in this scenario can be fixed by placing a turnstile before the readers can enter . As long as there is no writer attempting to enter the critical section, the readers can each pass through the turnstile and invoke enter() on the lightswitch.

One common solution for the readers-writers problem aligns with the asymmetric lightswitch described previously. As the readers allow concurrent access, the reader thread would use the enter() and leave() routines to access the shared resource. The writers, however, require mutual exclusion, both with other writers and with all of the readers. Consequently, the writers do not need to employ the lightswitch, simply accessing the semaphore directly.

## 6. Explanation for real world application and how did apply the problem

the Readers and Writers problem is useful for modeling processes which are competing for a limited shared resource. A practical example of a Readers and Writers problem is an airline reservation system consisting of a huge data base with many processes that read and write the data. Reading information from the data base will not cause a problem since no data is changed. The problem lies in writing information to the data base. If no constraints are put on access to the data base, data may change at any moment. By the time a reading process displays the result of a request for information to the user, the actual data in the data base may have changed. What if, for instance, a process reads the number of available seats on a flight, finds a value of one, and reports it to the customer. Before the

customer has a chance to make their reservation, another process makes a reservation for another customer, changing the number of available seats to zero.