READER-WRITER PROBLEM

-Starve free solution

Overview

A database is to be shared among several concurrent processes. Some of these processes may only read the database(referred as readers), whereas others may want to write (referred as writers) to the database.If two readers access the shared data simultaneously, no adverse effects will result. However, if a writer and some other process (either a reader or a writer) access the database simultaneously results in adverse effects.

So the solution could be :

- First readers-writers problem : No reader be kept waiting unless a writer has already obtained permission to use the shared object.
- Second readers-writers problem : Once a writer is ready, that writer performs its write as soon as possible.

First Readers-Writers Problem :

1. As per the statement reader is given higher priority compared to writer .
2. Race condition : A condition in which two or more processes are waking up simultaneously and trying to enter the critical section

   E.g. two readers increment the readcount at the same time.

Implementation :

Variables used for implementation are

Mutex = Semaphore(1); `// to ensure that no other reader will execute entry section while a present reader is in it i.e to ensure mutual exclusion .`

Int Readerscount = 0; // number of reader processes in critical section initialized to 0;

Resource = Semaphore(1); // used by both readers and writers to lock resource.

Pseudo Code :

Reader :

Do{

wait(mutex);

Readerscount = Readerscount+1;

if(Readerscount==1) wait(resource); //if it is first reader it reserves resource for other readers by locking resource from writers

signal(mutex);

Critical section

wait(mutex);

Readerscount = Readerscount-1;

if(readerscount==0) signal(resource);// If it is last reader unlock resource and make it available for writers.

signal(mutex);

}while(true)

Writer :

Do{

wait(resource); //request for resource and if granted lock the resource as no writer or reader can enter if writer is in CS

Critical section

signal(resource); `//release the shared resource`

}while(true);

Explanation :

- For a reader before it enters critical section it should go through entry section.To avoid race conditions every reader which enters the entry section will lock the entry section for themselves until they are done with it, with the help of semaphore mutex.
- Locking is done with wait(mutex) and unlocking with signal(mutex).
- The same is valid for the exit section.
- Once the first reader is in entry section it locks the resource preventing other writers from accessing it.
- Other readers can use the resource but the last reader must unlock the resource making it available for writers.
- For a writer it waits for unlocking of the resource and once all the readers are done it then locks the resource and enters the critical section avoiding readers from accessing it.
- In this solution a stream of readers can stop all the writers and starve them because after the first reader locks the resource, no writer can use it, before lock gets released and it will only be released by the last reader.
- Hence this solution is not fair.

Second Readers-Writers Problem :

- As per the statement writer is given higher priority compared to reader.

Implementation :

Variables used for implementation are

Int Readerscount = 0; `// number of reader processes in critical section initialized to 0`;

Int Writerscount = 0; //number of writer processes in critical section initialized to 0;

Readmutex = Semaphore(1); // to ensure that no other reader will execute readers entry section while a present reader is in it i.e to ensure mutual exclusion

Writemutex = Semaphore(1); // to ensure that no other writer will execute writers entry section while a present writer is in it i.e to ensure mutual exclusion

Resource = Semaphore(1);

R = Semaphore(1); // to lock readers for reserving space for other writers

PseudoCode :

Reader :

Do {

wait(mutex);

wait(readmutex);

Readerscount = Readerscount + 1;

if(Readerscount == 1) wait(Resource); // if it is first reader lock the resource

signal(readmutex); //release entry section for other readers

signal(mutex);

Critical section

wait(readmutex);

Readerscount = Readerscount -1;

if(Readerscount==0) signal(resource);//if it is last reader release the resource

signal(readmutex);

}while(true);

(only another mutex lock is added)

Writer:

```
Do{
wait(writemutex);
Writerscount = Writerscount+1;
if(Writerscount==1) wait(mutex); //if it is first writer then it locks critical section from other readers
signal(writemutex);
wait(Resource) //lock the shared resource as no writer or reader can enter if writer is in CS
```

Critical section

```
signal(resource); //release the shared resource
wait(writemutex);
Writerscount = Writerscount-1;
if(Writerscount==0) signal(mutex);// If it is last writer allow readers to enter critical section by unlocking resource
signal(writemutex);
}while(true)
```

Explanation :

- Here preference is given to writers as readers have to lock and unlock mutex individually while writers don't need to.
- The first writer will lock the mutex and then all subsequent writers can simply use the resource .The last writer must release the mutex semaphore,opening the gate for readers to try reading.
- The reader must wait for last writer to unlock the mutex and resource.
- If a reader locks the mutex semaphore it indicates to writer that there is a reader in critical section.So the writer will wait for the reader to release the mutex and then the writer will immediately lock it .
- Though, the writer will not be able to access the resource until the current reader has released the resource, after the reader is finished with the resource in the critical section.

- A reader will know that there are no writers waiting for resource with mutex semaphore It enters critical section by releasing mutex..
- As soon as a writer shows up, it will try to set the mutex and hang up there waiting for the current reader to release the mutex. It will then take control over the resource as soon as the current reader is done reading and lock all future readers out. All subsequent readers will hang up at the mutex semaphore waiting for the writers to be finished with the resource and to open the gate by releasing mutex.

Starve-free Readers-Writers Problem :

As both above solutions lead to starvation, third readers-writers is proposed with the principle **no thread shall be allowed to starve**.

Implementation :

Variables used for implementation are

Int Readerscount = 0; `// number of reader processes in critical section initialized to 0;`

Resource = Semaphore(1); `//used for requesting access to CS`

Queue = Semaphore(1); `//used for maintaining FIFO order of processes for using critical section`

Readmutex = Semaphore(1); `//used for avoiding race conditions in entry section and exit section`

Pseudo Code:

Reader :

Do{

wait(Queue); `//put blocked process in the queue to be processed`

wait(Readmutex);

Readerscount = Readerscount +1;

if(Readerscount ==1) wait(Resource); // if it is first reader lock the resource for writers

signal(Queue); // let next process in queue to be serviced

signal(Readmutex);

Critical section

wait(Readmutex);

Readerscount = Readerscount -1;

if(Readerscount == 0) signal(Resource); // If no readers are left release the resource so it can be used by other writer in queue

signal(Readmutex);

}while(true);

Writer :

Do{

wait(Queue); //put blocked process in the queue to be processed

wait(Resource); //lock the resource as no writer or reader can enter if writer is in CS

signal(Queue); // let next process in queue to be serviced

Critical section

Signal(Resource); //release resource for next reader/writer

}while(true);

Explanation :

- It is satisfied if semaphores preserve first-in first-out ordering while blocking and releasing threads.
- A reader will first wait in queue after it gets access it enters entry section .

- If it is first reader then it locks the resource stopping writers from accessing it.
- Then it gives access to the next process in queue .
- If there are no readers left then it unlocks the resource giving access to other writers.
- A writer will first wait in queue after it gets access it enters entry section .
- It then locks the resource avoiding other readers from accessing it.
- After it writes in critical section it then unlocks the resource giving access for next reader/writer.

Correctness :

Mutual exclusion :

If there is a writer in critical section no other process can access .This is ensured by resource semaphore .If there is reader in critical section no writer is allowed .It is ensured by the step 'wait(resource)' as writer needs to unlock the resource to enter critical section.So mutual exclusion is followed.

Progress :

Both reader and writer processes take finite time in critical section and also release the semaphores while leaving the critical section thus there is no possibility of deadlock here.

Bounded Wait :

The processes wait in FIFO queue.The queue semaphore ensures that it is done in FIFO sequence so all are executed after a finite time and therefore no starvation.