Airline Reservation System

Operating System project

Content :

01 Readers and Writers Problem Mercury

Mars Deadlock 02

03 Starvation Saturn

Jupiter Real-word Example 04

01 Readers and Writers Problem

Consider a situation where we have a file shared between many people.

- If one of the person tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- However if some person is reading the file, then others may read it at the same time.

Precisely in OS we call this situation as the **readers-writers problem,** Problem parameters:

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it
- If at least one reader is reading, no other process can write
- Readers may not write and only read

Here priority means, no reader should wait if the share is currently opened for reading.
Three variables are used: **mutex, wrt, readcnt** to implement solution

1. **semaphore** mutex, wrt;
2. **int** readcnt;

Functions for semaphore :
– wait() : decrements the semaphore value.
– signal() : increments the semaphore value.

1) Solution when Reader has the Priority over Writer 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.

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.

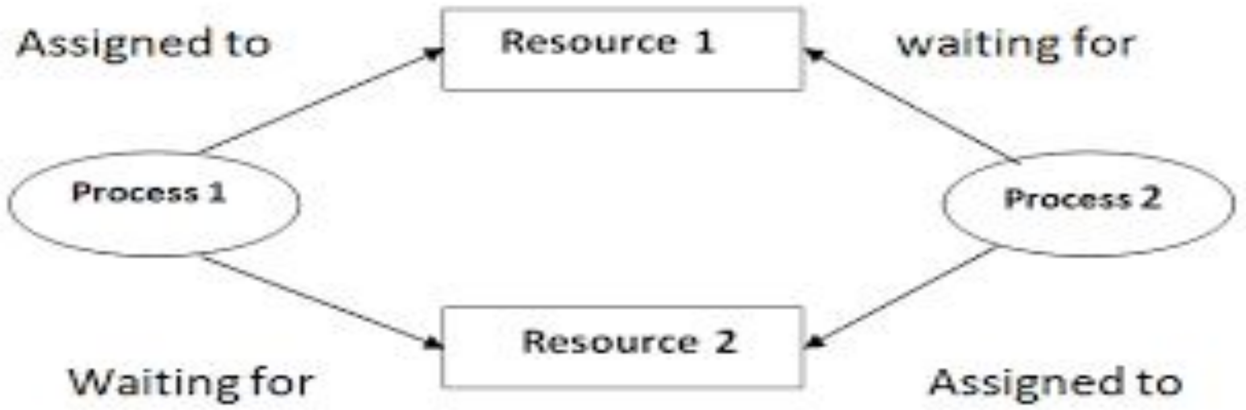
02

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.

Examples of Deadlock

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.
A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).
For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1



Deadlock can arise if the **following four conditions hold simultaneously (Necessary Conditions)**
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.

Methods for handling deadlock

There are three ways to handle deadlock

1) Deadlock **prevention** or **avoidance**: The idea is to not let the system into a deadlock state.

One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock.

Avoidance is kind of futuristic in nature. By using strategy of "Avoidance", we have to make an assumption. We need to ensure that all information about resources which process will need are known to us prior to execution of the process. We use **Banker's algorithm** in order to avoid deadlock.

2) Deadlock **detection and recovery**: Let deadlock occur, then do preemption to handle it once occurred.

3) **Ignore the problem altogether**: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

03 Starvation

Examples of starvation

The readers-writers problem has several variations, all involving priorities.

- The simplest one, referred to as the first readers-writers problem, requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting. (Priority for readers) In this case, **writers may starve**.
- The second readers-writers problem requires that, once a writer is ready, that writer performs its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading. (Priority for writers) In this case, **readers may starve**

How did solve starvation

Optimized Solution (Starve Free)

Logic

This solution with an optimization in the implementation of reader. Here too, the queue of in_mutex serves as a common waiting queue for the readers and the writers.

Global Variables

Global variables shared across all the processes.

Logic for the readers

In the previous solution, we had to enclose the entry part inside two mutex locks. But here, only one lock is sufficient.
Hence, the process need not be blocked twice.
This saves a great deal of time as blocking a process causes a lot of additional temporal overhead.
Here, initially we have the `in_mutex`.
Again, all the readers and writers have to queue in this mutex to ensure equal priority.
Once a reader acquires the `in_mutex` (after a writer completes its execution or after a fellow reader signals the mutex), it shows its presence by increasing the variable `readers_started` and then immediately signals the `in_mutex`.

The only thing that can keep a reader waiting, in this algorithm, is the wait for in_mutex.
Hence, the reader directly proceeds to its critical section.
Note that all the readers can read at the same time as only writers have a critical section in between the wait() and signal() methods of in_mutex. After the reader executes its critical section, the reader has to demark that it has completed its critical section execution and does not need the resource anymore.
So, it waits for the out_mutex and once it acquires it, it increments the variable readers_completed, thus, announcing its completion of resource usage.
Further, it goes to check if any writer is waiting by checking the variable writer_waiting.
If yes, it checks if any fellow readers are executing in their critical sections. If not, it signals the writer to start its execution by calling signal() on the semaphore write_sem.
After this, it signals the out_mutex to release the variable readers_completed for other readers and continues to its remainder section.

Logic for the writers

Firstly, the writers wait on the in_mutex with all the readers. After acquiring the in_mutex, the writer goes on to wait on the out_mutex. Now, after acquiring the out_mutex (which is just a means to introduce mutual exclusion for the variable readers_completed), it compares the variables readers_started and readers_completed. If they are equal, it means all the readers that had started their reading have completed it. That is, no reader is executing in its critical section currently. If that is the case, the writer simply signals the out_mutex, thus, releasing its control over the variable readers_completed and continues with its critical section. Note that any other reader or writer cannot execute in their critical sections as in_mutex is not signalled yet. If the variables readers_started and readers_completed are not equal i.e. there is are reader processes executing their critical sections, then, the writer changes the variable writer_waiting to true to state its presence and then, signals the out_mutex. Also, since the resource is busy, the writer waits in writer_sem for all the readers to complete their execution. Once it acquires writer_sem, it changes the variable writer_waiting back to false and proceeds to its critical section. Once the writer completes the critical section, it signals the in_mutex to state that it does not need the resource anymore. Now, the process next in queue of in_mutex can proceed.

04

Real-word Example

Airline reservation systems (**ARS**) are systems that allow an airline to sell their inventory seats. It contains information on schedules and fares and contains a database of reservations or passenger name records and of tickets issued if applicable. ARSs are part of passenger service systems PSS, which are applications supporting the direct contact with the passenger.

Additional Details About Our System :

1- Our system contain of 4 models that represent the need of our system
2- applied threads concepts on several classes such as the class that create Passenger ticket and more in our code ..
3- used the semaphore technique to avoid the occurrence of starvation and deadlock problem when two passengers try to take one ticket left
4- made a system for passengers who want to book a flight but there is no left tickets , to make their state waiting until opening a new flight.

And more and more in our project ….

Operating system 2 Project – Cover sheet

Project Title : **Airline Reservation System**

Readers-Writers Problem

Group# …………………………………………………..

Discussion time:- ………*1:00*…………………………………………..
Instructor …………………………………………………

ID	Name(Arabic)	Bounce	Minus	Total Grade	Comment
202000472	ضحى حسانين				
202000257	حبيبه عبد الغني				
202000661	كريم اشرف				
20200036	شريف شعبان				
202000409	مريم محمد				
202000409	سهيلة عبد الكريم				
202000005	ابراهيم عصام				