Project os2

Reader_writer problem pseudo code:

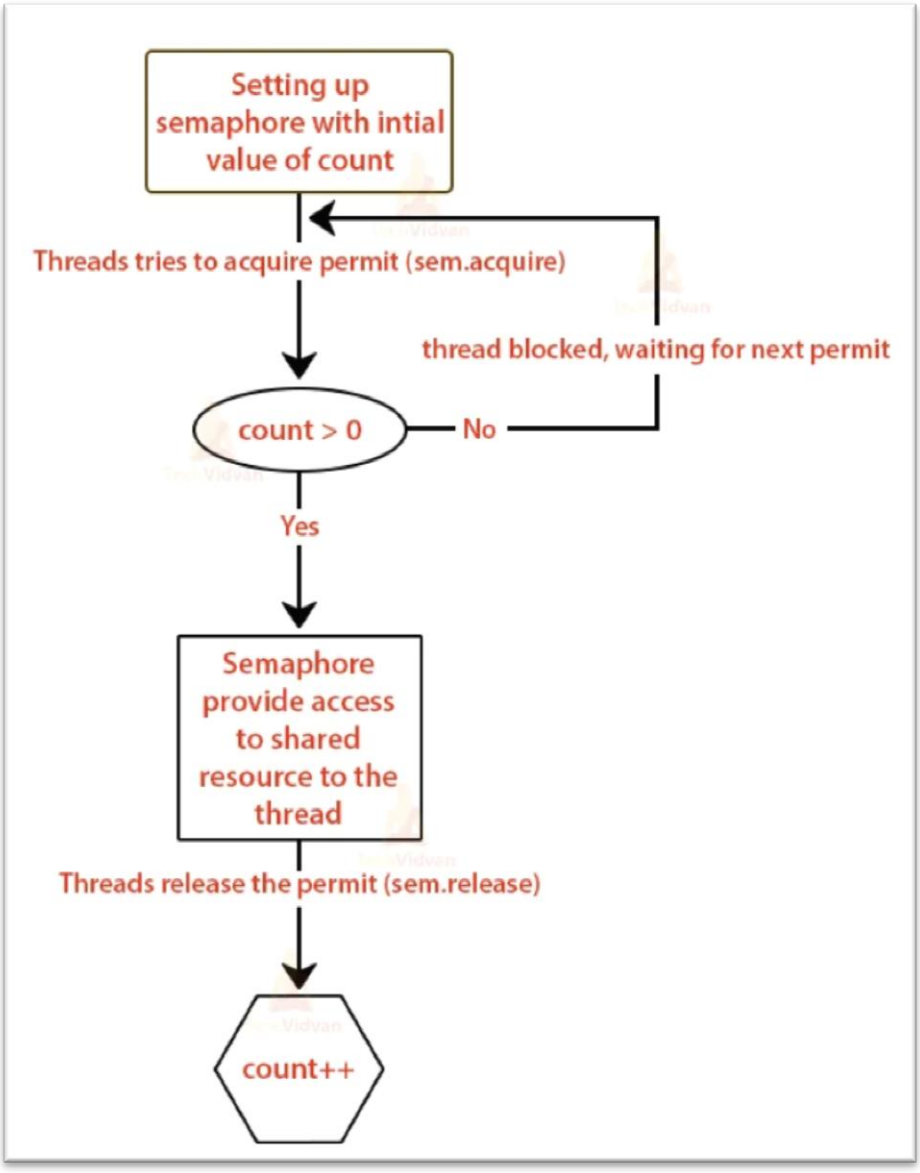
Pseudo code for a writer's process:

```
1. While (true)
2.      Set wait(rw_mutex)
3                . Writing is performed
4.      Set signal(rw_mutex)
5. End while
```

Pseudo code for a writer's process:

```
1. Do
2. Set wait(mutex)
3.      Read_count read_count + 1
4.      If read_count =1 then
5.              Wait(rw_mutex)
6. Set Signal(mutex)
7. Reading is performed
8. Set Wait(mutex)
9.      Readcount readcount – 1
10.     If readcount = 0 then
11.             Signal(rw_mutex)
12. Set Signal(mutex)
13. end while (true)
```

Flowchart with explanation of the Solution :



EXAMPLES OF DEADLOCK IN READER_WRITER PROBLEM:

Examples when the deadlock can occur and how we solved it :

Suppose that thread1 and thread2 are two threads that are in a deadlock. The thread thread1 holds the lock for the resource R1 and waits for resource R2 that is acquired by thread thread2.

At the same time, thread thread2 holds the lock for the resource R2 and waits

for R1 resource that is acquired by thread thread1. But thread2 cannot release the lock for resource R2 until it gets hold of resource R1.

Since both threads are waiting for each other to unlock resources R1 and R2, therefore, these mutually exclusive conditions are called deadlock in Java. Let's understand the concepts of deadlock with realtime examples.

EX1- Reader-Writer problem Without Monitor Cooperation:

```
class Message {

String message;

booleanempty = true;

public synchronized String read() {

while (empty) ;

empty = true;

return message;

  }

public synchronized void write(String message) {

while (!empty) ;

this.message= message;

empty = false;

   }

}
```

This scenario is called a Deadlock.When we started the threads from the Main class, both the threads called the **run()** method.Note that both threads are sharing a common message object. Now the Reader thread called the synchronized **read()** method and hence acquired the lock of the message object.As initially the boolean empty flag was set to true, the Reader

thread keeps executing while loop infinitely.Also, the Writer thread won't be able to execute the **write()** method as the lock of the message object is already acquired by the Reader thread.

EX2- Reader-writer problem

•The structure of a writer'sprocess

```
while (true) {
wait(rw_mutex);
.../* writing is performed */ ...
signal(rw_mutex);
}
```

•The structure of a readerprocess

```
do {
wait(mutex);
read_count++;
if (read_count== 1)
 wait(rw_mutex);
signal(mutex);
.../* reading is performed */ ...
wait(mutex);
 if (read_count== 0)
 signal(rw_mutex);
read count--;
signal(mutex);
} while (true);
```

-For this code will be cause daedlock .because last reader donnot enter if condition and read count will be zero after if condition so reader donnot release lock of semaphore **(rw_mutex)** This problem will prevent any recent reader from reading the data , and then writer couldn't acquire the lock .this will cause deadlock.

SOLUTION OF DEADLOCK IN READER_WRITER PROBLEM:

How we solved it :

// leave the lock to writer -and how allow to enter after Reader finish, so any deadlock problems are solved.

```
if (readCount == 0 ) {
        Writer.writeLock.release();
}
readLock.release();
// tack the lock Writer and prevent form enter with Reader, so any deadlock problems are solved.
if (readCount == 1  ) {
        Writer.writeLock.acquire();     }
readLock.release();
```

1- Solution of Reader-Writerproblem With Monitor Cooperation:

• Simply when the Writer thread writing a message add the **wait()** to read method until the writer finishes writing and release the lock and wake up the Reader thread using **notify()** or **notifyall()**

• Also, when the reader has the lock and reads the message, simply add the **wait()** to write method until the Reader finishes reading and releases

the lock and notifies the writer thread using **notify()** or **notifyall().**

```
class Message{
String message;
booleanempty = true;
public synchronized String read() {
while (empty) {
try {
wait();
} catch (InterruptedExceptione) {
System.out.println(Thread.currentThread().getName() + "Interrupted.");
}
}
empty = true;
notifyAll();
return message;
}
public synchronized void write(String message) {
while (!empty) {
try {
wait();
} catch (InterruptedExceptione) {
System.out.println(Thread.currentThread().getName() + "Interrupted.");
}
}
this.message= message;empty = false;
```

notifyAll();

}

}

Wait(): Reader thread waits until Writer invokes the notify()method or the notifyAll() method for 'message' object. Reader thread releases ownership of lock and waits until Writer thread notifies Reader thread waiting on this object's lock to wake up either through a call to the notify method or the notifyAllmethod.

Writer thread waits until Reader invokes the notifyAll() method for 'message' object. Writer thread releases ownership of lock and waits until Reader thread notifies Writer thread waiting on his object's lock to wake up .

notifyAll(): Wakes up all threads that are waiting on 'message' object's monitor(lock). This thread (Reader) releases the lock for 'message' object.

Wakes up all threads that are waiting on 'message' object's monitor(lock). This thread (Writer) releases the lock for 'message' object**.**

EX2- solution Reader-writer problem

•The structure of a readerprocess

do {

wait(mutex);

read_count++;

if (read_count== 1)

wait(rw_mutex);

signal(mutex);

.../* reading is performed */ ...

wait(mutex);

read count--;

if (read_count== 0)

signal(rw_mutex);

signal(mutex);

} while (true);

-In this code, the deadlock did not occur because the code was rearranged correctly,**read count--** in correct place the number of reader will decrease to zero and last reader will enter if condition and release lock of semaphore .

Case	Process 1	Process	Allowed / Not Allowed
Case 1	Writing	Writing	Not Allowed
Case 2	Reading	Writing	Not Allowed
Case 3	Writing	Reading	Not Allowed
Case 4	Reading	Reading	Allowed

The structure of writer process :

while (true) {

wait(rw mutex);

...

/* writing is performed */

...

signal(rw mutex);

}

---

wait(write);
WRITE INTO THE FILE
signal(wrt);

The structure of reader process :

while (true) {

wait(mutex);

read count++;

if (read count == 1)

wait(rw mutex);

signal(mutex);

...

/* reading is performed */

...

wait(mutex);

read count--;

if (read count == 0)

signal(rw mutex);

signal(mutex);

}

---

static int readcount = 0;
wait (mutex);

```
readcount ++; // on each entry of reader increment readcount

if (readcount == 1)
{
  wait (write);
}
signal(mutex);

--READ THE FILE?

wait(mutex);
readcount --; // on every exit of reader decrement readcount
if (readcount == 0)
{
  signal (write);
}
signal(mutex);
```

First starvation:

The first starvation occurs when the writer arrives and tries to go to the critical section, 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.

Second starvation :

The second starvation occurs when give the priority to the writer because once writer is ready , it performs as soon as possible and so maybe when the wirter want to exit he gives the give the Exclusive Key {rw mutex} to other writer so the reader will blocked until the last writer out from the system

Solutions of starvation:-

Initialisation	Reader	Writer
mx = Semaphore(1) wrt = Semaphore(1) ctr = Integer(0)	- Wait mx - if (++ctr)==1, then Wait wrt - Signal mx	- Wait wrt [Critical section]
	[Critical section]	- Signal wrt
	- Wait mx - if (--ctr)==0, then Signal wrt - Signal mx	

The only downside it has is the starvation of the Writer: a Writer thread does not have a chance to execute while any number of Readers continuously entering and leaving the working area.

To avoid this problem the following commonly known solution is proposed.

Initialisation	Reader	Writer
in = Semaphore(1) mx = Semaphore(1) wrt = Semaphore(1) ctr = Integer(0)	- Wait in - Wait mx - if (++ctr)==1, then Wait wrt - Signal mx - Signal in [Critical section] - Wait mx - if (--ctr)==0, then Signal wrt - Signal mx	- Wait in - Wait wrt [Critical section] - Signal wrt - Signal in

This solution is simple and fast enough. However the penalty in comparison to the previous one is that the Reader must lock two mutexes to enter the working area. If the working area is fast and assuming that mutex locking is a heavy system call, there would be a benefit of having an algorithm which allows locking one mutex on entering the working area and one on exiting.

The Reader-Writer Problem:

It relates to an object such as a file that is shared between multiple processes.

It 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.

Imagine, for example, an **airline reservation system**, with many competing processes wishing to read and write it. It is acceptable to have multiple processes reading the database at the same time, but if one process is updating (writing) the database, no other processes may have access to the database, not even readers.

In this solution, the first reader to get access to the database does a down on the semaphore *db.* Subsequent readers merely increment a counter, *rc.* As readers leave, they decrement the counter, and the last one out.

Suppose that while a reader is using the database, another reader comes along. Since having two readers at the same time is not a problem, the second reader is admitted. Additional readers can also be admitted if they come along.

Now suppose that a writer shows up. The writer may not be admitted to the database, since writers must have exclusive access, so the writer is suspended.

Later, additional readers show up. As long as at least one reader is still active, subsequent readers are admitted. As a consequence of this strategy, as long as there is a steady supply of readers, they will all get in as soon as they arrive.

The writer will be kept suspended until no reader is present. If a new reader arrives, say, every 2 seconds, and each reader takes 5 seconds to do its work, the writer will never get in.

Solution using semaphore:

```
typedef int semaphore;

semaphore mutex = 1;

semaphore write = 1;

int rc = 0;

void reader(void)

{

        while (TRUE)

        {
        down(mutex);

        r c = rc + 1;

        if (rc== 1)

        down(write);

        up(mutex);

        read_data_base();

        down(mutex);

        rc = rc - 1;

        if (rc = 0)
```

```
        up(write);

        up(mutex);

        use_data_read();
        }
}
  void writer(
```