A Report

On

Starve Free Reader Writer Problem

Submitted in requirement for the course

OPERATING SYSTEM (CSN-232)

by

Prakhar Agarwal

(17118050)

1. Problem

The Reader-Writer problem is an example of a synchronisation problem. There are two processes Reader and Writer that are sharing common resources e.g file at a time.
The former one only reads the database, whereas the latter one wants to update (that is, to read and write) the database.

- The problem is when one process starts writing the database and another performs starts reading the database at the same time then the system will go into an inconsistent state.
- Another problem arises when one process starts writing the database and another process also starts writing. As both processes want to update the value of the data present in the file at the same time it will lead to the inconsistency in the database.

Reader-Writer is categorized into three variants.
1.1 Give readers priority: when there is at least one reader currently accessing the resource, allow new readers to access it as well. 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.
1.2 Give writers priority: here, readers may starve.
1.3 Give neither priority: all readers and writers will be granted access to the resource in their order of arrival. If a writer arrives while readers are accessing the resource, it will wait until those readers free the resource, and then modify it. New readers arriving in the meantime will have to wait.

2. Solution Approach

2.1 Use of Critical Section

```
SomeProcess(){
        ...
        read(a) //instruction 1
        a = a + 5 //instruction 2
        write(a) //instruction 3
        ...
}
```

The above code is accessed by all processes and hence lead to data inconsistency,
To overcome this we placed the code in the critical section. In critical Section one process can access the code at one time. All the shared variables or resources are placed in the critical section.

2.2 Reader-Reader cause no problem

When a reader is present in critical section no writer should be granted access to enter there, while allowing multiple readers to access the critical section at a time (as this won't cause any problem)

3. Solution

The synchronisation problem can be solved with the help of semaphore variables.

1) **Semaphore ‘write_mutex’**
   It is used by the process that is writing in the file and it ensures that no other process should enter the critical section at that instant of time.

2) **Semaphore ‘r_mutex’**
   It is used to achieve mutual exclusion during changing the variable that is storing the count of the processes that are reading a particular file.

3) **Semaphore ‘seq_mutex’**
   It check for the turn to get executed, puts the blocked process in a fifo queue if it's not its turn currently

They are used to achieve mutual exclusion and initially their value is set to 1.

readers_count variable is used to count the number of processes that are reading a particular file. Initially the value is initialized to 0.
The wait() function is used to reduce the value of a semaphore variable by one.
The signal() function is used to increase the value of a semaphore variable by one.
All readers and writers will be granted access to the resource in their order of arrival (FIFOorder). If a writer arrives while readers are accessing the resource, it will wait until those readers free the resource, and then modify it as soon as the resource is released by the reader. New readers arriving in the meantime will have to wait.

4. Pseudo Code

Reader’s Code:

```
void *reader(void *id)
{
<Entry Section>
wait(seq_mutex);
wait(r_mutex);
readers_count++;
if(readers_count==1)
wait(write_mutex);
signal(seq_mutex);
signal(r_mutex);
<CRITICAL SECTION>
<Exit Section>
wait(r_mutex);
```

```
readers_count--;
if(readers_count==0)
signal(write_mutex);
signal(r_mutex);
}
```

Writer’s Code

```
void *writer(void *id)
{
sem_wait(&seq_mutex);
sem_wait(&write_mutex);
sem_post(&seq_mutex);
<CRITICAL SECTION>
sem_post(&write_mutex);
return id;
}
```

5. Correctness Of Solution

5.1 Case 1: Writing - Writing

It is the case when two writers try to update the data in the file at the same time.
Let’s see whether the code is working or not?

The initial value of semaphore (write_mutex) = 1
Suppose two processes p0 and p1 wants to write, let p0 enter first the writer code, The moment p0 enters sem_wait(&write_mutex); will decrease semaphore write_mutex by one, now write_mutex = 0 And p0 continues writing the file...
Now suppose p1 wants to write at the same time (will it be allowed?) let's see.
p1 does sem_wait(&write_mutex) since the write_mutex value is already 0 , hence P1 can never write anything, till P0 is writing.
Now suppose p0 has completed the task, it will do sem_post(&write_mutex) ; which will increase semaphore write_mutex by 1, now write_mutex = 1
if now p1 wants to write it can write since semaphore write_mutex > 0

This proves that, if one process is writing, no other process is allowed to write.

5.2 Case 2: Reading - Writing

It is the case when one or more processes is reading the file and another process try to update the data in the file at the same time.
Let’s see whether the code is working or not?

Initial value of semaphore r_mutex = 1 and variable readers_count = 0
Suppose two processes p0 and p1 are in a system, p0 wants to read while p1 wants to write.
p0 enters first into the reader code, the moment p0 enters , wait(r_mutex); will decrease semaphore r_mutex by 1, now r_mutex = 0 and readers_count will be incremented to 1.
If readers_count = 1 then wait(write_mutex); will set the value of semaphore write_mutex to 0 . and signal(r_mutex); will increase r_mutex by 1, now r_mutex =1 allowing other readers to enter.
Now any writer p1 wants to enter into its code then:
p1 does sem_wait(&write_mutex) since the write_mutex value is already 0 , hence P1 can never write anything, till P0 is there..

This proves that, if one process is reading, no other process is allowed to write.

5.3 Case 3: Writing-reading

It is the case when one process is writing the file and another process try to read the file at the same time.
Let's see whether the code is working or not?
Initial value of semaphore write_mutex = 1 and variable readers_count = 0
Suppose two processes p0 and p1 are in a system, p0 wants to write while p1 wants to read.
p0 does sem_wait(&write_mutex) since the write_mutex value is 1 , it will decrement by 1 and set to 0 and p1 continues writing the file….
Now p1 enters and does wait(r_mutex) which will set r_mutex to 0 and readers_count incremented by 1. If readers_count = 1 then wait(write_mutex); will not allow the process to move further since write_mutex is set to 0 by p0 process.

This proves that, if one process is writing, no other process is allowed to read.

5.4 Case 4: Reading-reading

It is the case when two readers try to read the data in the file at the same time.
Let's see whether the code is working or not?
Let three processes p0,p1 and p2 comes and try to read the data in the file.
P0 will set r_mutex to 0 , readers_count to 1 and write_mutex to 0 then it will set r_mutex again to 1 making another process (p1,p2) to enter.
Now when p1 comes it will set r_mutex to 0 , readers_count to 2 then it will set r_mutex again to 1 again.
Suppose p0 is now exiting and p2 tries to come but as the exit section involves shared variable readers_count , semaphore variable r_mutex preserve mutual exclusion allowing either p0 or p1 to execute at one time.

This proves that, if one process is reading, another process can read the file at the same time.

6. References

- ➢ https://afteracademy.com/blog/the-reader-writer-problem-in-operating-system
- ➢ https://afteracademy.com/blog/what-is-process-synchronization-in-operating-system
- ➢ https://www.javatpoint.com/os-readers-writers-problem
- ➢ https://rfc1149.net/blog/2011/01/07/the-third-readers-writers-problem/#:~:text=Give%20writers%20priority%3A%20here%2C%20readers,resource%2C%20and%20then%20modify%20it.