Reader – Writer Problem Documentation

Contents:

1- Solution pseudocode:

- Reader Function :

```
Reader() {
        wait(entry_mutex)
        ++in_count
        signal(entry_mutex)

           // ***** CRITICAL SECTION ***** //

         wait(out_mutex)
        ++out_count
        if(writer_waiting == true && (in_count == out_count)){
          signal(rw_mutex)
        }
        signal(out_mutex)
}
```

- Writer Function

```
Writer() {
   wait(entry_mutex)
   wait(out_mutex)
   if(in_count == out_count) {
      signal(out_mutex)
   }
   else {
      // means some reader process is in critical section
      writer_waiting = true
      signal(out_mutex)
      wait(rw_mutex)
      writer_waiting = false
   }

      // ***** CRITICAL SECTION ***** //

   signal(entry_mutex)
}
```

- Main function :

```
Main(){

int resources = 1
int in_count = 0, out_count = 0;
bool writer_waiting = false;
semaphore rw_mutex = new semaphore(0)
semaphore entry_mutex = new semaphore(1)
semaphore out_mutex = new semaphore(1)
r=Reader()
w=Writer()
input ReaderNum
input WritterNum
for (i:=1,j:=1;i< ReaderNum or j < WritterNum ; i++,j++):
      if(i<= ReaderNum):
          t1 :=  Thread(r,i + " :")
               t1.start()
      if(j<= WritterNum):
          Thread(w,j + " :").start()
}
```

2- Examples of Deadlock :

Assume N of Processes started the first process will enter critical section but no other process will so deadlock occur

```
Reader() {

      wait(read_mutex)
      wait(rw_mutex)

      // ***** CRITICAL SECTION ***** //

      signa(read_mutex)
      signal(rw_mutex)


}
```

```
Writer() {

   wait(rw_mutex)
   wait(read_mutex)


      // ***** CRITICAL SECTION ***** //

   signal(rw_mutex)
   signa(read_mutex)
 }
```

2.1- How did solve deadlock :

The Solution that we add “read_count” variable in reader and remove “read_mutex” from writer

```
Reader() {
   wait(read_mutex)
   ++read_count
   if((read_count == 1) {
      wait(rw_mutex)
   }
   signal(read_mutex)

      // ***** CRITICAL SECTION ***** //

wait(read_mutex);
   --read_count;
   signal(read_mutex);
   if(read_count == 0){
      signal(rw_mutex);
   }
}
```

```
Writer() {

   wait(rw_mutex)


      // ***** CRITICAL SECTION ***** //

   signal(rw_mutex)

 }
```

3 - 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.

Note : ***rw_mutex*** is implemented with ***1***

```
Reader() {
   wait(read_mutex)
   ++read_count
   if(read_count == 1) {
      wait(rw_mutex)
   }
   signal(read_mutex)
      // ***** CRITICAL SECTION ***** //
wait(read_mutex);
   --read_count;
   signal(read_mutex);
   if(read_count == 0){
      signal(rw_mutex);
   }
}
```

```
Writer() {
   wait(rw_mutex)
     // ***** CRITICAL SECTION ***** //
   signal(rw_mutex)
}
```

3.1 - How did solve starvation:

This problem can be tackled by using another two binary semaphores, which I call ***"entry_mutex"*** and ***"out_mutex"*** and 2 variables ***"in_count"*** and ***"out_count"*** to take care of synchronization and run fast and Boolean to know if there's writer is waiting ***"writer_waiting"***

Note : ***rw_mutex*** is implemented with ***0***

```
Reader() {
   wait(entry_mutex)
   ++in_count
   signal(entry_mutex)
      // ***** CRITICAL SECTION ***** //
   wait(out_mutex)
   ++out_count
   if(writer_waiting == true && (in_count == out_count)){
      signal(rw_mutex)
   }
   signal(out_mutex)
}
```

```
Writer() {
   wait(entry_mutex)
   wait(out_mutex)
   if(in_count == out_count) {
      signal(out_mutex)   }
   else {
      writer_waiting = true
      signal(out_mutex)
      wait(rw_mutex)
      writer_waiting = false
   }
      // ***** CRITICAL SECTION ***** //
   signal(entry_mutex)
}
```

4- Explanation for real world application and how did apply the problem :

Airline System that stores the available seats in planes That let customers to view and reserve a seat in plane

Some customers try to view the available seats of a flight named ”Flight-1" while other customers try to Reserve a seat in this flight

The system doesn’t allow the viewer customer to view the available seats in the same time that the customer reserve ticket