Indian Institute of Technology,Roorkee

CSN-232: Operating Systems

Starve Free Readers-Writers Problem

Professor:
Sateesh Kumar Peddoju
Department of Computer Science

Student:
M.Dileep Kumar Reddy
19114050,CSE

April 28, 2021

Contents

1 Introduction **2**

2 Difficulties in Readers-Writers Problems **2**

3 Starvation-Free Readers-Writers Problem **2**

4 Correctness of Proposed Logic **5**

Abstract

Solution for Starve Free Readers-Writers Problem

1 Introduction

A classical Readers-writers problem is a situation in which a data structure can be modified by concurrent threads.Only one writer is allowed to access the critical section at a time.And many readers can access critical section as long as there are no writers accessing the critical section. To allow concurrent threads mutually exclusive access to the data structure, we use mutexes or semaphores or monitors,etc.

2 Difficulties in Readers-Writers Problems

Readers-Writers Problem can be implemented in the following ways

1.First Readers-Writers Problem – No reader is kept waiting unless a writer is accessing the critical section.
2.Second Readers-Writers Problem – Once a writer is ready, it is allowed access to critical section as soon as possible,i.e no reader is allowed access to critical section unless there is no waiting writer.

In the First Readers-Writers Problem writer is kept waiting indefinitely if the readers enter continuously. Similarly if writers enter continuously, readers are kept waiting indefinitely in Second Readers-Writers Problem. So both the solutions may lead to starvation.

3 Starvation-Free Readers-Writers Problem

In order to avoid Starvation, one of the ideas is to allow the competing (Here Readers vs Writers) processes to access the critical section if they are ready just like in Peterson's solution for critical section problems.

Common Data Area

```
1 lock - binary semaphore
2 r_sem,w_sem
3 nw_waiting,nw_active
4 nr_waiting,nr_active
```

Reader Process

```
do{
    //entry section
    wait(lock);
    if(nw_waiting+nw_active == 0)
    {
      nr_active++;
      signal(r_sem);
    }
    else
    {
      nr_waiting++;
    }
    signal(lock);
    wait(r_sem);

    //critical section
    reading....

    //exit section
    wait(lock);
    nr_active - -;
    if((nr_active == 0) && (nw_waiting > 0))
    {
      signal(w_sem); //allowing single writer to enter
      nw_active++;
      nw_waiting - -;
    }
    signal(lock);

    //remainder section

}while(true);
```

Writer Process

```
do{
    //entry section
    wait(lock);
    if(nr_waiting+nr_active == 0)
    {
      nw_active++;
      signal(w_sem);
    }
    else
    {
      nw_waiting++;
    }
    signal(lock);
    wait(w_sem);

    //critical section
    writing....

    //exit section
    wait(lock);
    nw_active - -;
    if((nw_active == 0) && (nr_waiting > 0))
    {
      while(nr_waiting > 0)
      {
       signal(r_sem); //allowing all waiting readers to enter
       nr_active++;
       nr_waiting - -;
      }
    }
    signal(lock);

    //remainder section

}while(true);
```

4 Correctness of Proposed Logic

Mutual Exclusion

1. The variables nr_waiting,nr_active,nw_waiting,nw_active are accessed mutual exclusively by using semaphore lock.
2. If there are no active writer processes or waiting writer process, all the reader processes can become active.
3. If any writer process is accessing the critical section, all the reader processes enter waiting state.
4. Thus Readers and Writers access data maintaining mutual exclusion principle.

Progress

1. Since a single Writer or a set of Readers are always accessing the critical section, this ensures progress.

Bounded Waiting

1. A Reader has to wait till the Writer process currently accessing the critical section is complete.
2. A Writer has to wait till all the Readers currently accessing the critical section are complete.
3. In either the Reader or Writer case, a process has to wait as long as the number of opposing waiting processes, which is bounded(Since the waiting queue is of finite length).
4. Hence Bounded Waiting is ensured.

Starvation

There is finite time between the instant process enters the waiting queue and the instant it is allowed access to critical section. So there is no indefinite waiting of any process. So Starvation is avoided. And due to the usage of "lock" semaphore deadlocks are avoided. Because only one of all the waiting processes can access the entry section of code.