SECOND READERS-WRITERS PROBLEM
Network Programming Lab (CS334)

May 13, 2021

Sheen Xavier A
Roll No : 57
University Roll No : TVE18CS058
Department of Computer Science and Engineering
College of Engineering, Trivandrum

Contents

Second Readers-Writers Problem

1.1 Aim

To implement the Second Readers-Writers problem which is a classical synchronisation problem.

1.2 Theory

Problem Statement

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context: the reader and the writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non zero number of readers accessing the resource at that time. Also, no writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is also called writers-preference.

Solution

The implementation involves the use of semaphores for synchronisation purposes. In this solution, preference is given to the writers. This is done by forcing every reader to lock and release the writer_reader mutex individually. In the case of writers, only the first writer will lock the writer_reader and then all subsequent writers can simply use the resource as it gets freed by the previous writer. The very last writer must release the writer_reader semaphore, thus opening the gate for readers to try reading. The resource semaphore can be locked by both the writer and the reader in their entry section. They

are only able to do so after first locking the writer_reader semaphore, which can only be done by one of them at a time.

If there are no writers wishing to get to the resource, as indicated to the reader by the status of the writer_reader semaphore, then the readers will not lock the resource. As soon as a writer shows up, it will try to set the writer_reader and wait for the current reader to release the writer_reader.

1.3 Algorithm

```
// READER PROCESS
BEGIN
  WAIT(WRITER_READER).
  // CRITICAL SECTION
  PERFORM READ AND PRINT THE VALUE OF SHARED VARIABLE, DATA.
  SIGNAL(WRITER_READER).
END

// WRITER PROCESS
BEGIN
  WAIT(MUTEX).
  // CRITICAL SECTION
  WRITER_CNT = WRITER_CNT + 1.
  IF WRITER_CNT == 1, THEN WAIT(WRITER_READER).
  SIGNAL(MUTEX).
  PERFORM MODIFICATION AND PRINT THE VALUE OF THE SHARED VARIABLE,
  DATA.
  WAIT(MUTEX).
  // CRITICAL SECTION
  WRITER_CNT = WRITER_CNT - 1.
  IF WRITER_CNT == 0, THEN SIGNAL(WRITER_READER).
  SIGNAL(MUTEX).
END

// MUTEX : THE MUTEX LOCK SEMAPHORE
// WRITER_READER : THE SEMAPHORE FOR SWITCHING FOR HANDLING READER AND
   WRITER PROCESSES
// DATA : THE COMMON SHARED DATA
// WRITER_CNT : KEEPS TRACK OF THE NUMBER OF WRITERS
```

1.4 Source Code

```
// Name   : readerwriter2.c
// Desc   : Program to demonstrate second readers-writers problem(writer's preference)
// Input  : None
// Output : Interaction of readers and writers with the shared resource
// Author : Sheen Xavier A
// Date   : 12/05/2021

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

// Semamphores for reader and writer processes
sem_t writer_reader;
pthread_mutex_t mutex; // Mutex lock
int data = 4; // Shared data
int writer_cnt = 0; // Keeps track of the count of readers

// Handles the operation of reader process
void *reader(void *rcno)
{
    sem_wait(&writer_reader); // Change to read mode
    // Read content
    printf("Reader ID : %d reads data from the shared resource as %d\n", *((int *)rcno), data);
    sem_post(&writer_reader); // Change to writer mode
}

void *writer(void *wcno)
{
    int prev_data = data;

    pthread_mutex_lock(&mutex); // Aquiring the mutex lock
    writer_cnt++;   // Increment the number of writers currently writing

    if(writer_cnt == 1) // If we're the first writer block the reader
        sem_wait(&writer_reader); // Change the mode to writer mode
```

```
37
38      pthread_mutex_unlock(&mutex); // Relinquish the mutex lock
39      data = data * 4; // Perform modification to data
40      printf("Writer ID : %d modified shared data from %d to %d\n", *((
    int *) wcno), prev_data, data);
41
42      pthread_mutex_lock(&mutex); // Aquire the mutex lock
43      writer_cnt--;   // Decrement the number of readers while exiting
44
45      if(writer_cnt == 0) // If we're at the last writer relinquish
    control
46          sem_post(&writer_reader); // Change mode back to reader
47
48      pthread_mutex_unlock(&mutex); // Relinquish the mutex lock
49  }
50
51  int main()
52  {
53      int i;
54      pthread_t readers[5], writers[5]; // Reader and Writer process
55      pthread_mutex_init(&mutex, NULL); // Initialize the mutex lock
56      sem_init(&writer_reader, 0, 1);  // Initialize the semaphore
    associated with reader and writer
57
58      int numbarray[5] = {1, 2, 3, 4, 5}; // For IDs of reader and writer
59
60      for(i = 0 ; i < 5 ; i++) // Create Writer and Reader Processes
61      {
62          pthread_create(&writers[i], NULL, (void *)writer, (void *)&
    numbarray[i]);
63          pthread_create(&readers[i], NULL, (void *)reader, (void *)&
    numbarray[i]);
64      }
65      sleep(1); // Bringing a delay
66      pthread_mutex_destroy(&mutex); // Removing the semaphore and mutex
    lock
67      sem_destroy(&writer_reader);
68
69      return 0;
70
71  }
```

1.5 Output

```
sheenxavi004@Beta-Station:~$ cc readerwriter2.c -pthread
sheenxavi004@Beta-Station:~$ ./a.out
Writer ID : 1 modified shared data from 4 to 16
Writer ID : 2 modified shared data from 16 to 64
Writer ID : 3 modified shared data from 64 to 256
Reader ID : 1 reads data from the shared resource as 256
Reader ID : 3 reads data from the shared resource as 256
Reader ID : 4 reads data from the shared resource as 256
Writer ID : 4 modified shared data from 256 to 1024
Reader ID : 2 reads data from the shared resource as 1024
Writer ID : 5 modified shared data from 1024 to 4096
Reader ID : 5 reads data from the shared resource as 4096
```

Figure 1.1: Interaction between five Reader and five Writer Processes

1.6 Result

The program was developed as per requirement and the output was verified. It was also tested against various test cases.