MIT World Peace University

Operating Systems
Second Year B. Tech, Semester 3

Process Synchronization - Simulation of Reader-Writer Problem in C

Assignment 2
Practical Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 3, 2022

1 Code

```
// Reader writer problem.

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

sem_t sem_wrt;
sem_t sem_mutex;
int shared_variable = 0;
int number_of_readers;

void *reader()
{
    sem_wait(&sem_mutex);
    printf("\nRead: %d\n", shared_variable);
    printf("Reader finished its CS so releasing mutex\n");
    sem_post(&sem_mutex);
}

void *writer()
{
    sem_wait(&sem_wrt);
    sem_wait(&sem_mutex);
    printf("Blocking sem wait and mutex variable so no other writer can write rn.
\n");
    shared_variable++;
    printf("Wrote to the shared variable %d\n", shared_variable);
    sem_post(&sem_wrt);
    sem_post(&sem_mutex);
}

int main()
{
    pthread_t t1, t2;
    sem_init(&sem_mutex, 0, 1);
    sem_init(&sem_wrt, 0, 1);

    printf("Enter how many readers and Writers you want (Same number of both are
taken by default): ");
    scanf("%d", &number_of_readers);

    for (int i = 0; i < number_of_readers; i++)
    {
        pthread_create(&t2, NULL, writer, NULL);
        pthread_create(&t1, NULL, reader, NULL);
    }

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&sem_mutex);
}
```

Listing 1: Assignment 5.Cpp

2 Input and Output

```
Enter how many readers and Writers you want (Same number of both are taken by
    default): 8
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 1

Read: 1
Reader finished its CS so releasing mutex
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 2

Read: 2
Reader finished its CS so releasing mutex
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 3

Read: 3
Reader finished its CS so releasing mutex
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 4

Read: 4
Reader finished its CS so releasing mutex
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 5

Read: 5
Reader finished its CS so releasing mutex
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 6

Read: 6
Reader finished its CS so releasing mutex

Read: 6
Reader finished its CS so releasing mutex

Read: 6
Reader finished its CS so releasing mutex
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 7
Blocking sem wait and mutex variable so no other writer can write rn.
Wrote to the shared variable 8
```

Listing 2: Input and Output.Cpp

OS – FAQ – Assignment – 5

8/11/22

(A) Reader Writer Problem

Krishnaraj PJ.
10322 10089
PA20 ; Batch A1

Q.1 Explain the working of semaphores.

→ If the same resource is to be modified and / or accessed ~~by~~ by 2 or more processes, they can either win or lose the race to change that variable first; depending on this, program may change its output.

This reduces reliability.

int a ;

Program – 1	Program – 2
read (a) ;	read (a) ;
a = 5 ;	a = 10 ;
read (a) ;	read (a) ;

5 10

$\boxed{a = ?}$

→ To solve this, we need semaphores. It is a solution to solve mutual exclusion.

→ Uses an integer to maintain state of processes.

→ Can only be accessed through wait () and signal ().

→ wait () tells the OS and other programs that current invoking process is critically using some resource. It then uses signal () to then signal the freedom of that resource.

→ wait (s)
{ while (s ≤ 0) s-- ; }

→ signal (s)
{ s++ }

→ So process can safely edit values of variables and resources without causing confusion

→ process_1 () // example of semaphore
{ { wait (m) ;
// critical task on some; resource
signal (m) ; // task done
} }

(Q.2) Discuss producer consumer problem and devise a solution with semaphores.

→ It is a classic problem of synchronization.
→ Producer produces an item and adds it to a buffer of limited size.
→ Consumer then takes out an item & consumes it
→ Buffer is a shared resource and must be used in ~~shared~~ mutual exclusion manner by both

→ Produce must be stopped from adding to a full buffer

→ Consume must be stopped from consuming an empty buffer.

(*) <u>Solution using semaphores</u>.

Shared initialization

```
char item, buffer [n];
semaphore full = 0, empty = n, mutex = 1;
char nextp, nextc;
```

→ producer process ()

```
do {  produce item is nextp
      wait (empty);
      wait (mutex);
      add nextp to buffer;
      signal (mutex);
      signal (full);
} while (true);
```

→ Consumer process ()

```
do {  wait (full);
      wait (mutex);
      remove an item from buffer to nextc;
      signal (mutex);
      signal (empty);
      consume item is nextc;
} while (true);
```

(4) Q.3. List and discuss different process synchronization ~~Problems~~ mechanisms..

→ ~~Petersons~~ There are a few solutions to solving and implementing process synchronization.

(1) **Peterson's Solution** :

→ When a process is executing in a critical section, then the other process executes the rest of the code. and vice versa. This makes sure only one process executes the critical section at a time.

(2) **Mutex locks** : it is a locking mechanism

→ used to synchronize access to a resource in the critical section. we use a lock over the critical section. It is set when process enters and unset when it exists.

(3) **Semaphores** :

→ It is a signaling mechanism ; and a process can signal a process that is waiting on a semaphore. It differs from mutex, in that mutex can only be notified upon exit, but semaphores use wait () and signal () functions for synchronization.