Patterns and Problems of Synchronization

M. Usman Awais

National University of Computing and Emerging Sciences

usman.awais@nu.edu.pk

March 17,2016

References

Allen B. Downey (2005)
A Little Book of Semaphores

Overview

Rendezvous Solution

$sem_1 = 0, sem_2 = 0$

1: $statement\ a_1$
2: $sem_1.signal()$
3: $sem_2.wait()$
4: $statement\ a_2$

Process A

1: $statement\ b_1$
2: $sem_2.signal()$
3: $sem_1.wait()$
4: $statement\ b_2$

Process B

Mutex

$mutex.wait()$
    //Critical Section
    $count++$
$mutex.signal()$

Mutex

To allow access to only one thread into the critical section

Multiplex

To allow access to n threads, initialize the mutex to n

Barrier Problem

Requirment

The requirement is that no thread executes critical point until after all threads have executed rendezvous.

Barrier Solution

```
rendezvous
mutex.wait()
    count++
mutex.signal()
if count == n:
    barrier.signal()
barrier.wait()
critical point
```

Barrier Solution

```
rendezvous
mutex.wait()
    count ++
mutex.signal()
if count == n:
    barrier.signal()
barrier.wait()
critical point
```

Deadlock

Only one thread comes out of the barrier when $count == n$, after that no other thread is signaled.

Barrier Correct Solution

```
rendezvous
mutex.wait()
    count++
mutex.signal()
if count == n:
    barrier.signal()
barrier.wait()
barrier.signal()
critical point
```

Deadlock Removed

Now each signaled thread signals another thread.

Queue

Playing Cricket

You need at least one bowler and one batsman to play cricket. We want to have a practice session of n bowlers and m batsmen.

Bowlers and Batsmen

There are two queues, one for bowlers and one for batsmen. We should only allow a practice session if there is a bowler and a batsman available.

Solution

$bowlerSem = 0$
$batterSem = 0$

Batsmen ...

1: $bowlerSem.post()$
2: $batterSem.wait()$
3: $Play()$

Bowlers ...

1: $batterSem.post()$
2: $bowlerSem.wait()$
3: $Play()$

Solution

$int\ bowlers = 0, int\ batsmen = 0$

$bowlerSem = 0, batterSem = 0, mutex = 1$

Batsmen ...

```
mutex.wait()
    if bowlers > 0:
        bowlers − −
        bowlerSem.post()
    else:
        batsmen + +
        mutex.post()
        batterSem.wait()
    Play()
mutex.post()
```

Bowlers ...

```
mutex.wait()
    if batsmen > 0:
        batsmen − −
        batterSem.post()
    else:
        bowler + +
        mutex.post()
        bowlerSem.wait()
    Play()
```

Producer Consumer Problem

Shared Buffer

A shared buffer *BUFF* between two processes, i.e. *Producer* and *Consumer*. For the fixed buffer size we assume that we have the buffer size represented by *BS*

Producer

Gets data from somewhere, e.g. hard disk and puts the data into *BUFF*.

Consumer

Gets data out from *BUFF* and uses it in some way.

Solution Developed in Class

$$sem_1 = BS, sem_2 = 0, mutex = 1, i = 0$$

while /*condition*/ **do**
  $sem_1.wait()$
    $mutex.wait()$
      $i++$
      $buffer.put(i, data)$
    $mutex.post()$
  $sem_2.post()$
end while

Producer

while /*condition*/ **do**
  $sem_2.wait()$
    $mutex.wait()$
      $i--$
      $buffer.get(i, \&data)$
    $mutex.post()$
  $sem_1.post()$
end while

Consumer

A Generic Solution

$$sem[n] = new\ \ sempahore[n], mutex = new\ \ semaphore()$$
$$sem[0] = c, sem[1\ \ to\ \ n] = 0, mutex = 1$$

while /*condition*/ **do**
    $sem[i].wait()$
        $mutex.wait()$
            //Critical Section
        $mutex.post()$
    $sem[(i+1)\%(n+1)].post()$
end while

The LBoS Solution

$$spaces = BS, items = 0, mutex = 1$$

```
while /*condition*/ do
    spaces.wait()
        mutex.wait()
            buffer.put()
        mutex.post()
    items.post()
end while
```

Producer

```
while /*condition*/ do
    items.wait()
        mutex.wait()
            buffer.get()
        mutex.post()
    spaces.post()
end while
```

Consumer

Readers Writers Problem

Shared Resource

A resource *Rsc* shared among more than two processes, i.e. some *Writers* and few *Readers*.

Writers

Always need exclusive access. When any one of the writers is writing, no one else should be able to access the *Rsc*.

Readers

Many readers can read the *Rsc* simultaneously.

Solution-1

$$int\ readers = 0$$
$$roomEmpty = 1, mutex = 1$$

```
roomEmpty.wait()
    Write(Rsc)
roomEmpty.post()
```

Writer

```
mutex.wait()
    readers ++
    if readers == 1 :
        roomEmpty.wait()
mutex.post()
Read(Rsc)
mutex.wait()
    readers --
    if readers == 0 :
        roomEmpty.post()
mutex.post()
```

Reader

Solution-2: No Starvation Solution

$int\ readers = 0, turnstile = 1$

$roomEmpty = 1, mutex = 1$

$tunrstile.wait()$

    $roomEmpty.wait()$

    Critical Section

    $roomEmpty.post()$

$tunrstile.post()$

Writer

$tunrstile.wait()$

$tunrstile.post()$

$mutex.wait()$

    $readers++$

    if $readers == 1$**:**

        $roomEmpty.wait()$

$mutex.post()$

Critical Section

$mutex.wait()$

    $readers--$

    if $readers == 0$**:**

        $roomEmpty.post()$

$mutex.post()$

A Slight Improvement

$int\ readers = 0, turnstile = 1$
$roomEmpty = 1, mutex = 1$

```
tunrstile.wait()
    roomEmpty.wait()
    Critical Section
tunrstile.post()
    roomEmpty.post()
```

The change gives better chance to writers to get inside critical section multiple times

```
tunrstile.wait()
tunrstile.post()
mutex.wait()
    readers++
    if readers == 1:
        roomEmpty.wait()
mutex.post()
Critical Section
mutex.wait()
    readers--
    if readers == 0:
        roomEmpty.post()
mutex.post()
```