Operating Systems 2
Homework 1

Abhinav Gupta
ES15BTECH11002

February 10, 2018

This document is generated by LaTeX

1. Consider the semaphore based solution to producer-consumer bounded buffer problem that we discussed in the class. Please answer the following:

a. What is the minimum that the semaphores full and empty can become? Please explain how.

b. Similarly, what is the maximum that the semaphores full and empty can become? Please explain how.

c. Please give the upper and lower bounds on the sum of full and empty in any given state.

Sol.

a. If busy waiting is allowed, that is if we are using traditional semaphores :

The minimum that the semaphore full can become is 0, at the beginning when no producer has yet produced anything.

The minimum that the semaphore empty can become is 0, when n producers have produced and no consumer has consumed yet. (n is the buffer size here)

If a queue is associated with each semaphore :

The minimum that the semaphore full can become is -inf, as any number of consumers can come and execute a wait operation on full semaphore (supposing no producer process has come yet).

The minimum that the semaphore empty can become is -inf, as any number of producers can come and execute a wait operation on empty semaphore (supposing no consumer process has come yet).

b. If busy waiting is allowed, that is if we are using traditional semaphores :

The maximum that the semaphore full can become is n,when n producers have produced and no consumer has consumed yet. (n is the buffer size here)

The maximum that the semaphore empty can become is n, at the beginning when no producer has yet produced anything(n is the buffer size here)

If a queue is associated with each semaphore :

The maximum that the semaphore full can become is n, as any number of producers can come and execute a wait operation on empty semaphore, but will be blocked if empty semaphore was 0 before this process entered and will not be able to execute signal operation on full.
Note: Until the semaphore empty do not go to zero, signal operation can be executed by the producer on full, thereby increasing its value.(supposing no consumer process has come yet).

The maximum that the semaphore empty can become is n, as any number of consumers can come and execute a wait operation on full semaphore, but will be blocked if full semaphore was 0 before this process entered and will not be able to execute signal operation on empty.
Note: Until the semaphore full do not go to zero, signal operation can be executed by the consumer on empty, thereby increasing its value.(supposing no producer process has come yet).

c. If busy waiting is allowed, that is if we are using traditional semaphores :
The sum of full and empty semaphores in any given state is a constant and

is equal to n.

If a queue is associated with each semaphore :

The lower bound is -inf and the upper bound is n.
The lower bound is achieved when only producers or consumers come in any number.
The upper bound is n as the sum can be increase only when a signal operation is executed on empty or full, but a signal operation is executed only after a wait operation, wait decreases the value of the sum by 1, signal increases it by 1.
This is achievable at the starting, when no consumer or producer has come yet, or when exactly n producers have come, or at any instant when the total number of processes in the queue are equal to n.

2. In the class we discussed the solution to the reader-writers problem using semaphores. We saw that this solution can cause the writer threads to starve. Please develop an alternative solution in which neither the reader nor the writers will starve. For this you can assume that the underlying semaphore queue is fair.

Sol.
Initialization

```
in = sem(1)
mutex = sem(1)
writer = sem(1)
counter = 0
```

Reader

```
wait(in)
wait(mutex)
counter++
if(counter == 1) wait(writer)
signal(mutex)
signal(in)

*critical section*
```

```
wait(mutex)
counter–
if(counter == 0) signal(writer)
signal(mutex)
```

Writer

```
wait(in)
wait(writer)

*critical section*

signal(writer)
signal(in)
```

Explanation:
in semaphore is used to prevent starvation, as soon as a writer is available and the remaining processes are if available reader are in critical section, the writer first calls a wait(in), thereby preventing any more readers to enter and waits until all the readers(the count is kept using counter variable) complete the critical section. The last reader calls a signal(write), thereby allowing writer to enter the critical section.
After the writer completes the critical section it signals write and in, to let the remaining processes execute.

mutex semaphore is used to prevent race condition by the concurrent operation of two threads on counter variable