Operating Systems - II: CS3523

Programming Assignment - III

Solving Producer Consumer Problem using Semaphores & Locks

Assignment Report

Sagar Jain - CS17BTECH11034

March 9, 2019

Contents

Salient Features of Program Design

The problem has been solved using semaphores and locks.

Semaphores

Semaphores have been implemented using the header ***semaphore.h***

1. I have used three semaphores, ***full***, ***empty*** and ***locker***.

2. ***locker*** is initialised as 1, ***full*** as 0 and ***empty*** as the capacity of the buffer.

3. ***full*** is used in the consumer thread to ensure that no consumer can consume if there are no full buffers and we increment it in the producer thread everytime something is produced.

4. ***empty*** is used in the producer thread to ensure that no producer can produce if there are no empty buffers and we decrement it in the consumer thread everytime something is consumed.

5. ***locker*** is used to ensure mutual exclusion between multiple producers and consumers.

6. **Algorithm**

   For Producer:

   ```
   sem_wait(&empty);
   sem_wait(&locker);
   // produce item
   sem_post(&locker);
   sem_post(&full);
   ```

   For Consumer:

   ```
   sem_wait(&full);
   sem_wait(&locker);
   // consume item
   sem_post(&locker);
   sem_post(&empty);
   ```

Locks

Mutex has been implemented using the header ***mutex***.

1. I have used one lock ***check_lock***. ***counter*** is used to keep the count of the number of filled buffer cells.

2. ***check_lock*** is used to ensure that no two processes read the same value of counter and update them leading to a race condition.

3. ***check_lock*** also makes sure that production and consumption does not overlap.

4. **Algorithm**

   For consumer:

```
while(true) {
        check_lock.lock();
        if(counter > 0) {
                counter --;
                break;
        }
        check_lock.unlock();
}
// consume item
check_lock.unlock();
```

   Similarly, for **producer**. (counter < capacity)

```
while(true) {
        check_lock.lock();
        if(counter < capacity) {
                counter ++;
                break;
        }
        check_lock.unlock();
}
// produce item
check_lock.unlock();
```

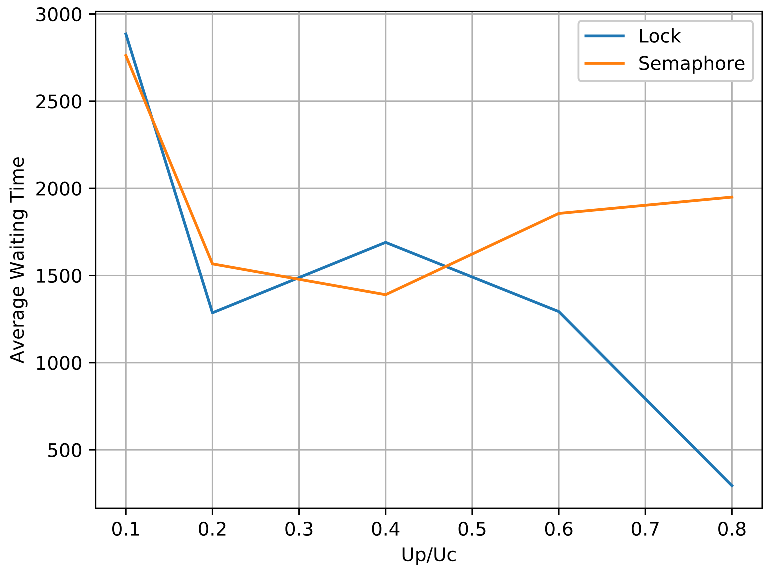
Program Output

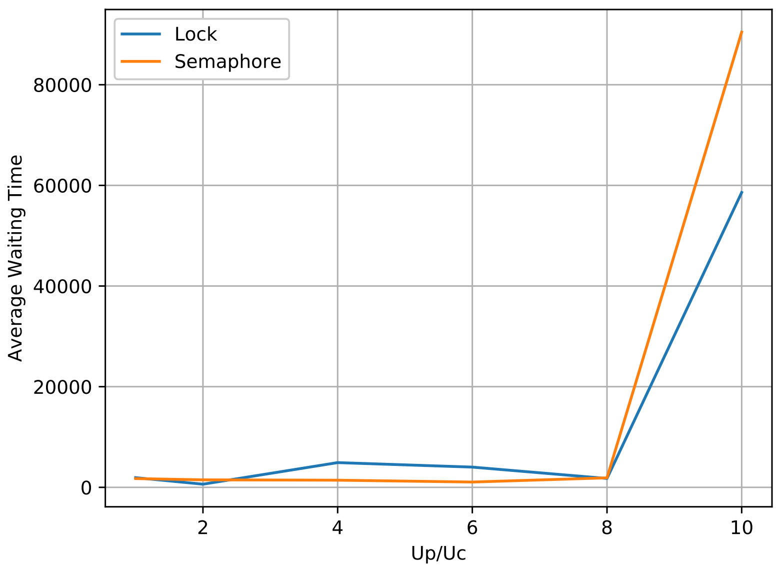
The programs output log files (*output-semaphore.txt*, *output-lock.txt*). These files have the data about the point in time at which any consumer or producr consumes or produces any item from or into the buffer. For Example:

14th item produced by thread 4 at 19:58:09 into buffer location 61
0th item read from the buffer by thread 0 at 19:58:09 from buffer location 60
1th item read from the buffer by thread 0 at 19:58:11 from buffer location 59

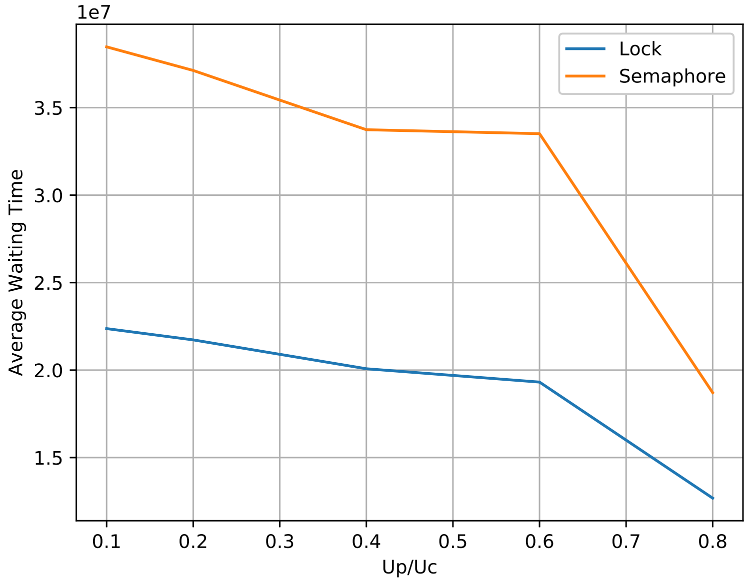
Results & Graphs

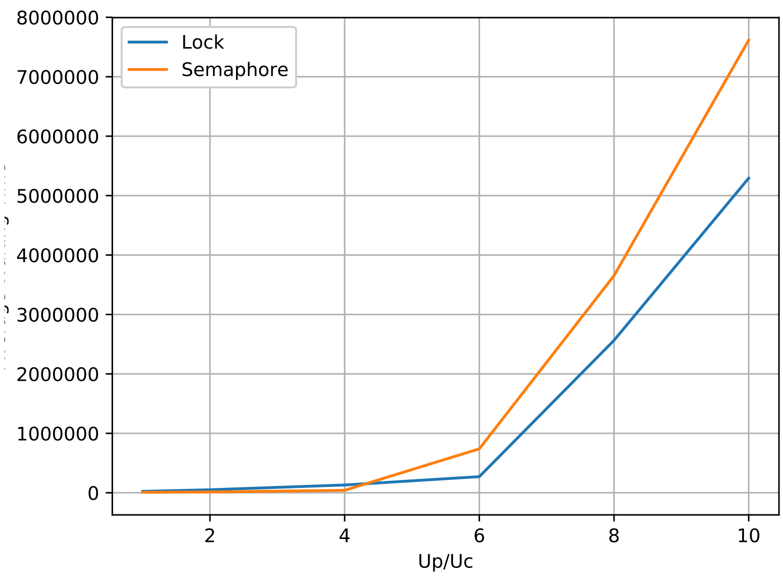
Producer Waiting Times vs Ratio of Mean Waiting Times





Consumer Waiting Times vs Ratio of Mean Waiting Times





Explaination of Results

1. From the graphs it is quite obvioius that locks perform better than semaphores.
2. This could be due to the fact that sleeping and waking a thread up is a very expensive task.
3. It is better to busy wait in this case.
4. It is visible that for large values the difference is clear between semphores and locks.
5. Busy wait is not suggested when there is only one core available.
6. One can use locks when there are multiple cores available, otherwise eventhough the semaphores perform worse sleeping of threads seems better than busy waiting.