9/20/2016 Distributed Operating Systems HW1

Nidhi Makhijani
3987-5129

Sol a. A few things that can go wrong when two processes are sharing access to the buffer.
1. The producer produces faster than the consumer can consume
In this case, the circular buffer will fill, and the tail will overtake the head and will overwrite the data.
2. The consumer runs faster than the producer
In this case, the head can overtake the tail, and the consumer will consume will consume garbage data from the buffer.
3. Consumer can start and try to consume before the producer has started producing
This is similar as case 2 above and it might be observed if consumer has higher priority than producer process.
4. Producer can start and complete before the consumer is started
This is similar as case 1 above and it might be observed if consumer has lower priority than producer process. This will result in producer waiting for consumer to then start before it can start again.

Sol b. A mutex can be implemented using semcreate(1). Such a mutex shall only have values 0 and 1 and can be used a lock. We can implement the mutex functions mutex_acquire and mutex_release by using semaphore wait and signal respectively. As this problem just has a single producer and consumer, we don't need to add any additional checks or conditions.

Sol c. Code is in main_mutex.c

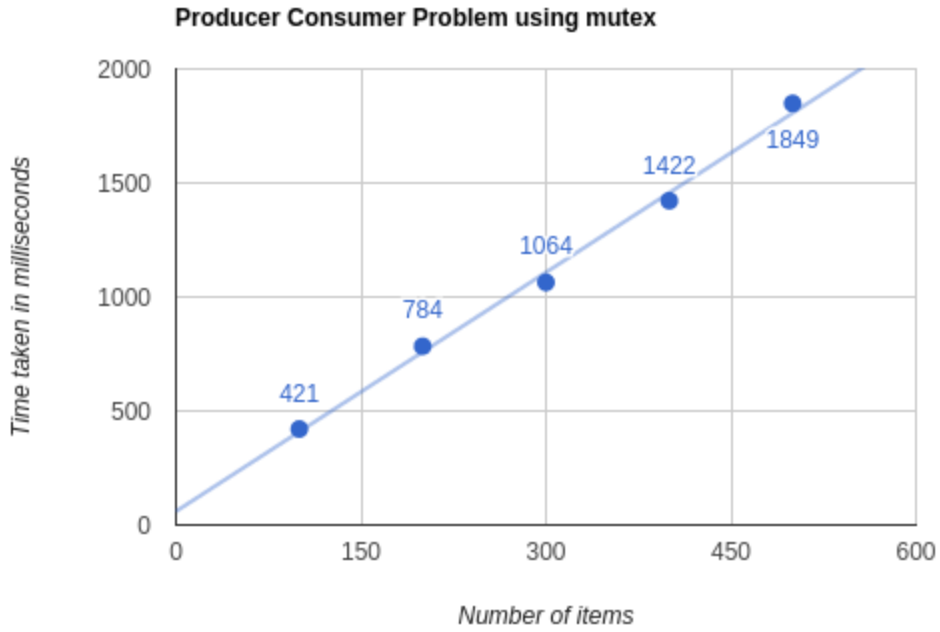
Sol d. Code is in main_sem.c

Sol e.

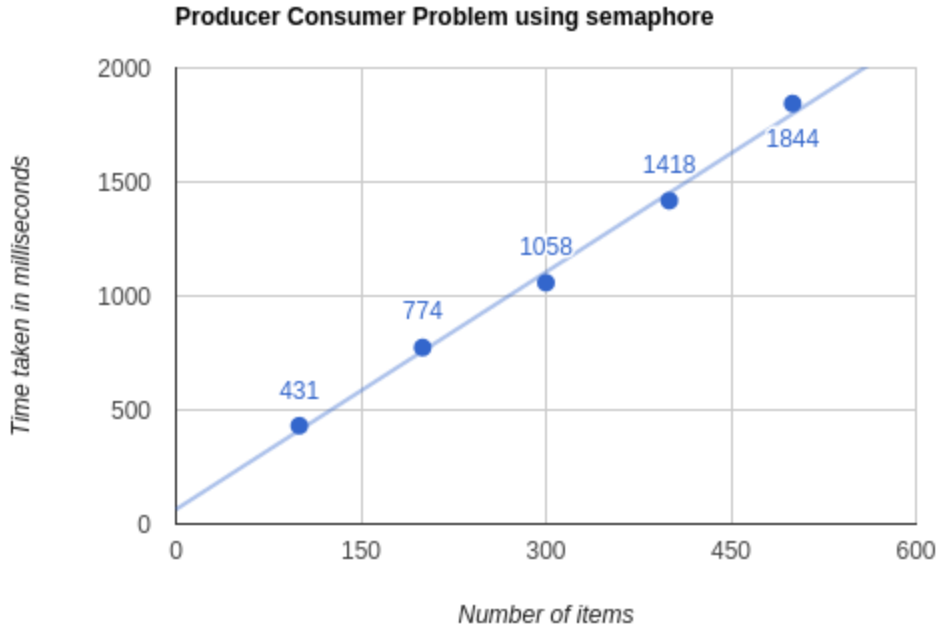
In the general producer consumer problem with multiple number of producers-consumers, the semaphore method would be more efficient. In the mutex method, processes are not waiting and are actively trying to get the processor after acquiring the shared_buffer_lock. After acquiring the shared_buffer_locak, they check if they can produce/consume and release the lock if they can't. This leads to waste of time as processes unnecessarily acquire the lock and waste time and processor.
In the semaphore method, if the shared buffer does not have space, all the producers would wait on the semaphore. Similarly all the consumers would wait if there are no items to consume. They are not competing with the other processes to obtain the lock unnecessarily.

Sol f.

Observe the graphs in the figure for the mutex method -



Observe the graph in the figure for the semaphore method-



We can see that the semaphore and mutex method is linear from the observed graph. It is because the semaphore and mutex methods take constant time to check and confirm if they can add items or remove items from the shared buffer. The only difference is in the number of items that have to be consumed or produced which is O(n) where n is the number of items.

Both the implementations are giving correct output. That is both are behaving in a the expected way and synchronising among themselves. We observe that they are not following our general hypothesis as in solution e. Out of the 2 implementations, the mutex method is taking lower time because in the single producer and single consumer problem, the semaphore operations(wait and signal) are taking more time than the simple boolean checks we are performing in the mutex method.

Conclusion

I have discussed the problem associated with the problem without the synchronisation methods in Sol. a. 2 methods that we could use are using semaphores(which also need a mutex for the shared buffer) and using mutex.

Both the solutions guarantee –

1. Producer blocks when buffer full
2. 2.Consumer blocks when buffer empty

Both the solutions have been implemented in sol c and sol d. In the solutions implemented we. we have seen that all the 4 scenarios that could arise if we don't use proper synchronisation concepts, have been covered and will not occur if we use the solutions. We can see and analysed their performance in sol. f.