CSE-3211: Assignment 1

Shahrear Bin Amin
FH-55

Al Mohammad Aladin
FH-59

October 22, 2018

Problem 1

In this problem there are 10 threads. Our task was to count upto 10000 using these 10 threads,but 10 threads all together counting wrong. The reason behind the threads' wrong counting is their unsynchronized access of shared resource which is the "counter" variable. Access of that global variable is needed to be mutually exclusive in order to prevent unexpected value increment.We have added a new binary semaphore (util) to give access only one thread in critical region at a time.In adder function from line 81 to 87 is critical region. We locked the critical region using "util" semaphore. Rest of the adder function can run parallely.

Problem 2

Maintaining concurrency was difficult because of accessibility of shared resources by multiple threads of similar or disimilar types. To resolve the issue regarding mutual exclusion several semaphores were used.

Here order_buffer is used for keep track of order and ready_buffer is used for keep track of serves.
Shared variable current_customers is used for looking how much customers are still working.
We used five semaphores:

1. **order_lock :** This semaphore is used to lock order_buffer when a customer's order is placing in order_buffer array & no other customer can place any order. Also no stuff can take any order from order_buffer at the same time.

2. **shipments_lock :** Semaphore shipments_lock is used to lock ready_buffer when a stuff is placing an orderd can mixed with specified tints in ready_buffer array .No customer can take any can while a stuff is placing a can in ready_buffer .

3. **customer_dec_lock :** When a customer is leaving then shared variable current_customers is decremented by 1. This is used to lock the shared current_customer variable.

4. **tints_sema[NCOLOURS] :** We didn't locked the all tints while a stuff is mixing in can. We just locked particular tints which is currently using by this semaphore. It increased parallelism.

5. **fill_order_lock :** When we are accessing tints_sema[NCOLOURS] there is posibillity of deadlock to avoid this we made this array exclusively accessible.

First customer order with a can with specified tint color, order is placed in order_buffer & it's mutually exclusive for customers. Customer waits untill this can is served to him. Then the stuff take order from order_buffer it's also mutually exclusive. Then stuff go to specefied tints mix them in can. Only using tint's are locked. Then when mixing is completed then can is placed in ready buffer.Customer takes the can.When a customers performed all iteration then it go home.

The program generates expected output except for the staff and corresponding number of tints served by them. As the staff thread run parralelly they finish their jobs almost simultaneously. As a result without mutual exclusion the kprintf method runs parralelly displaying seemingly scribbled output.