CSE3211: Operating System Lab
Assignment 1: Multi-resource Synchronisation
Design Document

Tanjid Hasan Tonmoy Sakib Hasan

1 Concurrent Mathematics Problem

The critical section in this problem is the increment of the counter variable which is shared among the threads. To ensure consistency, mutual exclusion should be maintained for the portion of code that increments the counter when the threads try to access that portion concurrently.

We have used a binary semaphore so that only one thread can enter the critical region at a time. Before a thread enters critical region it uses a **P()** or down operation on the semaphore count_mutex so other threads that try to enter the critical region are blocked. After executing the code in the critical region, the thread leaves critical region by using a **V()** or up operation on the semaphore.

2 Paint Shop Synchronization Problem

Two main tasks in this problem are implementing the functions in paintshop.c and making sure there is no synchronization issue. The activities in paintshop described in the instruction can be summarized in the following sequential steps

1. Customers put their orders in a buffer and wait for the order to become ready.
2. Staff takes an order from the order buffer and prepares that color.
3. After mixing color according to request, staff will put that in a ready buffer.
4. Customers take their requested color out of the ready buffer.

5. If a customer has no farther orders then the customer goes home.

6. Staff goes home if there is no customer left in the shop.

Step 1 and 4 are implemented in order_paint(), step 2 and 6 is implemented in take_order(), step 3 in fill_order() then serve_order(). These functions are executed by different concurrent threads. This gives rise to some problems with access to shared resources by different threads and we need synchronization to avoid deadlock and starvation. The functions open_paintshop() and close_paintshop() are used to initialize and cleanup the semaphores that were used for that purpose respectively. We have used two buffers, one for storing orders and the other for shipments when orders are ready. Number of remaining customers in shop are stored in a variable.

The critical sections are modification of the two buffers, accessing the tints in mix function and the variable representing number of remaining customers. We have used binary and counting semaphores as needed to deal with these issues.

The semaphores used are given in Table 1

	initial value
orderbuffer_full	0
orderbuffer_empty	buffer size
readybuffer_full	0
readybuffer_empty	buffer size
orderbuffer_mutex	1
readybuffer_mutex	1
tint_access	1
tints_mutex[NCOLOUR]	1
customer_no_mutex	1

Table 1: Semaphores used in paintshop.c

For synchronization, customer threads need to ensure that

- Order buffer has empty slots and that the thread has exclusive access for modifying the buffer.
- When checking if requested order is ready, ready buffer should not be empty.

Similarly, staff threads need to make sure

- Order buffer is not empty.
- The ready buffer has empty slot and that the staff thread has exclusive access to modify the buffer.

We should also make sure that other staff threads can mix if they do not require the same tints.

To accomplish these properties we have done P() operation on orderbuffer_empty before accepting an order from a customer thread to check whether the order buffer has empty slot and a P() operation on orderbuffer_mutex to control access to buffer before modifying the buffer and a V() operation after we were done with the modification. Then a V() operation on orderbuffer_full to indicate the increase in the buffer.

Similarly, We have used P() on readybuffer_full and readybuffer_mutex before checking ready order in the ready buffer. After entering this critical region, if an order is ready then a slot is cleared from the buffer and exclusive access is released by performing V() operation on readybuffer_mutex and readybuffer_empty to indicate more empty slots.

After a customer is done ordering, we decrease the number of remaining customers and control access to the variable while doing so with a semaphore.

The semaphore tint_access is used before threads lock a tint for their use, inside a loop we checked the requested colors and used P() operation for only requested tints in tints_mutex[NCOLOUR] and released tint_access afterward with a V() operation. So other threads may use the mix function if the do not require the same tints. After executing mix function, we release exclusive access of the tints used in mix function with V() operation.

P() operation on readybuffer_empty and readybuffer mutex used to check before serving an order. We check whether a can is ready using a for loop. If any can is ready then it is placed in the ready buffer. Exclusive access is released with a V() on readybuffer_mutex and on orderbuffer_full to indicate an increase in ready buffer.

3 conclusion

With the design that we have implemented, the synchronization problems should be resolved and any deadlock or starvation should not occur.