

OS Assignment 3

Aarushi Agarwal

2016216

Surabhi S Nath

2016271

Multiple Reader Writer with Synchronization

- **Description of your code and how you implemented the function – the logical and implementation details**

The code consists of a reader function and a writer function for performing read and write operations on the queue.

A queue data structure is maintained as a structure. The structure contains: front, rear, size, an array representing the queue. Code reference for implementing the queue is taken from Geeks for Geeks.

Functions for isFull(), isEmpty(), enqueue(), dequeue() and front element are made.

1. An array of semaphores allows each element of the queue to be locked. Hence the sem_t array of elementlock[] ensures the operations on an element are atomic
2. Every element has an associated read_count indicating the number of reader on every element
3. An array of mutexes sem_t mutex[] ensures read_count updates are not done simultaneously by any two readers. Mutex at element i denotes the lock on the read_count for the ith entry of queue
4. Since writes are to be atomic, ie, no two writers can enqueue elements into the queue simultaneously, therefore we keep another semaphore writer_check to lock the entire queue during enqueue and dequeue operations

The options implemented include:

1. The reader can read and random element from the queue and display it
2. The reader can deque an element from the front of the queue
3. The writer can enqueue elements into the queue

The reader chooses randomly between either reading an element at any index in the queue or the reader can deque the element at the front of the queue.

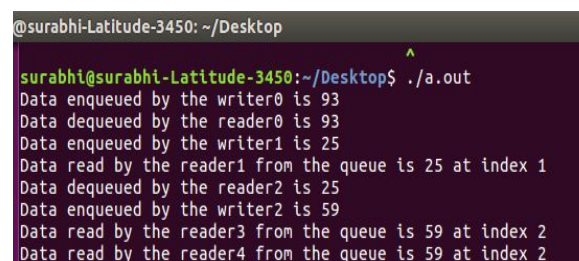
Also, The writer enqueues a randomly generated number into the queue at the rear end.

To ensure no reader can access any element while the writer writes, apart from acquiring lock on `writer_check`, every element of the array is also individually locked. Similar procedure is followed when reader dequeues front element.

Multiple reader and writer threads are created and all run parallelly. Twenty pthreads each for reader and writer are created and the respective `reader()` and `writer()` functions are called on them. Slight delay is introduced in order to facilitate interleaving.

- **Description of how to compile and test the program**

The make file is run for compiling the code. The executable `assignment3` is then run to execute the code. Print statements for readers and writers are put to indicate flow of the code. They show at each step when a reader or writer waits for a lock, when they enqueue, or dequeue or read elements and when the locks are released:

A terminal window with a dark background and light-colored text. The prompt is '@surabhi-Latitude-3450: ~/Desktop'. The user has entered 'surabhi@surabhi-Latitude-3450:~/Desktop\$./a.out'. The output shows a sequence of messages: 'Data enqueued by the writer0 is 93', 'Data dequeued by the reader0 is 93', 'Data enqueued by the writer1 is 25', 'Data read by the reader1 from the queue is 25 at index 1', 'Data dequeued by the reader2 is 25', 'Data enqueued by the writer2 is 59', 'Data read by the reader3 from the queue is 59 at index 2', and 'Data read by the reader4 from the queue is 59 at index 2'. A cursor is visible at the end of the last line.

```
@surabhi-Latitude-3450: ~/Desktop
surabhi@surabhi-Latitude-3450:~/Desktop$ ./a.out
Data enqueued by the writer0 is 93
Data dequeued by the reader0 is 93
Data enqueued by the writer1 is 25
Data read by the reader1 from the queue is 25 at index 1
Data dequeued by the reader2 is 25
Data enqueued by the writer2 is 59
Data read by the reader3 from the queue is 59 at index 2
Data read by the reader4 from the queue is 59 at index 2
```

These lines show how multiple readers can read the same data element at the same time

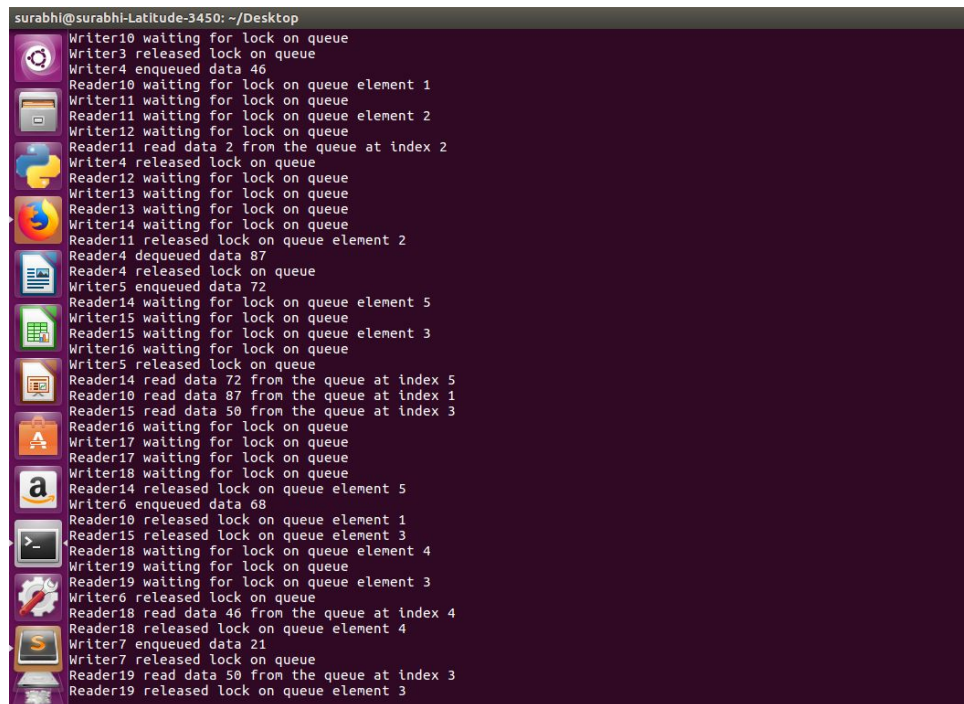
- **The inputs the user should give**

The user does not give any input in this implementation since user input would delay the processing and working of threads and we would not be able to clearly see the parallelization and locking mechanism in that case.

Thus all values to be enqueued are randomly generated and option for the reader to either read or dequeue is also obtained through a random variable according to which corresponding action is performed.

- **Expected output (and how to interpret it)**

The output is a detailed outline of the events performed at each step. Print statements track the threads executing, waiting, locking, releasing locks. Values and indices are also printed for sake of clarity and to able to track execution of threads efficiently.



```
surabhi@surabhi-Latitude-3450: ~/Desktop
Writer10 waiting for lock on queue
Writer3 released lock on queue
Writer4 enqueued data 46
Reader10 waiting for lock on queue element 1
Writer11 waiting for lock on queue
Reader11 waiting for lock on queue element 2
Writer12 waiting for lock on queue
Reader11 read data 2 from the queue at index 2
Writer4 released lock on queue
Reader12 waiting for lock on queue
Writer13 waiting for lock on queue
Reader13 waiting for lock on queue
Writer14 waiting for lock on queue
Reader11 released lock on queue element 2
Reader4 dequeued data 87
Reader4 released lock on queue
Writer5 enqueued data 72
Reader14 waiting for lock on queue element 5
Writer15 waiting for lock on queue
Reader15 waiting for lock on queue element 3
Writer16 waiting for lock on queue
Writer5 released lock on queue
Reader14 read data 72 from the queue at index 5
Reader10 read data 87 from the queue at index 1
Reader15 read data 50 from the queue at index 3
Reader16 waiting for lock on queue
Writer17 waiting for lock on queue
Reader17 waiting for lock on queue
Writer18 waiting for lock on queue
Reader14 released lock on queue element 5
Writer6 enqueued data 68
Reader10 released lock on queue element 1
Reader15 released lock on queue element 3
Reader18 waiting for lock on queue element 4
Writer19 waiting for lock on queue
Reader19 waiting for lock on queue element 3
Writer6 released lock on queue
Reader18 read data 46 from the queue at index 4
Reader18 released lock on queue element 4
Writer7 enqueued data 21
Writer7 released lock on queue
Reader19 read data 50 from the queue at index 3
Reader19 released lock on queue element 3
```

- **Error values and how to interpret them**

Errors handled include:

1. Queue overflow
2. Queue underflow
3. Reading from empty queue
4. Dequeueing from empty queue

- **References:**

- Geeks for Geeks
- <http://2k8618.blogspot.in/2011/02/readers-writers-problem-os-lab.html>