

Programming Assignment #1

COEN 283 Operating Systems
Department of Computer Engineering
Santa Clara University

Dr. Ming-Hwa Wang
Phone: (408) 805-4175
Course website: <http://www.cse.scu.edu/~m1wang/os/>
Office Hours: Tuesday & Thursday 9:00-9:30pm, Saturday 11:00am-12:00pm

Fall Quarter 2024

Email address: m1wang@scu.edu

<http://www.cse.scu.edu/~m1wang/os/>

Due date: 11:59pm October 13, 2024

Thread-Safe Shared Bounded Queue (200 points)

Please implement a shared and thread-safe bounded queue using Go or C/C++ with the pthread library. Your program should support the many-to-many or mailbox model by allowing multiple producers and consumers to enqueue and dequeue concurrently/simultaneously in the first-in-first-out (FIFO) order. Your program will run with input parameters (one per line) read from stdin in the order of 1) queue size, 2) number of producers, 3) the waiting time range (e.g., 4:6) in seconds for producers, 4) number of consumers, and 5) the waiting time range (e.g., 5:9) in seconds for consumers, and 6) number of produce/consume operations. When 3) is greater than 5), you are likely to see more underflows; otherwise, when 3) is less than 5), you are likely to see more overflows.

To test your program, assume the shared queue only supports putting/getting integer data to/from the queue and has a small queue size, and you need to run multiple producer/consumer threads.

E.g., to run your p1 with two producers (one generates odd numbers only and the other generates even numbers only) and two consumers (both dequeue from the shared queue), each producer or consumer has an infinite loop, each loop waits for a random time in the range [i.e., input 3) or input 5)] and then puts/gets data to/from the queue; then goes to the next iteration until the number of produce/consume operations [i.e., input 6)] is reached or the program is terminated by typing Ctrl-C (or by kill -9 command if your p1 runs in background). You can implement an unbounded shared queue first. Then enhance it to a bounded queue and support any scalar data type (e.g., floats, strings, etc.).

The output should show

- Which producer or consumer is putting/getting integer data to/from the queue with the size of the queue (e.g., product #0 puts 3 to the queue with 1

integer(s) in the queue, or consumer #1 gets 3 from the queue with 0 integer(s) in the queue).

- Show underflow (e.g., consumer #1 is waiting with queue size 0).
- Show overflow (e.g., producer #2 is waiting with full queue size 5)

Name:

ID:

Score:

Correctness and boundary condition (20%):

Whitespace convention compliance and error handling (5%):

Display all events including which producer/consumer, enqueue/dequeue with data, wait for queue is empty when dequeue, and wait for queue is full when enqueue, etc. (20%):

Avoid to use busy waiting (5%):

Handle overflow/underflow using wait/signal (20%):

Modular design, documentation (25%):

Automation (5%):

Subtotal:

Late penalty (20% per day):

Special service penalty (5%):

Total score: