CSC 452/552 Operations Systems

Project 2 The Classic Bounded Buffer Problem

Name: Brooklyn Grant

Bronco ID: 114142733

Date: 10/20/2024

1. Project Overview

   For the project, I implemented all of the functions in lab.h with C and POSIX threads. The objective of the project was to make a queue that would be accessed through multiple threads safer and efficient. I designed all the methods on my own, using mutexes and condition variables to handle the synchronization with the critical sections of the code. The project required creating functions for queue initialization, enqueueing, dequeuing, shutting down the queue, and checking if it was empty or shutdown.

2. Project Management Plan

   a) Task 1: Fork the starter repository

   b) Task 2: Insert the starter code

   c) Task 3: Implement the header file

      i. Task 1: Read the given articles to learn more about Threads in c and how I would implement this queue.
      ii. Task 2: Set up the node and queue structs so that we could access their properties for the methods.
      iii. Task 2: Implement the queue_init() function to initialize the queue with a given capacity, setting up the mutex and condition variables for the thread synchronization.
      iv. Task 3: Implement the enqueue() function to let producers add elements to the queue, and make sure that the queue does not exceed the capacity by using the pthread_cond_wait() function to block producers when the queue is full. Only let one consumer at a time.
      v. Task 4: Implement the dequeue() function, letting consumers remove elements from the front of the queue, while using condition variables to block consumers when the queue is empty. Only let one consumer at a time.

vi. Task 5: Implement the is_empty() function to check if the queue has no elements.
vii. Task 6: Implement the queue_destroy() function to properly free memory and resources associated with the queue, making sure all threads are notified before releasing the queue.
viii. Task 7: Implement the queue_shutdown() function, to shutdown the queue and notify all waiting threads so they can exit. Only let one consumer at a time.
ix. Task 8: Implement the is_shutdown() function to verify whether the queue is in the process of shutting down, so that threads can safely exit when needed.
x. Task 9: Test my results.

3. Project Deliveries

a) How to compile and use my code?

i. Run the 'make' command to compile the code. Run 'make check' to run the tests. Run 'make clean' to clean the repository.

b) Any self-modification?

i. I did not change any of the starter code. The only file I changed was the lab.c file to implement every function.

c) Summary of Results.

i. My program compiles and runs as expected. It passed all tests and all of my individual tests to make sure it avoided deadlocks.

4. Self-Reflection of Project 2

This project extremely helped my understanding of how to handle concurrency problem with something simple that I already knew- queues! I understand more about when to put a mutex and where I would put them in the methods. Also making sure to shutdown and destroy correctly with the mutexes and conditions. That was the most challenging part, was to shut down the queue correctly and learn what that actually meant and why I had to do it.