Munachimso Aneke

CS 470

Thread Synchronization Project

DISCUSSION

The program solves the Producer-Consumer problem using threads, unnamed semaphores, thread mutex lock, and a fixed buffer (5 elements). The main() takes the argument for the sleep time, number of consumer threads, and number of producer threads. Then, it initializes the buffer values, the semaphores, mutex, pointers (in and out), and buffer's index. All the items initialized are declared in the global scope as these are shared variables. Next, the main() creates the producer and consumer threads using a for loop, then sets the main thread to sleep for a user specified time before exiting. The producer and consumer threads manipulate the functions insert_item() and remove_item() respectively. The mutex lock is placed before the if statement that calls insert_item() (/remove_item()) in the producer (/ consumer) functions. This because the insert_item() (/ remove_item()) function accesses the critical section. Immediately after the if statement the unlock mutex is called. The semaphores are placed before and after the mutex lock and unlock calls, this is to prevent the producer from overwriting an item and the consumer from re-reading an item. If starting with the producer, the producer thread inserts a new item before signaling the consumer function.

QUESTION

1. The most difficult part was figuring out if I needed functions like pthread_join(), pthread_exit(), and if there was a needed for a pthread_attr_t since the attributes was default.
2. The easiest part of thread manipulation was creating the for loop for creating the threads.
3. The most difficult thread synchronization might have been writing the insert and remove item, because I was a bit confused how the buffer was working.
4. The easiest thread synchronization was creating and initializing the pthread mutex and semaphores.
5. I am not sure if the outputs should be ordered i.e alternating between producer output and consumer output. My output alternates in the beginning and changes weirdly at the end. If I can, I would love to change that. Also, I should have used a shorter name to save the main c file, because that added to my difficulty in creating the makefile.
6. In the makefile, I originally called -lpthread, "gcc -Wall -lpthread -o thread…." which lead to error claiming that there was an unreferenced call to sem_wait(), and sem_post() functions. I later realized I had to put -lpthread at the end of the compiler call: "gcc -Wall -o thread… -lpthread"