Lab 3 - Producer Consumer

Problem Description

The objective of this lab is to use pthread mutexes and condition variables to synchronize threads. The lab involves editing the two thread functions *producer()* and *consumer()* to produce functionality for them to communicate with each other through a shared buffer.

There are two pthread types used which are *pthread_mutex_t* and *pthread_cond_t*. The first type is a binary semaphore which provides mutual exclusion to the critical section using *pthread_mutex_unlock()* and *pthread_mutex_lock()*. The second type is a condition variable that is like condition queues in monitors. It provides functionality for a condition variable to wait using *pthread_cond_wait()*. It can also signal for a condition variable to wake up the first process that is waiting on the condition queue using *pthread_cond_signal()*. Lastly, it can wake up all of the processes waiting on the condition variable using *pthread_cond_broadcast()*.

The *producer()* function opens a data file, enters a loop that reads lines from that file one at a time, and then converts it to a number stored in an integer variable using *atoi()*. Before adding the value to the shared buffer, it enters a critical section and locks the mutex. It then checks to see if the buffer is full and must wait if it is. Likewise, if the buffer is empty, it should signal and resume the next process in the queue. After that, it proceeds to add the integer value to the buffer and increments the number of producers and total elements by 1. It has now reached the end of the critical section, so it unlocks the mutex and exits the loop. Outside of the loop, it decrements the number of producers and if it's the last producer, it will broadcast that it's empty. Lastly, it closes the file and returns NULL to exit the function.

The *consumer()* function loops and reads from the shared buffer, and opens/writes to a new output file. Just like the *product()* function, it enters the critical section and locks the mutex before it reads the value. If the buffer is empty and there are producers, it must wait. If the buffer is empty and there are no producers, then unlock the mutex and break out of the loop. If only the buffer is empty, signal to resume next process. It then reads the value from the buffer, unlocks the mutex, and exits the loop. Lastly, it closes the output file and returns NULL to exit the function.

The program is compiled through the make command "./main testNum numProducers numConsumers". For each input file of a producer, it will generate one output file for each consumer. In the code, there is also a function called *simulate_interrupt()* which calls the function *sched_yield()* with a probability of 33%. This creates a context switch if one of the other threads can run. If all threads are waiting, it does not have any effect.