ELEC 377 Lab 3 Program Description

Gia Lee, Valerie So, William Robson

Program Description:

This program implements a Producer-Consumer problem to enable multiple producer threads to read from input files and multiple consumer threads to read and write to output files using a share buffer. To ensure proper synchronization and avoid data races, the program utilizes various features from the pthread library, such as mutexes and condition variables.

To enable this, there are 4 functions in use:

1. Main function

   The main function initializes variables. Then, it creates the producer and consumer threads and returns after completion.

2. Producer function

   The producer function is responsible for reading data from input files and adding it to the shared buffer. It uses mutexes and condition variables for synchronization, ensuring that the buffer does not overflow. It also updates the numProdRunning variable and signals the consumer threads accordingly.

3. Consumer function

   This function consumes data from the shared buffer and writes it to output files. It again utilizes mutexes and condition variables for synchronization to ensure that the buffer is not emptied by the consumers if producers are still running.

4. Simulate_interrupt function

   This function simulates interrupts in the program.

Special Features Used:

The special features used in lab 3 was the inclusion of the pthread.h header file. Pthread.h is a header file that creates, manages and provides multilevel threading and threading synchronization and provide mechanisms for thread synchronization.

Multilevel threading involves creating a hierarchy of threads where higher level threads manage lower level threads. The function pthread_create() is used to create a new thread in the calling process and pthread_join() to join the threads. As C is a low-level programming language, it has finer control over memory, threads and synchronization which is essential for implementing multilevel threading for scenarios such as producer and consumers.

Thread synchronization is used to ensure that threads cooperate and communicate efficiently. Pthreads provide synchronization mechanisms such as mutexes, and semaphores. Functions such as pthread_mutex_lock() and pthread_mutex_unlock() are used to protect the access to the producer

running ensuring that only one thread is being modified at a time. The function pthread_mutex_lock() is used to acquire a lock on the mutex to ensure that only one thread is to enter critical section at a time. When a thread calls the lock function on a mutex, it checks whether the mutex is currently locked by another thread. If the mutex is not locked, the calling thread successfully acquired the lock and proceeds with executing the critical section. If the mutex is locked, the calling thread is blocked and waits until the mutex becomes available again. The function pthread_mutex_unlock() is used to release the lock on a mutex, allowing other threads to acquire it and enter the critical section. After a thread is finished executing the code within the critical section and no longer needs to lock the mutex, the unlock function is called to release the mutex and allows other threads to acquire the lock.