Program Description:

The producer program creates random integers to put into a sharedMem data type, table, for the consumer program to consume. Using semaphores, the producer checks for if the consumer is done consuming data in the shared memory before adding more produced data to it. The consumer then waits for the producer to signal it is done in the shared memory before consuming the data added there. A mutex lock initialized as a semaphore guarantees mutual exclusion, as when one program enters shared memory, the other must wait for the lock to be released before entering. There are also semaphores to check whether the producer/consumer memory is empty/full and signal when the program can continue based on that.

Executed results should look like:

item produced: 651

item consumed: 651

item produced: 49

item produced: 2182

item consumed: 49

item consumed: 2182

Usage instructions:

The program uses Linux-specific libraries and therefor only runs on a Linux operating system. Ubuntu is the operating system I chose to develop this project. This program should be able to compile and run using the Makefile commands “make producer,” “make consumer” and “make run” specified in this projects’ Makefile, however standard g++ commands can be used.

Explanation of Key Components:

Within this program, semaphores are used to guarantee mutual exclusion through three variables in the sharedMem data structure. The three key variables are “full,” “empty” and “mutex,” which individually, the full semaphore signals to the producer to stop production when the memory is full, the empty semaphore signals the consumer to stop consumption when there is no more data in the shared memory table to consume, and the mutex lock guarantees mutual exclusions by signaling the producer and consumer programs to wait when one is accessing the shared memory.

Examples and results:

While working on this project, I learned a lot about different libraries I had never used before, and Linux as an operating system. This project helped solidify my understanding of the producer/consumer problem, and the amount of overhead needed to implement it. Overall, I feel like I learned more about the structure of operating systems as a whole.

Example results:

item produced: 49323 🡨the “empty” semaphore is triggered prior

item produced: 55414 🡨 the “full” semaphore is triggered

item consumed: 49323 🡨FIFO, semaphore signals producer is out of memory

item consumed: 55414 🡨the “empty” semaphore is triggered

item produced: 19332 🡨consumer is finished in memory, semaphore signaled

item consumed: 19332 🡨producer finished producing, semaphore signaled