CS3401 : Operating Systems

Project 3:

Producer Consumer Problem

Mohamed Eliskandrani

900140998

**Introduction:**

The aim of the project is to simulate the producer consumer problem using pthreads. The program should create a thread for each consumer/producer and synchronize their actions. The producer should not add to the buffer if the buffer is full, and no 2 producers/consumers are to access the buffer at the same time. This is to be implemented using semaphores.

**Code:**

The main function takes 3 variables (amount of time the program will run for, number of producers, number of consumers) and makes use of the buffer.c and buffer.h files for cache implementation (removal and insertion). It then initializes the global semaphores full and empty to 0. The program then creates a thread for every producer and consumer to run their respective functions and sleeps for the amount of time specified at the initialization of the program. The producer function generates a random number and calls the function insert\_item.

Insert\_item first locks the semaphores empty and mutex. Both insertion and removal of items from the buffer require the calling of the function pthread\_mutex\_lock which locks mutex if mutex is unlocked and blocks the thread if it is previously locked until mutex becomes available. It then inserts the item into the buffer, increments count by 1 and sets the position of the next item to be added. This is done as a circular queue where insert\_item increments the variable in as follows: in=(in+1)%BUFFERSIZE. The function then unlocks the semaphores mutex and full and returns the success state of the operation.

Remove\_item first locks the semaphores full and mutex. It then removes the item from the buffer, increments count by 1 and sets the position of the next item to be added. This is done as a circular queue where remove\_item increments the variable out as follows: out = (out+1)%BUFFERSIZE. The function then unlocks the semaphores mutex and empty and returns the success state of the operation.

If a producer attempts to produce an item while an item is being consumed it will first lock the semaphore full and attempt to lock mutex and fail causing it to be blocked.

If a consumer attempts to consume an item while an item is being produced it will again be blocked until mutex becomes available.

The purpose of the full and empty semaphores is to prevent the function from locking mutex when the buffer is either full or empty and causing a deadlock. Every time we insert an item into the buffer we call sem\_wait(empty) which prevents us from adding items to a full buffer (since we call sem\_post(empty) when removing an item). Every time we remove an item from the buffer we call sem\_wait(full) which prevents us from removing items from an empty buffer (since we call sem\_post(full) when inserting an item).

**Test cases:**

If we shift the order of locking mutex and empty and full we would arrive at a deadlock once the buffer is full and a producer tries to insert or if the buffer is empty and the consumer attempts to consume.

If we initialize the function to have much more producers than consumers, there becomes a very large wait time due to the buffer being full most of the time and producers being blocked.

If we initialize the function to have much more consumers than producers, there becomes a very large wait time due to the buffer being empty most of the time and consumers being blocked.

If we decrease the buffer size the wait time also increases as the buffer becomes full or empty frequently.

The best situation under which the program operates is when there are roughly the same number of producers as consumers and the buffer size is large.