CSC332G LAB Task 5 Due March 28

In this folder are a number of code examples illustrating:

(1) Traditional shared-memory with semaphore management of the producer-consumer problem: ProducerConsumer.c
(2) A shared memory example which does not fork or create a thread but assumes each of 2 processes knows the name of the shared memory segment. This is implemented in posix API shared memory via memory mapped file. shm_msgclient.c, shm_msgserver.c
(3) A shared memory example from the textbook which forks a child and uses posix shared memory but does not use posix threads. The parent forks 32 children.
(4) Another example from the textbook which creates 50 posix threads which play with the value of a passed parameter, but which does not use shared memory.

Your Task 5 is to take these code examples and implement a producer-consumer solution which uses both:

Posix shared memory

Posix threads.

Task 5 A. Implement the producer-consumer in such a way that the parent is the producer and the thread is the consumer and the parent and thread update the values of a queue in the parent's memory. Are there race or deadlock conditions with this solution? Would you need to add a semaphore or mutex to protect the memory object?

Task 5 B. Implement the producer-consumer in such a way that the parent creates a shared memory segment using posix API (memory mapped file) and creates a producer thread with one routine to run, and a consumer child with another routine to run. Pass the name of the shared memory file as an argument to each thread. Does this solution require a mutex or semaphore, or does the file implementation take care of mutual exclusion? Can the producer and consumer threads write to the file at the same time?

Please take note:

UNIX/Linux header files required by gcc vary from one Linux to another. For example, programs that compile clean on the Debian Linux prepared for the OS book throw errors and warnings on Ubuntu in the lab!

The sample programs I gave you all compile clean or with minimal warnings on Ubuntu in the lab. I had to add include files in several cases. Still warnings with the format expressions in printf statements eg printf ("%d%t….

Also, when compiling with posix API functions you need to use the compile flag –lpthread, when using shared memory –lrt, and when using math functions such as max(), mod() and rand() use –lm

It is best practice (and sometimes necessary to get a compile) to put the flags at the end of the gcc statement eg.

gcc myfile.c –o myfile –lpthread –lrt –lm