Linux Assignment - 4

1. a) Create a shared memory object of 3 page size and attach current process to the shared-memory object.
   Now, write some data into pages of this shared memory region.
   Create a child process from this process and access the shared memory section in the child process – are you able to read the data that was written in the parent process ? You must be able to access every page of the shared memory region.

Note: you can create a shared memory region of any size as per your Requirements !!! Here, we are just using one example !!!

   b) can you repeat the above steps using 2 unrelated processes – meaning, the 2 processes must not be parent and child processes. Are you able to write data from one process into the shared memory section and read the same data from another process ? Once again, you must be able to access every page of the shared memory region.

Note: in this case, you must load 2 separate programs from 2 terminals - this will ensure that we are communicating between 2 unrelated processes !!!

Linux Assignment – 4.....

2. In the following problem, use the sample code provided in race.c !!!
   a) prove that race conditions can occur and lead to inconsistencies in computing !!!! you may need to modify the code to generate race conditions and inconsistencies !!! say, increase the number of Iterations !!!

Note: as mentioned in the class lectures, race conditions may not occur every second, millisecond, or every 10 years !!!! concurrency related problems are unpredictable !!!!

Note: you must test in uniprocessor environment and then on multiprocessor environment as well !!!

Note: you must understand your execution environment and scheduling aspects to generate race conditions – on hint could be changing the nice values of processes to reduce their time-shares – in addition, you must also load the system with while.c programs to share the cpu cycles among processes !!!

b) now, using a binary semaphore(whose value is fixed as 0 or 1) as a lock to protect critical sections in the above problem, prove that semaphore lock can prevent race-conditions and prevent inconsistencies in computing !!!

Note : prove this works in uniprocessor and multiprocessor environments !!!

linux Assignment – 4...continued

3. look into prod_test.c, prod_1_1.c , prod_1_2.c , cons_test.c , cons_1_1.c and cons_1_2.c examples -

   a) prove that you need 2 counting semaphores for synchronization, in these cases – try to test it practically with/without counting semaphores !!!

   b) do you need a critical section semaphore ? Why do you need it ??

linux Assignment – 4...continued

3. c) finally, rewrite the above set of examples such that you can do the following:
    - create a shared memory segment as needed in the parent process
    - create and initialize a semaphore object with necessary semaphores (may be, 3 semaphores)
    - create 2 child processes – one child will be producer and one another will be consumer
    - producer will read a string and write into shared circular buffer area of strings, not characters – meaning, each circular buffer slot will hold a string, not a character as in the sample codes !!!
    - parent process must create the children processes, wait for the children processes to terminate, clean-up the children processes, destroy shared memory, destroy semaphore objects and then terminate