Linux Assignment - 3

1. a) Create a shared memory region of 3 page size and attach to it in the current process.
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.

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.

Linux Assignment - 3

2. In a parent process, create a semaphore object with just one semaphore. Initialize the semaphore to initial value 0. Do a decrement operation in the child process and an increment operation in the parent process.

   what do you observe ?

   In addition to the above, print the semaphore value(by using semctl()) before decrementing the semaphore and after the decrement operation is completed.

   Also, print the semaphore value (by using semctl()) before incrementing the semaphore and after the increment operation is completed.

   What do you observe ?

linux Assignment – 3...continued

3. look into prod_test.c, prod_1.c , prod_2.c , cons_test.c , cons_1.c and cons_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 ??
   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
      - 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
      - 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