CS69201: Computing Lab-1
Systems Assignment 3: Inter-Process Communication
Deadline: October 16, 2024, 5:00 PM

Important Instructions:

1. Using Linux

a. This assignment is based on OS programming and will be beneficial for the students to do this assignment in a Linux OS.
b. If you don't have Linux, use Virtualbox + Linux image, which are both free. We suggest Ubuntu but feel free to use any Linux version. If you decide to use Cygwin or wsl, it's up to you, just remember that most of your batch mates (and the instructors) will use Linux (and Ubuntu), so, naturally, we might not be able to provide support for other systems.
c. In case of any difficulties in understanding any functionalities, you may use the same functionality in a Linux Terminal to get a better understanding of how it works.

2. Programming language: This assignment will use the C programming language.

3. Error handling & input-output

a. A proper error handling (e.g., when the syscall or the library calls fail) is expected in this assignment.
b. The inputs should be taken from the Terminal and results displayed in the Terminal, if explicitly not mentioned.

4. Deliverables: Submit a zip file named ***<your_roll_no>_A11.zip***

a. It should include all .c files *in the format <your_roll_no>_A11_T.c*
b. An Additional readme file (name it "*README.txt*") containing details about how to compile the files and any additional information required for the instructor during checking that you feel necessary. We will follow this file verbatim to compile and run your code while grading. **[Note: No marks without this]**

Description :

In the last assignment, we implemented codes for handling signals [SIGINT] and also implemented codes for handling concurrent threads to achieve synchronization by performing operations on a shared matrix. Today, we try to achieve synchronization on the same problem like last day but we use processes instead of threads.

Task: Using Semaphores for Handling Concurrent Processes and Achieving Synchronization

[60]

Write a C program which uses mutexes and semaphores to achieve process synchronization in a shared memory simulation. It consists of main processes (Default code process), order processes and chaos processes as discussed below.

Main Process :

1. The main process takes two positive integers, m and n, as inputs.
2. Create a shared memory and create a two-dimensional matrix M(m,n) in that memory where M is a matrix of size m x n.
3. Fill each element of the matrix M with a random integer in the range [1,1000].
4. Print "Random Matrix M of size (m,n) is created in shared memory." where m,n will be the dimension of matrix M.
5. Print all the values of the matrix M (in m x n format) in the main process.
6. Create two static integer arrays A and B of size 1000 in the shared memory.
7. Print "Shared Arrays A and B are created".
8. Spawn 6 (six) "Order Processes" and 3 (three) "Chaos Processes".
9. Print "I am order" and "I am chaos" on the successful creation of order process and chaos processes respectively.
10. The order and chaos processes will perform their operations which will be discussed below.
11. After all the order and chaos processes operations are done, print the final matrix M (in a m x n format) in the main process, destroy the shared memory and terminate the program.

Chaos Process :

1. The chaos process will choose a random cell from the matrix M and update the value with a random number from 1 to 1000. That means you have to take three random integers i,j,k where i, and j are the cell row and element numbers respectively and k is the random element with which the cell needs to be updated i.e. M[i][j] = k.
2. It will print "Chaos: Updated element at cell i x j with value k." (print appropriate values of i, j, k in the printed message).
3. The chaos process will store the changed row number i in array A in some index. The corresponding index of array A in array B will be initialized with macro UNPROCESSED. (Just an example: use #define UNPROCESSED and initialize with value -1)
4. After each change, the chaos process will sleep for two seconds.
5. After 30 such updates, the chaos process will print "CHAOS PROCESS ENDS" and exit.
6. Ensure that two (out of three) chaos processes can work concurrently in the matrix given they are not working on the same element.

Order process :

1. Each of the 6 order processes will run the same code. Details start below.
2. The order process will check the shared arrays A and B for new unprocessed entries. If there is any, then one order process will lock the shared arrays A and B and read one changed row number. Mark it with a macro PROCESSED and then unlock the arrays A and B.
3. It will print “Order: Detected updated element at row i” (Print the appropriate value of ‘i’ in the printed message)
4. Next, an order process will take the changed row in the 2D matrix M, sort the changed row in non-decreasing order and unlock the row of M.
5. It will print “Order: row i is sorted now” (Print the appropriate value of ‘i’ in the printed message).
6. It will then print “older row i: ” [row values before sorting, space separated] (Print the appropriate value of ‘i’ in the printed message).
7. It will then print “new row i:” [row values after sorting, space separated] (Print the appropriate value of ‘i’ in the printed message).
8. Then the order processes will check the shared arrays A and B for more unprocessed elements.
9. After processing thirty such updates, order processes will print “ORDER PROCESS ENDS” and exit.
10. Ensure that five of the six order processes can work concurrently on A, B and M when they are working on different indices.