CS69201: Computing Lab-1
Systems Assignment 2: Handling Signal Processes, and Mutexes and Semaphores
Deadline: October 15, 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>_A10.zip***

a. It should include all .c files [2 files for the 2 problems] *<your_roll_no>_A10_T1.c, <your_roll_no>_A10_T2.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 :

Till now, we have implemented the threading approaches in C/++. basic Pthreads, blocking std::threads, a non-blocking solution using C++20 coroutines and lock-free programming. Now, we move on to something more. We shall now deal with handling signals, mutexes and semaphores in C. This assignment is divided into two tasks each of which is discussed below.

Task 1: Handling Signals [15]

Write a C/C++ program which executes a while loop and waits for an input character to be entered by the user in the Terminal. For any input, it does the following :

1. If the input is an uppercase or a lowercase alphabet (except "x" (ASCII: 120)), the code just prints the entered input character followed by a new line [*see Sample Run for more*] and continues to take input from the user.
2. If the input is not an alphabet, the code prints "Do you speak my language?" and continues to take input from the user.
3. If Ctrl + C is invoked [Ctrl + C is a keyboard shortcut in Terminal which invokes a system interrupt signal (SIGINT) to the process which forcefully stops the running process], the code prints "I am unstoppable!" and waits for the next input.
4. If the input is "x", the code terminates after printing "Valar Morghulis" followed by a new line.

Sample Run :

```
> gcc <your_roll_no>_A9_T1.c -o problem1
> ./problem1
> a                          // Input
> a                          // Outputs a followed by a new line and takes input as in the line below
> 1                          //Input
> Do you speak my language?  //Output
> [press Ctrl + C on keyboard] //Input
> I am unstoppable           //Output: Note your code should not terminate on the above input!
> x                          //Input
> Valar Morghulis            //Output
//Program Terminates
```

Submission Format :

Save your code in a single file as <your_roll_no>_A10_T1.c.

Task 2: Using Mutexes and Semaphores for Handling Concurrent Pthreads and Achieving Synchronization [25]

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

Main Thread :

1. The main thread takes two positive integers, m and n, as inputs.
2. Create a two-dimensional matrix M(m,n) 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." 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 thread.
6. Create two static integer arrays A and B of size 1000.

7. Print “Shared Arrays A and B are created”.
8. Spawn 3(three) “Order Threads” and 1(one) “Chaos Thread”.
9. Print “I am order” and “I am chaos” on the successful creation of order threads and chaos threads respectively.
10. The order and chaos threads will perform their operations which will be discussed below.
11. After all the order and chaos thread operations are done, print the final matrix M (in a m x n format) in the main thread and terminate the program.

Chaos Thread :

1. The chaos thread 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 thread 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 thread will sleep for two seconds
5. After 30 such updates, the chaos thread will print “CHAOS THREAD ENDS” and exit.

Order Thread :

1. Each of the 3 order threads will run the same code. Details start below.
2. The order thread will check the shared arrays A and B for new unprocessed entries. If there is any, then one order thread 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, the order thread will lock 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 thread will go on checking the shared arrays A and B for more unprocessed elements.
9. After processing thirty such updates, order threads will print “ORDER THREAD ENDS” and exit.

Submission Format :

Save your code in a single file as <your_roll_no>_A10_T2.c.