CSC 415 Operating Systems Principles
Summer 2018, Anthony J Souza

HW #4. Processes, Threads & Race Conditions

Assignment (60 points). For this assignment you will create a group of threads and then try to coax them into producing ***as many race conditions as possible***. First, your program should declare a single shared integer variable (i.e. a global variable) with initial value zero; then your program should spawn off the number of threads specified in the source file using the MAXTHREADS defined constant. We will use a for-loop to create these threads. We will designate threads created on even numbered iterations as adders and threads created on odd iterations as subtractors.

Adder Threads: (Adding threads WILL have their own thread function)
Adder threads will execute the following steps:

1. Read global value and store it into temp
2. add 1 to the temp value
3. store temp value into global variable
4. print the following to the console:
   "Current Value written to Global Variables by thread : tid is val"
   Were tid is the thread id and val is the value of the temporary variable AFTER adding 1 to it.
5. repeats steps 1 through 4 for MAXITERATIONS (this is a defined constant in the given source code).

Subtractor Threads: (Subtracting threads WILL have their own thread function)
Subtractor threads will execute the following steps:

1. Read global value and store it into temp
2. subtract 1 from the temp value
3. store temp value into global variable
4. print the following to the console:
   "Current Value written to Global Variables by thread : tid is val"
   Were tid is the thread id and val is the value of the temporary variable AFTER subtracting 1 from it.
5. repeats steps 1 through 4 for MAXITERATIONS (this is a defined constant in the given source code).

Once all threads have finished, please print the value in main in the following way:
"Final Value of Shared Variable : val"

- Where val is the value stored in the shared variable.

Since these threads are unsynchronized, there is a possibility of race conditions occurring, depending on the actual interleaved execution order of the various threads. ***Insert nanosleep() commands into your thread code to induce as many race conditions as possible by forcing an (in)appropriate interleaved execution sequence.*** A timespec struct has been given to you. You can change the values in the struct. The first value is the amount of time to sleep in seconds and the second value is the amount of time to sleep in nanoseconds. Modify these values to try to get the final value the furthest away from 0 zero as possible in either the negative or positive direction. As your program executes, it will produce a trace of output from

CSC 415 Operating Systems Principles
Summer 2018, Anthony J Souza
each thread showing the actual interleaved order of execution – Where you place the ***nanosleep()*** call will affect the amount of race conditions that occur and how much your print statements are interleaved. ***Implement your solution using Pthreads on Linux***

What to submit:

1. Please fill in readme.txt file given. This includes the following:
    - Name Class and Date.
    - Build and run instructions
    - Then please explain why your code produces the wrong output.
2. Push all completed code to your given repository by the deadline.

How to submit:

- To submit your work please do the following:
    - git add .
    - git commit -m “message”
    - git push