**CSE-381: Operating Systems**

**Exercise #6**

Max Points: 10

|  |
| --- |
| **Objective**: The objective of this exercise is to:   * Operate with C++ threads * Observe and control threads externally via Linux tools. * Explore race conditions in multithreaded applications.   **Submission**: Save this MS-Word document using the naming convention *MUid*\_Exercise6.docx prior to proceeding with this exercise. Upload the following at the end of the lab exercise:   1. This MS-Word document saved with the convention *MUid*\_Exercise6.docx. 2. Program developed in part #1 of the exercise named with the convention ex6\_1.cpp. 3. Program developed in part #2 of the exercise named with the convention ex6\_2.cpp. 4. Program developed in part #3 of the exercise named with the convention ex6\_3.cpp.   You may discuss the questions with your instructor. |

|  |  |
| --- | --- |
| **Name:** |  |

# Preliminary Task

If you are using your own Linux installation for this exercise, then it is your responsibility to ensure that you have the GNU debugger and necessary packages installed on your Linux distribution prior to proceeding with this exercise.

# Part #1: Develop a simple multi-threaded C++ program [4 points]

*Estimated time to complete: 35 minutes*

**Background**: C++ ties into the native threading infrastructure supported by various operating systems, such as: Native POSIX Threading Library (NPTL) on Linux or Threading Library on Windows™. In C++ any function (aka static method) or a member method in an object can be directly run as a thread. Furthermore, in C++ the language uses only one main thread and does not automatically create additional background threads. Consequently, the net background threads reported by the operating system is consistently the same as the number of threads created by the program.

**Exercise**: The objective of this exercise is to create a simple multi-threaded C++ program by implementing the following functionality in the main method:

1. Thread Count: Uses the first command-line argument (if any) to indicate number of threads. If this value is not specified, then the number of threads is assumed to be 5.
2. Sleep Duration: Uses the second command-line argument (if any) to indicate the duration time for which the threads must sleep (for conducting various tests in the second part of the exercise). If a second command-line argument is not specified then this value is assumed to be 2400.
3. Use the getpid() method (defined in OS-specific header unistd.h) to print the PID of the process.
4. Create the specified number of threads to run the given timer method with the following arguments:
   1. The first argument to the timer method is the Sleep Duration.
   2. The second argument is a logical index number for the thread, starting from zero.
5. Wait for the threads to finish running.

**Sample Output**:

|  |
| --- |
| C++ process PID: 17831  Starting up 5 threads.  2397 seconds left... |

# Part #2: Developing a data parallel multi-threaded program in C++ [3 points]

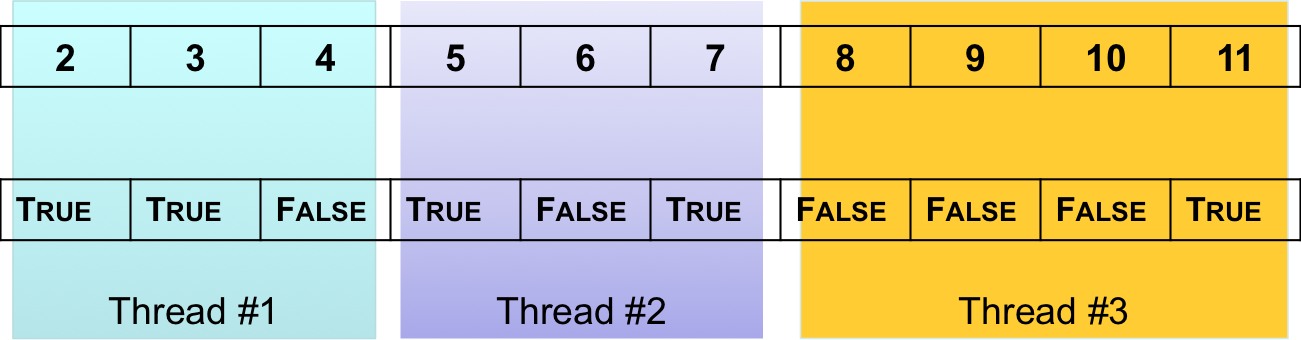
*Estimated time to complete: 30 minutes*

**Background**: Programs that do not manipulate the same regions of memory from different threads do not require any synchronization between threads to accomplish their task. Such programs are typically classified as data parallel programs and are excellent candidate problems to which multithreading can be efficiently applied.

**Functionality**: The objective of this exercise is to translate a simple prime-check C++ program into a corresponding explicitly multithreaded C++ program. The program is attempting to accomplish the following functionality:

* Given a list of numbers (say numberList) use thrCount number of threads to determine if each one of the numbers is prime or not prime.

For example, given 10 numbers (2 through 11) and 3 threads, each thread would iterate on the following subset of values:



**Exercise**: You are supplied with a starter code for this exercise that you can extend to complete this exercise in the following manner:

1. Download the supplied ex6\_2.cpp starter code and save program to the Linux machine and briefly review the methods in the program.
2. Using the examples discussed in class (slides available of Canvas) complete the createThreads methods to run the primeCheck method (with suitable parameters) from thrCount number of threads.

**Sample Output**:

Note that the results from the program should be exactly the same immaterial of the number of threads used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **$ ./ex6\_2 numbers.txt 1**  5915587277: is prime  1500450271: is prime  3267000013: is prime  5915587279: not prime  1500450275: not prime  3267000011: not prime  5754853343: is prime  4093082899: is prime  9576890767: is prime  3628273133: is prime  9576890765: not prime  3628273131: not prime  2860486313: is prime  5463458053: is prime  3367900313: is prime  5463458051: is prime  3367900319: not prime |  | **$ ./ex6\_2 numbers.txt 2**  5915587277: is prime  1500450271: is prime  3267000013: is prime  5915587279: not prime  1500450275: not prime  3267000011: not prime  5754853343: is prime  4093082899: is prime  9576890767: is prime  3628273133: is prime  9576890765: not prime  3628273131: not prime  2860486313: is prime  5463458053: is prime  3367900313: is prime  5463458051: is prime  3367900319: not prime |  | **$ ./ex6\_2 numbers.txt 3**  5915587277: is prime  1500450271: is prime  3267000013: is prime  5915587279: not prime  1500450275: not prime  3267000011: not prime  5754853343: is prime  4093082899: is prime  9576890767: is prime  3628273133: is prime  9576890765: not prime  3628273131: not prime  2860486313: is prime  5463458053: is prime  3367900313: is prime  5463458051: is prime  3367900319: not prime |

# Part #3: Developing a multi-threaded program to demonstrate race conditions [3 points]

*Estimated time to complete: 15 minutes*

**Exercise**: You are expected to develop a multithreaded C++ program that demonstrates a race condition.

**Sample outputs**:

In the space below illustrate a sample output from your program highlighting a race condition. In addition, indicate what is the correct output expected from the program.

|  |
| --- |
| In the space below illustrate expected output from your program **assuming it did not suffer from a race condition**: |
|  |

|  |
| --- |
| In the space below illustrate actual output from your program. Highlight the output illustrating the race condition: |
|  |

# Part #4: Submit files to Canvas

Once you have successfully completed the lab exercise, upload (One copy per team) the following at the end of the lab exercise:

1. This MS-Word document saved with the convention *MUid*\_Exercise6.docx.
2. Program developed in part #1 of the exercise named with the convention ex6\_1.cpp.
3. Program developed in part #2 of the exercise named with the convention ex6\_2.cpp.
4. Program developed in part #3 of the exercise named with the convention ex6\_3.cpp.

Upload each file individually to Canvas. Do not upload archive files such as zip/7zip/rar/tar/gz etc. Ensure you click the Submit button on Canvas once you have uploaded all the necessary files.