EE 4374 Operating Systems Design
Lab #3: Multi-threaded Prime Number Search
Due Date: Lecture 1 Week 11 (before Midnight) [Apr. 7, 2016]

Objectives:

1) To learn how to create multi-threaded programs using the POSIX pthreads library.
2) To learn how to provide user interactivity in a multi-threaded program.
3) To learn how to balance the load among threads to provide the highest speedup.

Tasks:

1) Unpack the Lab 3 template provided by the instructor into your home directory: 'tar zxvf student_lab3.tgz'. This will create a directory called 'student_lab3', please rename this directory to firstinitiallastname_lab3 using the 'mv' command. For example, the instructor would rename the directory by executing 'mv student_lab3 mmcgarry_lab3'. Next, go into the directory and rename all of the files from 'student_*' to 'firstinitiallastname_*' much like you renamed the directory.

2) Write a test_prime() function that returns a 0 if the integer argument is not a prime number, 1 if it is prime.

```
int test_prime(int n);
```

3) Write a prime_search() function that serves as a start routine for your prime search threads. This function will be given a range of integers and determine which integers in that range are prime. Prime numbers will be printed to a file "primesx", where x is the thread number. Each prime number will be on a separate line in the file.

```
void *prime_search(void *param);
```

4) Write a mini_shell() function that serves as a start routine for the interactivity thread. This function will display a prompt and take commands from the user.

The following single character commands must be supported:

'1' : will return the integer prime search thread 1 is currently checking
'2' : will return the integer prime search thread 2 is currently checking
'a' : will return the integers both prime search thread 1 and 2 are currently checking

```
void *mini_shell(void *param)
```

5) Write a main() program that searches the first **five million** integers for prime numbers by using two threads. A third thread is created to enable the user to

check the search status of the two prime search threads while they are running. When the prime number search is completed the main function will combine the individual files “primes1” and “primes2” into a single output file “primes”. Implement test_prime(), prime_search(), and mini_shell() as library functions in firstinitiallastname_prime.c/h and main() in firstinitiallastname_lab3.c.

6) Balance the search load among the two search threads to decrease the execution time as much as you can.

7) Use a Makefile to build your program.

8) Submit the deliverables, indicated below, as a single tarball file named firstinitiallastname_lab3.tgz to the Lab TA’s email with subject line: “[EE4374] Lab 3 Submission”.

Deliverables:

1) Submit all of the source files in your Lab 3 directory as a single tarball file. You can create this by changing to the directory above your Lab 3 directory and execute ‘tar zcvf firstinitiallastname_lab3.tgz firstinitiallastname_lab3’. As an example, the instructor would execute ‘tar zcvf mmcgarry_lab3.tgz mmcgarry_lab3’. This file will be emailed to the Lab TA. Submissions that do not follow these specifications will be ignored!

Scoring:

Lab grades will be based on four criteria that will determine your overall grade. The first criterion determines if your program compiles and executes correctly. The correct result must adhere to the **Tasks** section. The second criterion determines if your program uses the specified libraries and/or function prototypes as specified in the **Task** section. The last criterion determines if your source code is well-documented and adheres to submission guidelines. Your source code must include (at the top) your name, class section, due date, assigned date, and a small description of your program. Also, every function/method must be commented by specifying the purpose of the input values (if any) and their respective output values (e.g. void foo(int x) /* input: x > 0, output: x = x * 2*/).

Operation/Successful Demonstration	**70%**
Does the program compile? 30%	
Is the program multi-threaded as described? 30%	
Has the load among the threads been optimally balanced? 10%	
Adherence to Interface Specification	**20%**
Does your program use the libraries/APIs specified in the	

lab assignment? 10%	
Does your program use the function prototypes specified in the lab assignment? 10%	
Comments/Adherence to Submission Specification	**10%**
Does your submission adhere to the filename guidelines? 5%	
Is the source code well-documented? 5%	
Lateness	**10%** per day (including weekends and holidays)