2016.1 Multicore Computing, Project #2

(Due : May 20th 11:59pm)

Submission Rule

1. Create a directory {studentID#}_proj2 (example: 20123601_proj2). In the directory, create subdirectories 'prob1' and 'prob2'.

2.a For problem 1, write (i)'C with OpenMP' source code prob1.c, (ii)a document that reports the parallel performance of your code, and (iii) readme.txt into the directory "proj2". The document that reports the parallel performance should contain (a) in what environment (e.g. CPU type, memory size, OS type ...) the experimentation was performed, (b) **tables and graphs** that show the execution time (unit:milisecond) for thread number = {1,2,4,8,16}. (c) The document should also contain **explanation on the results and why such results are obtained**. In **readme.txt** file, you should briefly explain how to compile and execute the source codes you submit. Insert the files (i), (ii), and (iii) into the subdirectory 'prob1'.

2.b For problem 2, write 'C with pthread' source code prob2.c and insert the file 'prob2.c' into the subdirectory 'prob2'.

3. zip the directory {studentID#}_proj2 into and submit the zip file into eClass homework board.

※ The experimentation should be done **in a quad-core CPU computer. Quad-core CPU PCs are** available in our 4th floor laboratory and etc.

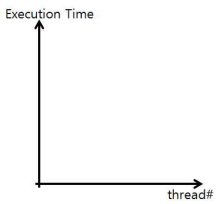
[Problem 1] In our lab. class, we looked at the JAVA program (ex4.java) that computes the number of 'prime numbers' between 1 and 200000. The java code (ex4.java) creates threads for parallel computation using static load balancing approach. However, The parallel implementation of ex4**.java** does not give satisfactory performance because of bad load balancing. The problem is that (i) higher ranges have fewer primes and (ii) larger numbers are harder to test. Therefore thread workloads become uneven and hard to predict. For better performance, we implemented dynamic load balancing approach as project 1 where each thread takes a number one by one and test whether the number is a prime number.

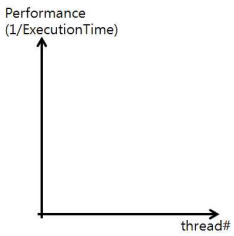
(i) Write 'C with OpenMP' code that computes the number of prime numbers between 1 and 200000. Your program should take two command line arguments: scheduling type number (1=static, 2=dynamic, 3=guided) and number of threads (1, 2, 4, 8, 16) as program input argument. Use **schedule(static) , schedule(dynamic,4) ,** and **schedule(guided,4)**. Your code should print the execution time as well as the number of the prime numbers between 1 and 200000.

command line execution: > **a.out scheduling_type# #_of_thread**

execution example> **a.out 2 8** <---- this means the program use dynamic scheduling using 8 threads.

(ii) Write a document that reports the parallel performance of your code. The graph that shows the execution time when using 1,2,4,8,16 threads. There should be at least three graphs the show the result of static, dynamic, and guided scheduling policy. Your document also should mention which CPU (dualcore? or quadcore?, clock speed) was used for executing your code.





exec time	1	2	4	8	16
static					
dynamic					
guided					

performace (1/exec time)	1	2	4	8	16
static					
dynamic					
guided					

[Problem 2] This is exactly the same as **[Exercise 5: Garage Parking Simulation] in Lab. 2 (May 2nd)**. Write a multi-threaded C code 'prob2.c' using pthread library (Use LINUX or cygwin environment). You should use mutex lock and conditional variables to correctly implement it. (As implemented in sample JAVA code, assume that the number of free parking space in the garage is 10, and there can be 40 cars.)

compilation example> **gcc prob2.c -lpthread**
execution example> **a.out**

```
Car 38: entered
Car 21: entered
Car 12: entered
Car 22: entered
Car 23: left
Car 5: entered
Car 32: entered
Car 28: entered
Car 18: entered
Car 5: left
Car 37: entered
Car 22: left
Car 35: entered
  ...
```