CPSC 410 Java Programming

Prime Threads Program

Name: Fernando Simon  
ID #: 970784  
Instructor: Julius Dichter  
Date: February 6, 2014

**Table of Contents**

Abstract 3

Introduction 3

Screenshot of Output 4

Source Codes 5  
 SwitchTest.java

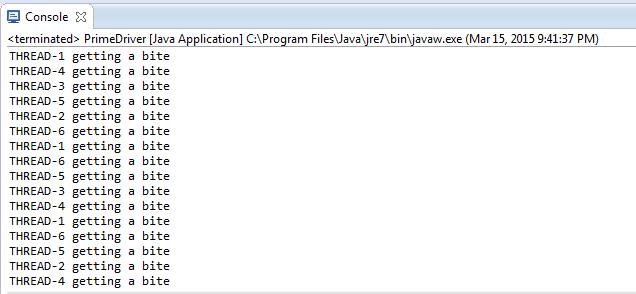
Work Cited 10

**Abstract**

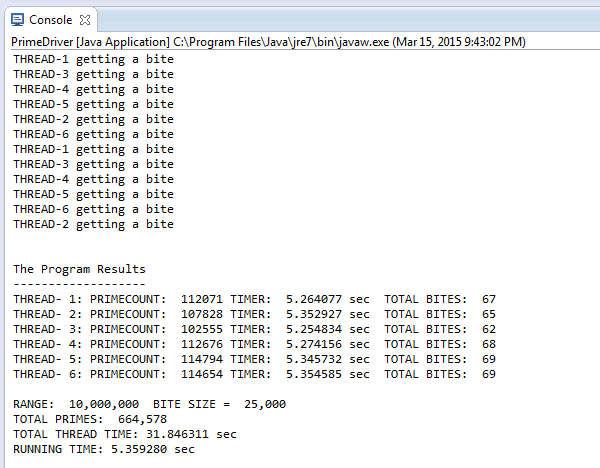
Make a program that calculates primes given a range of numbers. The program split the task in several task and compares with a single thread program running the same amount n of numbers

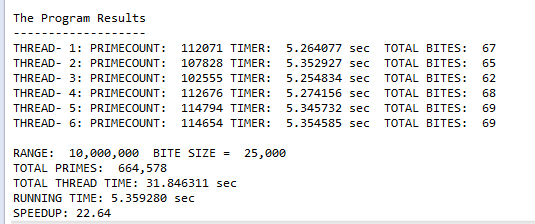
**Introduction**

Once the program start, it will take small bites from a range of n numbers, e.g. if the range is 10,000,000 the bite could be 25,000. Once the program starts it divide the task into threads, e.g. 6 threads. The final output will show the time that each thread take to calculate, the number of bites and number of primes processed by each thread. The program also compares the results with a single thread program in speedup results.

**Screenshot of Output**Normal Output with 6 threads  
  


Calculating speed up



Speed up shown  


**Source Code**PrimeDriver.java

/\*\*

\* **@author** Fernando Simon

\* **@version** 1.0

\*/

**public** **class** PrimeDriver {

**private** **static** **int** *N* = 6; //# of Threads

//-----Vars used to calculate times

**private** **static** **double** *timeAcc*;

**private** **static** **double** *runninTime*;

**private** **static** **double** *speedup*;

**private** **static** **int**[] *numOfPrimes*;

**private** **static** **double**[] *timer*;

**private** **static** **int**[] *biteCount*;

**private** **static** **double** *start*;

**private** **static** **double** *end*;

**private** **static** **double** *elapseTime*;

//----------------------------------------

**private** **static** **int** *range* = 10000000;

**private** **static** **int** *bite* = 25000;

**private** **static** **int** *totalPrimes*;

**public** **static** **void** main(String[] args) **throws** InterruptedException {

PimeExec[] worker = **new** PimeExec[*N*];

PrimeWork pw = **new** PrimeWork(3, *range*, *bite*);

*numOfPrimes* = **new** **int**[*N*];

*timer* = **new** **double**[*N*];

*biteCount* = **new** **int**[*N*];

*start* = System.*nanoTime*();

**for** (**int** i = 0; i < *N*; i++) {

worker[i] = **new** PimeExec(pw, "THREAD-" + (i + 1));

}

**for** (**int** i = 0; i < *N*; i++) {

worker[i].start();

}

**for** (**int** i = 0; i < *N*; i++) {

worker[i].join();

*numOfPrimes*[i] = worker[i].count;

*timer*[i] = worker[i].accumTime;

*biteCount*[i] = worker[i].bite;

*totalPrimes* = *totalPrimes* + *numOfPrimes*[i];

*timeAcc* = *timeAcc* + *timer*[i];

}

*end* = System.*nanoTime*();

*elapseTime* = *end* - *start*;

System.*out*.println();

System.*out*.println("The Program Results");

System.*out*.println("-------------------");

//--------Compare to Single Core and Print Results--------------------------

**for** (**int** i = 0; i < *N*; i++) {

String Results = String

.*format*("THREAD-%2d: PRIMECOUNT: % 7d TIMER: % .6f sec TOTAL BITES: %3d",

(i + 1), *numOfPrimes*[i], *timer*[i], *biteCount*[i]);

System.*out*.println(Results);

}

*printResults*();

}

//---Summary------

**public** **static** **void** printResults() {

*runninTime* = *elapseTime* / (1000000000);

System.*out*.println();

System.*out*.printf("RANGE: %, d BITE SIZE = %, d \n", *range*, *bite*);

System.*out*.printf("TOTAL PRIMES: %, d \n", *totalPrimes*);

System.*out*.printf("TOTAL THREAD TIME: %.6f sec\n", *timeAcc*);

System.*out*.printf("RUNNING TIME: %.6f sec\n", *runninTime*);

/\* Single-Core Prime calculation \*/

PrimeSingle ps = **new** PrimeSingle(3, *range*);

ps.run();

// Calculate speed based on single core

*speedup* = ps.elapsedTime - *runninTime*;

System.*out*.printf("SPEEDUP: %.2f", *speedup*);

}

}

PrimeExec.java

**public** **class** PimeExec **extends** Thread {

**double** start;

**double** stop;

**double** timeElapsed;

**public** **double** accumTime;

**public** **int** bite = 0;

**private** **int** lo;

**private** **int** hi;

**private** **int** localCount;

**private** PrimeWork prime;

**private** String mess;

Range range;

**int** counter = 0;

**public** **int** count;

**public** PimeExec(PrimeWork pw1, String mess) {

prime = pw1;

**this**.mess = mess;

}

**public** **void** run() {

**while** (!prime.complete) {

System.*out*.println(mess + " getting a bite");

range = prime.getWork();

hi = range.hi;

lo = range.lo;

start = System.*nanoTime*();

count = countPrimes(lo, hi); // Counts the primes.

stop = System.*nanoTime*();

timeElapsed = (stop - start) / (1000000000);

accumTime = accumTime + timeElapsed;

bite++;

}

}

**public** **int** countPrimes(**int** lo, **int** hi) {

**for** (**int** i = lo; i <= hi; i++) {

**if** (*isPrime*(i)) {

counter++;

}

}

**return** counter;

}

**public** **static** **boolean** isPrime(**int** number) {

**int** sqrt = (**int**) Math.*sqrt*(number) + 1;

**for** (**int** i = 2; i < sqrt; i++) {

**if** (number % i == 0) {

// number is perfectly divisible - no prime

**return** **false**;

}

}

**return** **true**;

}

}

PrimeWork.java

**public** **class** PrimeWork {

**private** **int** lo;

**private** **int** hi;

**public** **int** bite;

**public** **int** current;

**public** **boolean** complete;

**public** PrimeWork(**int** max, **int** bite) {

lo = 3;

hi = max;

current = lo;

**this**.bite = bite;

}

**public** PrimeWork(**int** min, **int** max, **int** bite) {

lo = min;

hi = max;

current = lo;

**this**.bite = bite;

}

**public** **synchronized** Range getWork() {

Range range = **null**;

**if** (!complete) {

**if** (current + bite > hi) {

range = **new** Range(current, hi);

complete = **true**;

System.*out*.println();

} **else** {

range = **new** Range(current, current + bite - 1);

current = current + bite;

}

}

**return** range;

}

}

PrimeSingle.java

**public** **class** PrimeSingle {

**int** localCount = 0;

**int** start;

**int** end;

**long** time1;

**public** **double** elapsedTime;

**public** PrimeSingle(**int** start, **int** end) {

**this**.start = start;

**this**.end = end;

}

**public** **void** run() {

// # of primes in current thread

time1 = System.*nanoTime*();

**for** (**int** number = start; number <= end; number++) {

// check if numbers are prime

**if** (*isPrime*(number)) {

localCount++; // count the local primes

}

}

elapsedTime = (System.*nanoTime*() - time1) / (1000000000);

}

**public** **static** **boolean** isPrime(**int** number) {

**int** sqrt = (**int**) Math.*sqrt*(number) + 1;

**for** (**int** i = 2; i < sqrt; i++) {

**if** (number % i == 0) {

// number is perfectly divisible - no prime

**return** **false**;

}

}

**return** **true**;

}

}

# Works Cited

Oracle. *Java™ Platform, Standard Edition 7*. n.d. 4 2 2014.