Project 1: Problem 1

소프트웨어학부 20204898 박소은

## **Environment**

* **Processor**: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
* **Number of cores**: 4개
* **RAM**: 16.0GB(15.8GB available)
* **OS**: Windows 11 (64 bit)

## **Tables and graphs**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exec time** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** | **32** |
| static(block) | 7198 | 5791 | 4241 | 2911 | 2788 | 2511 | 2447 | 2352 | 2390 | 2253 |
| static(cyclic) | 7114 | 4404 | 2787 | 2669 | 2304 | 2270 | 2161 | 1885 | 1814 | 1810 |
| dynamic | 7188 | 3867 | 2308 | 1865 | 1813 | 1784 | 1778 | 1774 | 1770 | 1770 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** | **32** |
| static(block) | 0.000122579 | 0.00017 | 0.00024 | 0.00034 | 0.000359 | 0.000398 | 0.000409 | 0.000425 | 0.000418 | 0.000444 |
| static(cyclic) | 0.000133103 | 0.00023 | 0.00036 | 0.00037 | 0.000434 | 0.000441 | 0.000463 | 0.000531 | 0.000551 | 0.000552 |
| dynamic | 0.000139958 | 0.00026 | 0.00043 | 0.00054 | 0.000552 | 0.000561 | 0.000562 | 0.000564 | 0.000565 | 0.000565 |

## **Explanation / Analysis**

## **Java source code**

### **Static (Block)**

public class pc\_static\_block {

private static int NUM\_END = 200000; // default input

private static int NUM\_THREADS = 1; // default number of threads

public static void main(String[] args) {

if (args.length == 1) {

NUM\_THREADS = Integer.parseInt(args[0]);

}

int blockSize = (int) Math.ceil(NUM\_END / NUM\_THREADS);

int totalCounter = 0;

BlockThread[] threads = new BlockThread[NUM\_THREADS];

long startTime = System.currentTimeMillis(); // program execution time starts

// start threads

for(int i=0; i<NUM\_THREADS; i++) {

int end;

if (i == NUM\_THREADS-1) {

end = NUM\_END; // if thread[i] is last thread: end number is NUM\_END

} else {

end = i\*blockSize + blockSize; // if thread[i] is not last thread: calculate by blockSize

}

threads[i] = new BlockThread(i\*blockSize, end);

threads[i].start();

}

// Thread join()

for (int i=0; i<NUM\_THREADS; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {}

}

// Get the total number of prime numbers

for (int i=0; i<NUM\_THREADS; i++) {

totalCounter += threads[i].counter;

}

long endTime = System.currentTimeMillis(); // program execution time ends

long timeDiff = endTime - startTime;

// print the result

System.out.println("\n < RESULT > ");

for (int i=0; i<NUM\_THREADS; i++) {

System.out.println(i +" Thread: " + threads[i].timeDiff + " ms");

}

System.out.println("\nTotal Program Execution Time: " + timeDiff + "ms");

System.out.println("1... " + (NUM\_END - 1) + " prime# counter=" + totalCounter + "\n");

}

}

class BlockThread extends Thread {

int start, end, counter;

long timeDiff;

/\*

BlockThread tests whether the number is a prime number or not

from 'start' number ~ to 'end' number

\*/

BlockThread(int start, int end) {

this.start = start;

this.end = end;

this.counter = 0;

}

@Override

public void run() {

long startTime = System.currentTimeMillis();

for (int num=start; num<end; num++) {

if(isPrime(num)) counter++;

}

long endTime = System.currentTimeMillis();

timeDiff = endTime - startTime;

}

private static boolean isPrime(int x) {

if (x<=1) return false;

for(int i=2; i<x; i++) {

if (x%i == 0) return false;

}

return true;

}

}

### **Static (Cyclic)**

public class pc\_static\_cyclic {

private static int NUM\_END = 200000; // default input

private static int NUM\_THREADS = 1; // default number of threads

private static int TASK\_SIZE = 10; // default task size

public static void main(String[] args) {

if (args.length == 1) {

NUM\_THREADS = Integer.parseInt(args[0]);

}

int totalCounter = 0;

CyclicThread[] threads = new CyclicThread[NUM\_THREADS];

long startTime = System.currentTimeMillis(); // program execution time starts

// Run threads

for(int i=0; i<NUM\_THREADS; i++) {

threads[i] = new CyclicThread(i, NUM\_END, NUM\_THREADS);

threads[i].start();

}

// Thread join()

for (int i=0; i<NUM\_THREADS; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {}

}

// Get the total number of prime numbers

for (int i=0; i<NUM\_THREADS; i++) {

totalCounter += threads[i].counter;

}

long endTime = System.currentTimeMillis(); // program execution time ends

long timeDiff = endTime - startTime;

// print the result

System.out.println("\n < RESULT > ");

for (int i=0; i<NUM\_THREADS; i++) {

System.out.println(i +" Thread: " + threads[i].timeDiff + " ms");

}

System.out.println("\nTotal Program Execution Time: " + timeDiff + "ms");

System.out.println("1... " + (NUM\_END - 1) + " prime# counter=" + totalCounter + "\n");

}

}

class CyclicThread extends Thread {

private static int TASK\_SIZE = 10; // default task size

int threadIndex, NUM\_END, NUM\_THREADS, counter;

long timeDiff;

/\*

A CyclicThread get the number to calculate by its 'threadIndex'

if 'threadIndex' == 0 and NUM\_THREADS == 4:

it starts calculating from 0 to 9,

the next number to calculate is (0 + TASK\_SIZE\*NUM\_THREADS),

therefore calculates 40 ~ 49,

and then 80 ~ 89 ... and so on.

\*/

CyclicThread(int threadIndex, int NUM\_END, int NUM\_THREADS) {

this.threadIndex = threadIndex;

this.NUM\_END = NUM\_END;

this.NUM\_THREADS = NUM\_THREADS;

this.counter = 0;

}

@Override

public void run() {

long startTime = System.currentTimeMillis();

/\*

'num' is a starting number to calculate.

If num is 0, start calculating from 0,

If num is 3, start calculating from 30...

(because the TASK\_SIZE is 10)

\*/

int num = threadIndex \* TASK\_SIZE;

while (num <= NUM\_END) {

for(int i=num; i<num+TASK\_SIZE; i++) {

if(isPrime(i)) counter++;

}

num += NUM\_THREADS \* TASK\_SIZE;

}

long endTime = System.currentTimeMillis();

timeDiff = endTime - startTime;

}

private static boolean isPrime(int x) {

if (x<=1) return false;

for(int i=2; i<x; i++) {

if (x%i == 0) return false;

}

return true;

}

}

### **Dynamic**

public class pc\_dynamic {

private static int NUM\_END = 200000; // default input

private static int NUM\_THREADS = 1; // default number of threads

private static int TASK\_SIZE = 10; // default task size

public static void main(String[] args) {

if (args.length == 1) {

NUM\_THREADS = Integer.parseInt(args[0]);

}

int totalCounter = 0;

DynamicThread[] threads = new DynamicThread[NUM\_THREADS];

IndexGenerator indexGenerator = new IndexGenerator(); // threads share the indexGenerator

long startTime = System.currentTimeMillis(); // program execution time starts

// Start threads

for(int i=0; i<NUM\_THREADS; i++) {

threads[i] = new DynamicThread(indexGenerator, NUM\_END, NUM\_THREADS);

threads[i].start();

}

// Thread join()

for (int i=0; i<NUM\_THREADS; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {}

}

// Get the total number of prime numbers

for (int i=0; i<NUM\_THREADS; i++) {

totalCounter += threads[i].counter;

}

long endTime = System.currentTimeMillis(); // program execution time ends

long timeDiff = endTime - startTime;

// print the result

System.out.println("\n < RESULT > ");

for (int i=0; i<NUM\_THREADS; i++) {

System.out.println(i +" Thread: " + threads[i].timeDiff + " ms");

}

System.out.println("\nTotal Program Execution Time: " + timeDiff + "ms");

System.out.println("1... " + (NUM\_END - 1) + " prime# counter=" + totalCounter + "\n");

}

}

class DynamicThread extends Thread {

private static int TASK\_SIZE = 10; // default task size

int NUM\_END, NUM\_THREADS, counter, num;

long timeDiff;

IndexGenerator indexGenerator;

DynamicThread(IndexGenerator indexGenerator, int NUM\_END, int NUM\_THREADS) {

this.indexGenerator = indexGenerator;

this.NUM\_END = NUM\_END;

this.NUM\_THREADS = NUM\_THREADS;

this.counter = 0;

}

@Override

public void run() {

long startTime = System.currentTimeMillis();

num = indexGenerator.generateIndex(); // IndexGenerator gives the number to calculate

while (num <= NUM\_END) {

for(int i=num; i<num+10; i++) {

if(isPrime(i)) counter++;

}

num = indexGenerator.generateIndex();

}

long endTime = System.currentTimeMillis();

timeDiff = endTime - startTime;

}

private static boolean isPrime(int x) {

if (x<=1) return false;

for(int i=2; i<x; i++) {

if (x%i == 0) return false;

}

return true;

}

}

class IndexGenerator {

public static int index = 0;

/\*

IndexGenerator stores the last number the threads have calculated.

Threads can get the number by 'generateIndex()'

'synchronized' keyword is used to make the function a critical section

because 'index' variable should be protected when one thread is accessing 'index'

\*/

public synchronized int generateIndex() {

index += 10;

return index;

}

}

## **Screen capture image of program execution and output**

* pc\_static\_block

텍스트이(가) 표시된 사진

자동 생성된 설명

* pc\_static\_cyclic

텍스트이(가) 표시된 사진

자동 생성된 설명

* pc\_dynamic

텍스트이(가) 표시된 사진

자동 생성된 설명

## **How to compile and execute the source code**

* **Static (Block)**
  + Compilation: $ javac pc\_static\_block.java
  + Execution

Default: $ java pc\_static\_block

N Threads: $ java pc\_static\_block N

* **Static (Cyclic)**
  + Compilation: $ javac pc\_static\_cyclic.java
  + Execution

Default: $ java pc\_static\_cyclic

N Threads: $ java pc\_static\_cyclic N

* **Dynamic**
  + Compilation: $ javac pc\_dynamic.java
  + Execution

Default: $ java pc\_dynamic

N Threads: $ java pc\_dynamic N