Ahmad Atout 11822841

OS Assignment

Part 1: running threads without Sync

Code:

public class OS {

public static int sharedVariable = 0; // shared variable to be incremented by threads

public static void main(String[] args) {

long start = System.nanoTime(); // start time of executing the code

int N = 841 + 500; // N is the total number of threads to be created (11822841 = 841 + 500)

Thread[] threads = new Thread[N]; // create an array to hold N threads

// create N threads and start them

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

threads[i] = new Thread(new MyRunnable());

threads[i].start();

}

try {

// wait for all threads to finish

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

threads[i].join();

}

} catch (Exception e) {

System.out.println(e.toString());

}

// print the expected and actual values of the shared variable

System.out.println("////////////////////////////////////////////");

System.out.println("Expected value is N^2 = "+ 1341\*1341+" , counter value = "+ sharedVariable);

System.out.println("////////////////////////////////////////////");

long duration = (System.nanoTime() - start)/1000000; // current time - start time to measure performence

System.out.println("Performence time in ms: "+ duration);

}

// this class implements the Runnable interface, so it can be used as a task to be executed by threads

static class MyRunnable implements Runnable {

public void run() {

for(int i= 0; i < 841+500; i++) {

// sleep for TID%10 nano seconds

System.out.println("I'm thread "+Thread.currentThread().getId()+" about to go to sleep for "+Thread.currentThread().getId()%10

+" nanoseconds");

try {

Thread.sleep((Thread.currentThread().getId()%10)/1000000);

} catch (Exception E) {

System.out.println(E.toString());

}

// increment the shared variable

System.out.println("I'm thread "+ Thread.currentThread().getId()+"; about to increment the counter, old value was "

+ sharedVariable );

sharedVariable++;

System.out.println("I'm thread "+ Thread.currentThread().getId()+"; finished incrementing the counter, new value is "

+ sharedVariable );

}

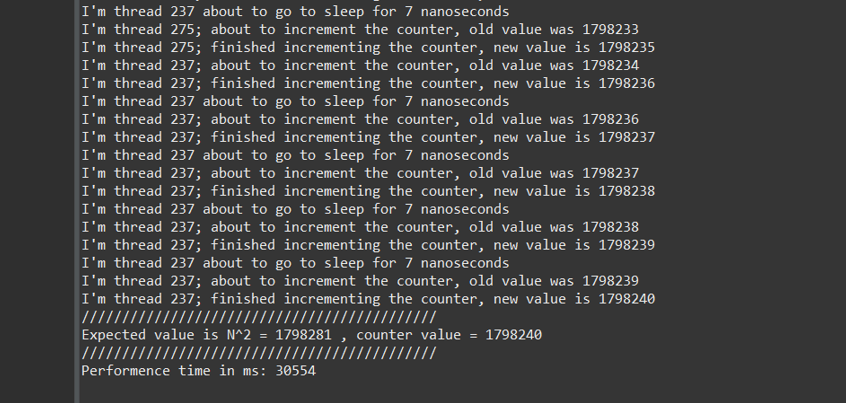
}

}

}

Code is explained through comments.

Output:



Expected value does not equal the value of counter because we don’t have synchronization here so race conditions happen between threads so some threads may read or change the wrong value other than the value they should be manipulating.

Total runtime performance: 30554 ms.

Part 2: using java synchronization ( Synchronization method) 11822841%3 = 0

Code:

public class OS {

public static int sharedVariable = 0;

// Synchronized method to increment the shared variable

public static synchronized void incrementCounter() {

sharedVariable++;

}

public static void main(String[] args) {

long start = System.nanoTime(); // start time of executing the code

int N = 841 + 500; // set the number of threads to create my student ID = 11822841

Thread[] threads = new Thread[N];

// Create and start all the threads

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

threads[i] = new Thread(new MyRunnable());

threads[i].start();

}

try {

// Wait for all the threads to finish

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

threads[i].join();

}

} catch (Exception e) {

System.out.println(e.toString());

}

// Print the final counter value and expected value

System.out.println("////////////////////////////////////////////");

System.out.println("Expected value is N^2 = " + 1341 \* 1341 + " , counter value = " + sharedVariable);

System.out.println("////////////////////////////////////////////");

long duration = (System.nanoTime() - start)/1000000; // current time - start time to measure performence

System.out.println("Performence time in ms: "+ duration);

}

// Inner class implementing the Runnable interface

static class MyRunnable implements Runnable {

public void run() {

// Loop 1341 times and increment the shared variable

for (int i = 0; i < 841 + 500; i++) {

// sleep for TID%10 nanoseconds

System.out.println("I'm thread " + Thread.currentThread().getId() + " about to go to sleep for "

+ Thread.currentThread().getId() % 10 + " nanoseconds");

try {

Thread.sleep((Thread.currentThread().getId() % 10) / 1000000);

} catch (Exception E) {

System.out.println(E.toString());

}

// Use the synchronized method to increment the shared variable

System.out.println("I'm thread " + Thread.currentThread().getId()

+ "; about to increment the counter, old value was " + sharedVariable);

incrementCounter();

System.out.println("I'm thread " + Thread.currentThread().getId()

+ "; finished incrementing the counter, new value is " + sharedVariable);

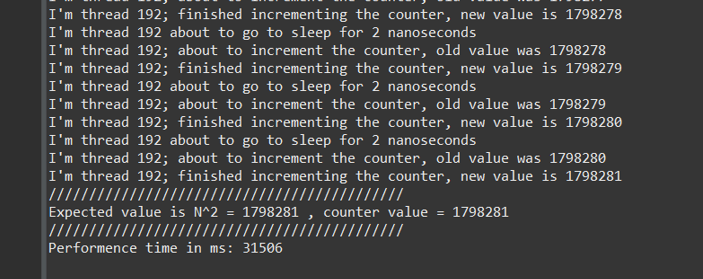
}

}

}

}

Output:



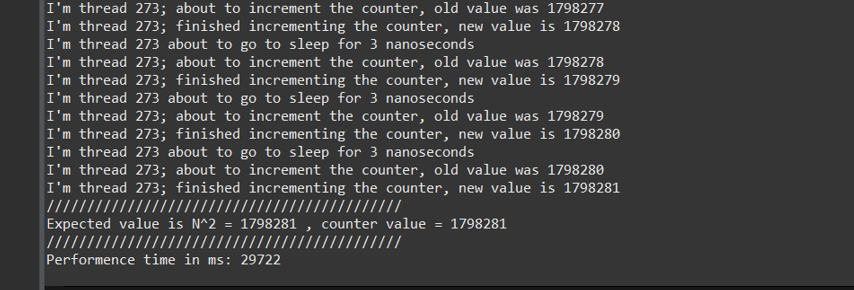
Code is explained Through comments.

Here we used a synchronized method to manipulate the value of the shared variable so only one thread at a time could be executing this critical section only so the expected value is equal to the counter value.

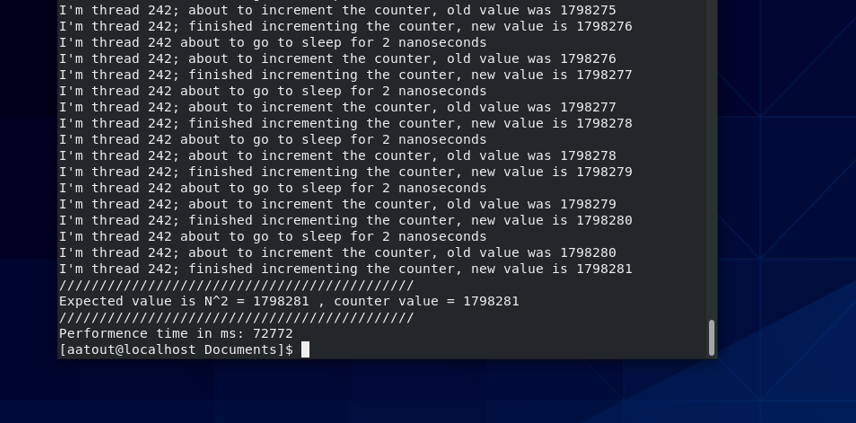
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

After removing the sleep part of the synchronized code and running it on both windows and linux here is the output:

Windows output:



Linux (VM) output:



As we can see the windows did it in less time and this could be because of:

1. Virtualization overhead: Running a virtual machine adds an additional layer of overhead, which can slow down performance compared to running directly on the host operating system.
2. OS differences: The two operating systems may handle certain tasks differently, which can affect performance. For example, the way the scheduler handles thread scheduling could impact how quickly the code runs.