Programming Systems and Environments – Lab 5

Jakub Grzana, 241530

Task 1 – Simple multithreading application

I've did what the task asks for. Code written by me allows naming threads with strings, but also with number (ID) Additionally, I've made only 5 threads, but you can change it by changing one number.

Multithreading and subthread classes are realized in Java with Thread class and Runnable class template.

Task 2 – NumericDataGenerator

Prime check uses slightly more efficient algorithm than typical check of all divisors until sqrt(n), but it isn't probability based check.

I've implemented two classes: NumericDataGenerator, which is main class of this module, and NumericDataParser which is called in misleading manner, cause it is just subthread of NumericDataGenerator. I should've named it in another way but too late now.

NumericDataGenerator is responsible for user interface and management of subthreads: it takes in filename to which data is supposed to be saved, minimal and maximal value of numbers, number of threads. Numbers to be checked are generated from range specified in constructor, it takes all numbers from min to max, then splits them between all threads. Additionally, NumericDataGenerator is responsible for opening output file in appending mode – each subthread receives reference to this file.

NumericDataParse checks assigned numbers one by one, then adds to list of prime numbers if it was indeed prime. In the end, subthread locks access to output file and prints all it's output in one go.

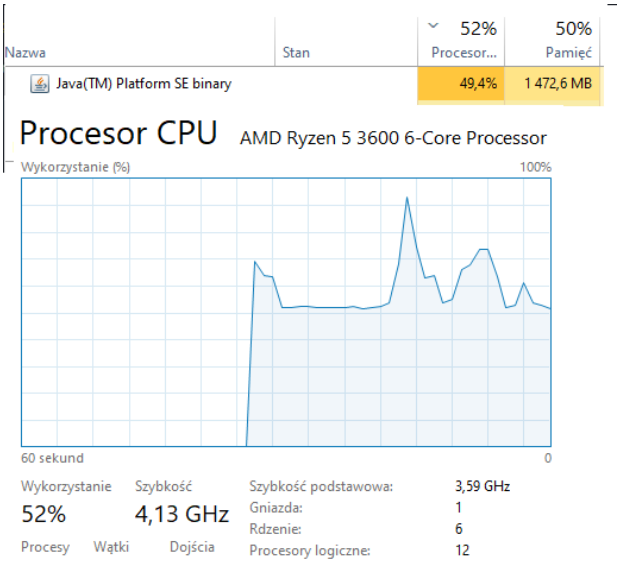
Later I've extended this application to support CSV file with one long number per row as input (instead of range).

Conclusions

Multithreading in Java is much more convenient than C, C++ or Python counterparts. Synchronized keyword is incredibly powerful tool.

Note that in Task 2 I'm actually storing output in CSV file in another way than described in task – I'm saving to file only prime numbers, without storing the result. It is because I misread the task. I could change the code, but it is very small change and I doubt it was purpose of the task anyway.

On following picture you can clearly see usage of processor with 5 threads computing primality test.



So it does work. Quick look into output file shows that synchronize keyword also did it's job. Segments of numbers that were computed by different threads are saved one after the other. This means that only one thread had access to reference to output file, meaning race isn't possible.

Bibliography:

Prime check [access 12.04.2022]:
https://www.geeksforgeeks.org/java-program-to-check-if-a-number-is-prime-or-not/

Reading entire file to single string [access 12.04.2022]:
https://howtodoinjava.com/java/io/java-read-file-to-string-examples/

Multithreading [access 08.05.2022]
https://www.tutorialspoint.com/java/java_multithreading.htm

Generating Random numbers from range [access 08.05.2022]
https://www.baeldung.com/java-generating-random-numbers-in-range#intstreamiterate-1

CODE: Multithreading.java

```
public class Multithreading {
   public static void main(String[] args) throws Exception
   {
      // Task 1
      ArrayList<MyThread> threads = new ArrayList< >();
      for(int i = 0; i < 5; ++i)
      {
         MyThread e = new MyThread(i,"Thread"+i);
         threads.add(e);
         e.start();
      }
      for(MyThread t : threads)
      {
         t.join();
      }

      // Task 2
      NumericDataGenerator generator = new
NumericDataGenerator("file.csv",5,0,100000);
      generator.start();
      generator = new NumericDataGenerator("file.csv", "output.csv", 5);
      generator.start();
   }
}
```

CODE: MyThread.java

```
class MyThread implements Runnable {
   private Thread t;
   final private int threadnum;
   final private String threadname;

   public int RandInt(int min, int max) {
    Random random = new Random();
    return random.nextInt(max - min) + min;
   }

   MyThread( int id, String threadname) {
      this.threadnum = id;
      this.threadname = threadname;
      System.out.println("Creating " + this.threadnum );
   }

    @Override public void run() {
      System.out.println("Running " +  this.threadnum );
      try {
         for(int i = 0; i < 5; ++i) {
            int val = this.threadnum + i;
            System.out.println("Thread: " + this.threadname + ": " + val);
            int time = RandInt(500, 3000);
            Thread.sleep(time);
         }
      } catch (InterruptedException e) {
         System.out.println("Thread " +  this.threadname + " interrupted.");
      }
      System.out.println("Thread " +  this.threadname + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  this.threadnum );
      if (t == null) {
         t = new Thread (this, this.threadname);
         t.start ();
      }
   }

   public void join() throws Exception
    {
        t.join();
    }
}
```

CODE: NumericDataGenerator.java

```
class NumericDataParser implements Runnable
{
   private Thread t;
   final private FileOutputStream file;
   final private ArrayList<Long> numbers;

   private static boolean isPrime(long n)
   {
      // Corner cases
      if (n <= 1)
         return false;
      if (n <= 3)
         return true;

      // This is checked so that we can skip
      // middle five numbers in below loop
      if (n % 2 == 0 || n % 3 == 0)
         return false;

      for (long i = 5; i * i <= n; i = i + 6)
         if (n % i == 0 || n % (i + 2) == 0)
            return false;

      return true;
   }
```

```
@Override public void run()
{
    ArrayList<Long> primes = new ArrayList<>();
    String data_to_save = new String();
    for(Long i : this.numbers)
    {
        boolean feedback = isPrime(i);
        if(feedback)
        {
            primes.add(i);
            data_to_save = data_to_save + String.format("%d\n", i);
        }
    }
    byte[] bytes = data_to_save.getBytes();
    synchronized(this.file)
    {
        try
        {
            System.out.println("Saving...");
            this.file.write(bytes);
        }
        catch (Exception e)
        {
            // nothing
        }
    }
}

NumericDataParser(FileOutputStream fileref, ArrayList<Long> numbers)
{
    this.file = fileref;
    this.numbers = numbers;
}

public void start () {
  if (t == null) {
     t = new Thread (this);
     t.start ();
  }
}
```

```
    public void join() throws Exception
    {
        t.join();
    }
}

public class NumericDataGenerator {
    final private String path;
    final private FileOutputStream file;
    private ArrayList<NumericDataParser> threads;

    private ArrayList<Long> GenerateRange(long minval, long maxval)
    {
        ArrayList<Long> output = new ArrayList<>();
        for(long i = minval; i < maxval; ++i)
        {
            output.add(i);
        }
        return output;
    }

    private ArrayList<Long> LoadData(String path)
    {
        String content = readAllBytesJava7(path);
        String literals[]  = content.split("\n");
        ArrayList<Long> output = new ArrayList<>();
        for(String s : literals)
        {
            long r = Long.parseLong(s);
            output.add(r);
        }
        return output;
    }
```

```
   private static String readAllBytesJava7(String filePath)
   {
      String content = "";

      try
      {
         content = new String ( Files.readAllBytes( Paths.get(filePath) ) );
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
      return content;
   }

   NumericDataGenerator(String path, int threadnum, long minval, long maxval) throws
Exception
   {
      this.path = path;
      this.file = new FileOutputStream(path, true);
      this.threads = new ArrayList<>();
      long r = (maxval - minval) / threadnum;
      for(int i = 0; i < threadnum; ++i)
      {
         // min: r*i;
         // max: r*(i+1)
         ArrayList<Long> numbers = this.GenerateRange(r*i, r*(i+1));
         this.threads.add(new NumericDataParser(this.file, numbers));
      }
   }
   NumericDataGenerator(String input, String path, int threadnum) throws Exception
   {
      this.path = path;
      this.file = new FileOutputStream(path, true);
      this.threads = new ArrayList<>();
      ArrayList<Long> data = this.LoadData(input);
      int size_per_thread = data.size()/threadnum;

      for(int i = 0; i < threadnum; ++i)
      {
         ArrayList<Long> numbers = new ArrayList<>(data.subList(size_per_thread*i,
size_per_thread*(i+1)));
         this.threads.add(new NumericDataParser(this.file, numbers));
      }
   }
```

```
    public void start() throws Exception
    {
        for(NumericDataParser thread : this.threads)
        {
            thread.start();
        }
        for(NumericDataParser thread : this.threads)
        {
            thread.join();
        }
    }
}
```