Technical University of Cluj-Napoca
Faculty of Automation and Computer Science

PC performance test

Indre Bogdan
Group: 30432

Table of Contents

1. Task Description

Design a PC performance testing software. The system tests the processor type, its frequency, the amount of memory, transfer speed of a block of data and the execution speed of some arithmetic and logic operations.

2. Project Design

To design such a system, I decided to use Java as the main programming language with some Assembly on top.

The first task was to find information about the processor and the memory. In order to do that, OSHI (Operating System & Hardware Information) was used. OSHI is a free JNA-based library for Java that allows developers find more about the PC their programs run on.

Next step was the transfer of a data block. To do this, I transferred the data from a newly created RandomAccessFile of 1 MB to another file and measured the time of the transfer. This test was done five times and an average speed was taken.

The final step was the speed of several operations. Here JVM creates a small problem as it is hard to get around it. To obtain a good result, my system uses JNI (Java Native Interface) to pass an array of integers to several assembly snippets of code where the operations are done, and the result is returned to the main program and the speed is computed.

Lastly, a minimalistic GUI was added.

3. Implementation

The system consists of the classes: PerformanceTest and View. The main class where the requirements are done is PerformanceTest.

3.1 Hardware Information

```
private static void findHardware(PrintWriter printWriter, View view) {
    SystemInfo si = new SystemInfo();
    HardwareAbstractionLayer hal = si.getHardware();
    printWriter.write("Processor: " + hal.getProcessor().getName() + "\n");
    printWriter.write("Frequency: " + hal.getProcessor().getVendorFreq() + "\n");
    printWriter.write("Memory: " + FormatUtil.formatBytes(hal.getMemory().getAvailable()) + "/"
            + FormatUtil.formatBytes(hal.getMemory().getTotal()) + "\n");
    view.write("Processor: " + hal.getProcessor().getName() + "\n");
    view.write("Frequency: " + hal.getProcessor().getVendorFreq() + "\n");
    view.write("Memory: " + FormatUtil.formatBytes(hal.getMemory().getAvailable()) + "/"
            + FormatUtil.formatBytes(hal.getMemory().getTotal()) + "\n");
}
```

Using the OSHI library, it is very easy to find this information.

3.2 Transfer test

```
private static long runTransferTest() {

    long startTime = 0, endTime = 0;
    File f1 = new File("Transfer.dat");
    File f2 = new File("Transfered.dat");
    try {

        RandomAccessFile transferFile = new RandomAccessFile(f1, "rw");
        transferFile.setLength(1024 * 1024 * 1);
        RandomAccessFile transferedFile = new RandomAccessFile(f2, "rw");
        byte[] data = new byte[1024 * 1024 * 1];
        startTime = System.nanoTime();

        transferFile.read(data);
        transferedFile.write(data);

        endTime = System.nanoTime();

        transferFile.close();
        transferedFile.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    f1.deleteOnExit();
    f2.deleteOnExit();
    return endTime - startTime;
}
```

The transfer test exchanges data between the files and we compute the time passed with System.nanoTime().

3.3 Operation test

The operation tests are a little bit more complicated as we must build and run assembly code during java run-time. To do this, we have build.bat. This will create a shared library “libPerformanceTest” from the assembly code that we can use with JNI to make our computations.

```
private static void runAssemblyOperations(PrintWriter printWriter, View view) {
    try {
        final Process process = Runtime.getRuntime().exec("cmd /c build.bat");
        process.waitFor();
    } catch (IOException e) {
        printWriter.write("Build not run\n");
        e.printStackTrace();
    } catch (InterruptedException e) {
        printWriter.write("Build interrupted");
        e.printStackTrace();
    }

    long startTime, endTime, time;
    double timeDouble;
    File file = new File("libPerformanceTest.so");
    System.load(file.getAbsolutePath());
```

We will use a simple ARRAY with integers from 1 to 10 for each operation. For the system to work, we must declare the methods native,

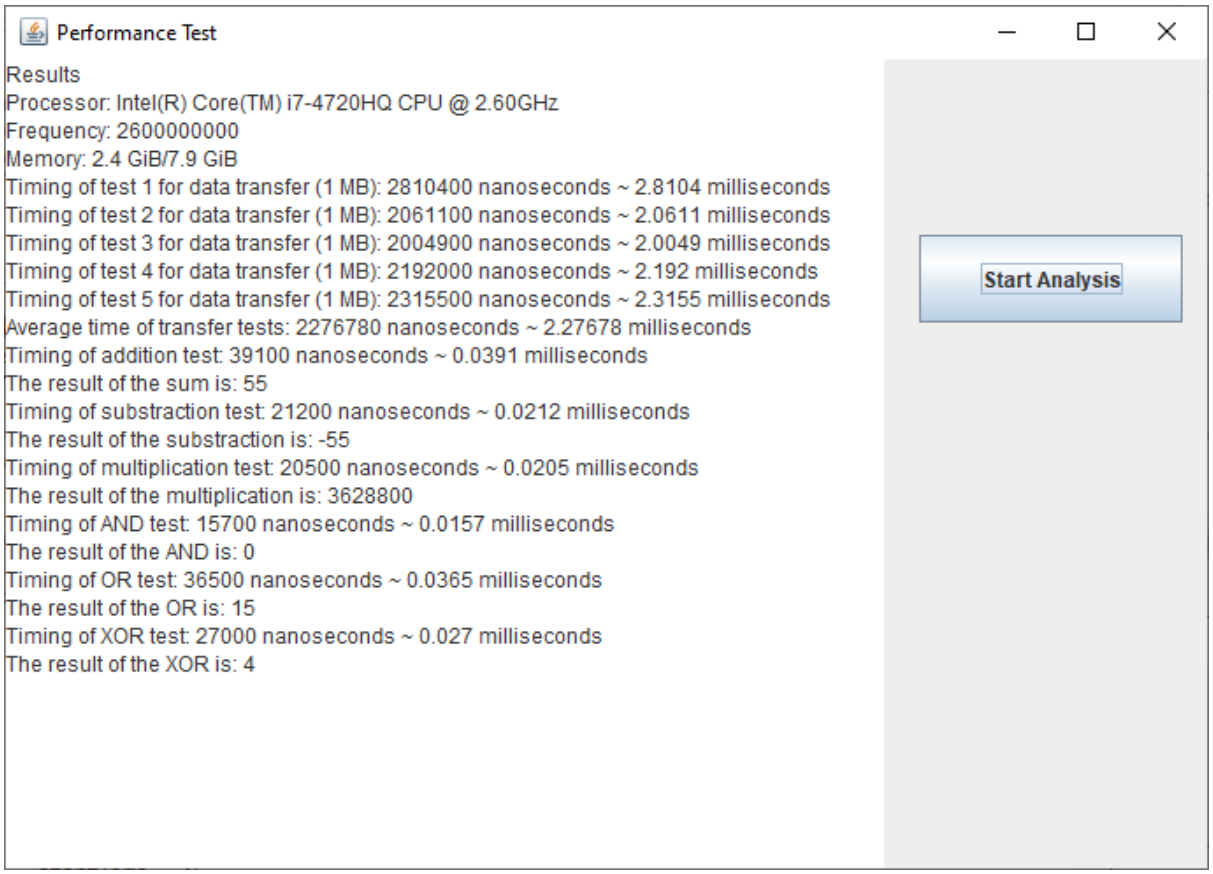
```
public static native long computeArraySub(int[] array, int arrayLength);
```

and call the operation from runAssemblyOperations.

```
long sub = computeArraySub(ARRAY, ARRAY.length);
```

4. Results

The results are displayed on the GUI, but they are also saved in a file. On a PC with Intel Core i7-4720HQ and 8 GB memory we get this.



5. Bibliography

- https://github.com/oshi/oshi
- https://dzone.com/articles/pushing-the-jni-boundaries-java-meets-assembly