Lab 2 : Source Code Energy Measurements and Eco-Design

1 Comparing Internet Browsers

In this exercise, we will compare the CPU energy consumption of internet browsers using PowerJoular.

Software Requirements

We will be using a powermeter and compare three different internet browsers : Mozilla Firefox, Google Chrome and Gnome Web. You can also use other internet browsers in addition to these three.

1.1 System-wide monitoring with PowerJoular

Measuring the Idle Power Consumption

We will first measure the idle power consumption of the computer in order to have a measurement baseline. Restart your computer, then close all running user programs if any are open (email client, browser, etc.).

This can be achieved with the following command (which also saves data to a csv file) :

```
powerjoular -f idle.csv
```

Stop the logging (by holding `CTRL+c` in the terminal), and calculate the average power consumption from the generated CSV log file, and its standard deviation. We will consider this value as the idle power consumption.

Monitoring Power Consumption of Internet Browsers

Open one browser, and open the following video : `https://cdn.kde.org/promo/Announcements/Plasma/5.20/Video.webm`. Don't start the video (rewind to the beginning if it autostarted). Start logging power using the powermeter and name the csv file with the browser's name (such as *firefox-1.csv*), then run the video in the browser. When the video ends, stop the logging (`CTRL+c` in the terminal).

Close the browser, and repeat the same experiment at least 2 more times (for a minimum total of three separate files the browser).

Repeat the same experiment for each of the remaining browsers.

Analyzing Power Data

Collect the different CSV log files, and extract the power consumption evolution for each browser. For each line of the collected data, subtract from it the idle power consumption. Finally, compare the power consumption and total energy for all browsers (produce charts of power evolution, energy comparison, and summary analysis).

Compare all three browsers. Which one is the most energy-efficient in this experiment ?

1.2 Process Monitoring

In this part of the exercise, we will repeat the same steps as above, but we will use PowerJoular's capabilities in monitoring a particular process directly. This can be achieved with the following command (which also saves data to a csv file) :

```
powerjoular -p pid -f file.csv
```

Where `pid` is the process ID of the process we want to monitor. PowerJoular will automatically calculate the CPU power and energy consumption of the process and only this process.

Run the experiment in the previous questions but use the `-p pid` option of PowerJoular to monitor the process of each browser. To get the PID of a browser, you can check the task manager, or use one of the following commands (use the main PID only for each browser in case of multiple ones) :

```
ps aux | grep firefox (or the name of the brower)
pidof firefox
```

1.3 Application monitoring

PowerJoular can also monitor an application by its name instead of its PID. This is useful in case of multiple PIDs created by one application, or if an application is creating or destroying processes throughout its lifecycle. PowerJoular can follow the PIDs of an application through its lifecycle. Monitoring an application by its name can be done with the following command(which also saves data to a csv file) :

```
powerjoular -a appName -f file.csv
powerjoular -a firefox -f firefox-1.csv
```

Run the experiment in the previous questions but use the `-a appName` option of PowerJoular to monitor all processes of each browser.

1.4 Comparison

Compare the power and energy consumption of all three browsers and for all three monitoring methods. What can you conclude ?

2 Comparing Programming Languages

In this exercise, we will compare the CPU energy costs of a program written in different programming languages, using JoularJX.

Software Requirements

We will be using JoularJX and PowerJoular, and the Ray Casting algorithm from the Rosetta Code archive for Java, Python and C : rosettacode.org/wiki/Ray-casting_algorithm.

Make sure all programs compile and run correctly. Use javac (Java 11+), gcc and Python 3.

As you have noticed, the Java version does not print expressive text when running (only true/false strings). Create a second version of the Java program (make sure to rename the file and Java class), and modify the output to be similar to the C version.

Also you noticed each program have different input values. Modify them all to match the Java version input values.

Benchmark

Implement a benchmark batch script to run each program in a loop to have sufficient time to monitor power consumption :

— Write a loop to run each program multiple time, for at least 30 seconds.
— Automate the executing of all 5 programs, and keeping timestamps for start time and end time of each program.
— Add instruction for wait/sleep for 10 seconds between each program. This will help isolate the power consumption of each program later on.

Then start PowerJoular to start monitoring the power consumption and save it to a file, then run the batch script (which will run all programs).

Analyzing Power Data

Collect the different CSV log files, and extract the power consumption evolution for each program. Then, compare the power consumption and total energy for all programs (produce charts of power evolution, energy comparison, and summary analysis).

Which programming language is the most energy-efficient on your machine for the Ray Casting algorithm ?

Source Code Power Consumption

Follow the instruction to download and run JoularJX from its GitLab readme file.

Modify the source code of both Java versions of RayCasting (original and the modified one) so it will run in an infinite loop (`while (true)`), and the class to be part of a new package called RayCasting.

Modify the `config.properties` file to add RayCasting as filtered method :
`filter-method-names=RayCasting`.

Then, run the Java version with JoularJX Java agent :

```
java -javaagent:joularjx-$version.jar yourProgram
```

Repeat the experiment with the modified Java version as above.

As the program will run in an endless loop, you need to stop it manually in order for the program to quit and for JoularJX to generate the final output files. To stop it, tap `CTRL+c` in the terminal.

Open the generated energy files (app energy files), and compare the energy consumption of all methods both Java version in order to detect the energy hotspots.

3 Comparing Collections

In this exercise, we will compare the energy consumption of Java collections (lists and maps).

Compile and run the different implementations of Java collections provided, and monitor their energy consumption with JoularJX (app), and also check the energy impact of the JDK's methods (all).

Now, create a Java program which executes three tasks, individually in order :

— Inserting elements to a collection
— Accessing all elements
— Deleting all elements

Do this for all collections studied in this exercise (lists and maps).

From the results, calculate the average energy consumed for inserting one element, accessing one element, and removing one element, for each of the collections.

Comapre the results. Which collection is the most energy-efficient for each task ?