Mystery Program Analysis

Mini-Lab Computer Architecture

Contents

1 Goals

The goal of this mini-lab is to analyze unknown programs with debug tools such as *Activity Monitor* (MacOs), *Task Manager* (Windows) or *btop* (Linux, MacOS, Windows) as well as performance analysis tools such as **`hyperfine`** (Linux, MacOS, Windows) or **`time`** (Linux, MacOS).

2 Installation

First, you need to install the various tools that we will use for performance testing.

To simplify the installation, the tools are installed via the following package managers:

MacOS **"brew"**, bash:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
HEAD/install.sh)"
```

Windows **"scoop"**, powershell:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

2.1 hyperfine and time

hyperfine is a command line benchmarking application.

hyperfine is available at https://github.com/sharkdp/hyperfine?tab=readme-ov-file#installation.

MacOS	Windows
`brew install hyerfine`	`scoop install hyperfine`

The program **time** is already installed on MacOS and Linux, it is not available on Windows.

2.2 btop

btop is a terminal-based system monitoring tool similar to built-in tools such as *Task Manager* (Windows) or *Activity Monitor* (MacOS).

btop is available at https://github.com/aristocratos/btop?tab=readme-ov-file#installation for Linux and Mac. For Windows, use the fork **btop4win** https://github.com/aristocratos/btop4win?tab=readme-ov-file#installation.

MacOS	Windows
`brew install btop`	`scoop install btop-lhm`

2.3 flamegraph & framelens (optional)

flamegraph is a visualization tool for profiling data. It helps to identify performance bottlenecks in applications by generating flame graphs from stack traces.

flamelens is available at https://github.com/flamegraph-rs/flamegraph?tab=readme-ov-file#installation.

`framelens` is available at https://github.com/YS-L/flamelens

MacOS	Windows

`cargo install flamelens --locked --all-features`	`scoop bucket add extras`
	`scoop install extras/flamelens`

2.4 Verify the installation

To verify that the installation was successful, run the following commands in a terminal:

```
hyperfine --version
btop      --version
time time           # Linux MacOS only
```

3 Mystery Program Analysis

The binary programs in the folder **`car_labs/dbg/release/`** can be executed with different parameters.

```
Usage: rust_mystery_v1_0_0_Mac_AARCH64 [OPTIONS]

Options:
  -m, --mystery <MYSTERY>
  -h, --help                Print help
  -V, --version             Print version
```

> ⚠ Depending on the operating system, a different binary file must be executed.
>
> - **`rust_mystery_v1_0_0_Mac_AARCH64`** for MacOS
> - **`rust_mystery_v1_0_0_Linux_x64`** for Linux
> - **`rust_mystery_v1_0_0_Windows_x64.exe`** for Windows
>
> **Adapt the commands below accordingly.**

The option **`-m`** or **`--mystery`** expects a value from **`1`** to **`5`**. Each value leads to a different behavior of the program.

The program can be executed directly with the option **`-m`**.

```
./rust_mystery_v1_0_0_Mac_AARCH64 -m 1
```

The programs only last a short time. To measure the execution speed, we use the tool **`hyperfine`** as well as **`btop`**.

Start by running **`btop`** in a separate terminal to monitor system usage. Then you can run the respective program with **`hyperfine`**. For example, Mystery 4:

```
hyperfine --warmup 3 --export-markdown mystery-4.md --show-output --min-runs 10
"release/rust_mystery_v1_0_0_Mac_AARCH64 -m 4"
```

> Run all variants **`-m 1`** to **`-m 5`** of the program and analyze the output of **`hyperfine`** and **`btop`**.

3.1 Analysis with **`hyperfine`**

At the end of a **`hyperfine`** benchmark, a summary of the execution speed is displayed.

```
Time (mean ± σ):      64.9 ms ±  13.3 ms    [User: 14.8 ms, System: 12.5 ms]
Range (min … max):    58.3 ms … 147.6 ms    43 runs
```

> These informations are also available in the markdown files exported by **`hyperfine`**, here **`mystery-4.md`**.

In this case, the program was executed 43 times. On average, an execution took **`64.9ms`** with a variation of **`13.3ms`**. The minimum execution time was **`58.3ms`** and the maximum **`147.6ms`**.

The values **`User`** and **`System`** are also important. These values indicate how long the program spent in the user and system space. In this case, **`14.8ms`** and **`12.5ms`**.

> What do the values **`User`** and **`System`** mean concretely? How do they differ from each other?

> Why is the sum of **`User`** and **`System`** not equal to the total execution time?

Look at the execution time, user and system values for all programs. In parallel, observe the CPU, GPU, memory … with **`btop`**.

Try to understand why the programs take different lengths of time.

> Give a hypothesis on the operations that each program may be performing.
>
> Argue based on the measured values / consumptions.