Advanced Operating Systems:
Lab 1 – Getting Started with Kernel Tracing
Part II and L41 Assignment

Prof. Robert N. M. Watson

2022-2023

This first lab exercise introduces you to DTrace, asking you to take a series of basic kernel measurements and present the results in a JupyterLab Notebook. Unlike Lab 2 and Lab 3 assignments, this exercise is a tutorial shared by Part II and L41 courses. You will:

- Step through a set of example DTrace and Python scripts collecting and illustrating results from a simple `dd(1)` workload.
- Complete a set of short exercises set out in a Jupyter Lab Notebook, building on those examples.
- Capture data using several different DTrace *providers* including *systrace* (system-call tracing), *fbt* (function boundary tracing), and *profile* (timer-based profiling).
- Present this data using standard Python plotting mechanisms as well as Flame Graphs.
- Submit a generated PDF of these results for assessment and comments.

These skills will support your future labwork.

1 Assignment documents

All students should read *Advanced Operating Systems: Lab Setup*, which provides information on the lab platform and how to get started.

2 Background: DTrace

DTrace, or Dynamic Tracing, is an OS feature to dynamically instrument software behaviour on supporting operating systems. As explored in Lecture 2, DTrace is highly programmable, with its D programming language used to specify instrumentation points, conditions for data capture, and specific actions to take when those conditions hold. You can refer to *Advanced Operating Systems: DTrace Quick Start* for a quick tutorial to get you started, lecture slides and recordings, and the original research paper by Cantrill et al.

3 Getting started

Before you start, please review the *Advanced Operating Systems: Lab Setup* document, including how to log into your RPi4, how to extract the first lab bundle, and how to run JupyterLab.

It is possible to run the following commands from both the UNIX shell prompt, and also from within JupyterLab. For your labs, we generally recommend the latter. Either way, all commands will be run as the root user. Example command lines are prefixed with the `#` symbol signifying the shell prompt.

4 The JupyterLab notebook template

We have populated the directory `/advopsys-packages/labs` with the Lab 1 JupyterLab tarball, which you will unpack into the `/data` directory. You will also use `/data` as the working directory for JupyterLab.

```
# cd /data
# tar -xzf /advopsys-packages/labs/2022-2023-advopsys-lab1.tbz
```

This will extract the file `2022-2023-advopsys-lab1.ipynb`, which you will use to explore example tracing and plotting scripts that we have prepared for you.

5 The assignment

In this exercise, we use the command-line tool `dd(1)`, as also shown in Lecture 2, to explore some of the tracing techniques available via DTrace. We will then take those DTrace results and analyse them in a JupyterLab Notebook to build skills with that framework and its integration with DTrace, as well as Python plotting tools.

In later labs, we will use bespoke microbenchmarks as the targets of analysis; these also contain internal instrumentation to report performance data as well as collect statistics on their operation. This will provide more precise bracketing of the benchmark behaviour, as well as easy access to I/O or IPC statistics already captured by the kernel, and with lower probe effect than with DTrace.

5.1 Example executions, tracing, and plotting

First, step through the examples in the Lab 1 JupyterLab notebook that we provide. These explore how to execute the benchmark from within JupyterLab, using DTrace from both the command line and within Python, how to plot with Matplotlib and our own Flamegraph module, and several potential uses of DTrace to collect data about a running program including system-call tracing and timer-based profiling.

5.2 Exercises

Start by creating a new JupyterLab notebook clearly labeled with the current exercise number, name, your own name, and the date. Now complete the following exercises, including your D script with each piece of generated output, allowing us to see how you collected the data that you are presenting. Alongside your code blocks tracing the workload and presenting your results, use markdown blocks to both label which exercise is being addressed, and also to describe your work and conclusions. Ensure that plots are suitably labeled with units / scale, data-set labels, and indications of key thresholds or inflection points. Marks are directly awarded for clear, succinct presentation of your work.

1. Plot two histograms using matplotlib, one each showing the distribution of system-call wall-clock execution times for `read(2)` and `write(2)` system calls for our `dd(1)` workload. To do this, instrument the system-call entry and return probes for each of the two system calls. What do the distributions of these two data sets look like?

2. Plot a flamegraph illustrating stack traces to kernel `uma_zalloc_arg(9)` calls made throughout the kernel as `dd(1)` runs, instrumented using the DTrace `fbt` provider's function entry probes. What portion of execution drives the most slab memory allocations?

3. Plot two flamegraphs profiling stack traces using the `profile-4997` probe, once for our `dd(1)` workload with the default count of 10000 blocks, and a second time with a modified count of 1000 blocks. How does proportional execution time of various kernel functions differ between these two similar workloads?

4. Measure the probe effect of using time profiling on our workload; what is the percentage overhead that results from using DTrace in this way? As illustrated in Lecture 2, collect multiple samples, and comment on the statistical strength of your measured difference. Use the Print menu option on JupyterLab's File menu to generate a PDF with your report. Ensure that your output is not truncated by a page edge in the resulting PDF document.