Advanced Operating Systems: Lab Setup

Prof. Robert N. M. Watson

2022-2023

Advanced Operating Systems is taught through a blend of lectures and laboratory experiments. The purpose of the labs is threefold: to teach you about real-world operating systems, to teach you experimental methodology and practical skills, and to provide fodder for assessment. You will use tools such as DTrace to explore the behaviour of the system through 'potted' example programs that will trigger OS behaviours for you to investigate. Each lab is structured as a set of mandatory experimental questions; take care to ensure that your lab report or lab assignment submission addresses all of the assigned experimental questions.

Experimental platform

Our experimental platform is the open-source FreeBSD operating system running on the Raspberry Pi 4 Model B board, described in the remainder of this handout.

The operating system: FreeBSD

We will be using the open-source FreeBSD operating system's ARMv8-A port on the Rasberry Pi 4. FreeBSD is of particular interest due to its tight integration of a number of tracing and measurement tools (e.g., DTrace) and that it is built by default with the Clang/LLVM compiler suite, which make it easier to insert additional instrumentation. You can learn more about FreeBSD by visiting the FreeBSD Project's website:

`https://www.FreeBSD.org`

The course text, *The Design and Implementation of the FreeBSD Operating System, Second Edition* will be a useful reference, covering concepts such as the process model, inter-process communications, and filesystems. There is also a section on the implementation of DTrace, which may be useful background material for the labs. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD* is a detailed, user-facing guide on how to use DTrace, and will also be a useful reference for the labs. In addition to these books, FreeBSD has a rich set of *manual pages* for its commands and APIs, which can be read using the `man` command.

The board: Raspberry Pi 4 (Model B)

The Raspberry Pi 4 (Model B) is a single-board computer based on the Broadcom BCM2711 System-on-Chip (SoC); we have selected the 8GB version of the board for this course. The BCM2711 includes a quad-core superscalar Cortex-A72 64-bit ARMv8-A superscalar (out-of-order) processor clocked at up to 1.5Ghz. For this course, we underclock the Arm cores at 600MHz for reasons of performance determinism, so that as the boards heat up, and reduce their clock speed as part of power management, you don't see performance change. Each of the cores has a 48KB I-cache and 32KB D-cache; they share a common 2MB L2 cache. A key feature of this processor line is the inclusion of support for hardware performance counters. We use a 64 GB SDCard to hold the OS image and scratch areas.

The RPi4 cluster

The boards we use in this course (quantity 50) are rack-mounted in a machine room in the William Gates Building, and will be accessed remotely using SSH. As part of the lab work, you will use a Jupyter Lab notebook – a web-based frontend to Python that makes it easy to gather and plot data. You will need to SSH tunnel HTTP from your personal machine to the RPi4 board for this purpose.

You will need to run the labs as the `root` user, so that can apply tools such as DTrace and HWPMC to the kernel itself. We ask that you take great care to avoid bricking your board – e.g., by damaging the OS install! We do have additional boards in the rack, and most likely will be able to recover the board remotely, but it is best to avoid these scenarios – especially as the fix will involve reimaging the board's SDCard.

We strongly recommend that you keep your work mirrored on another machine so that, if we have to provide a replacement board, you lose as little as possible. One way to do this conveniently and easily is to use a tool such as git to keep version history and move your files between the board, your personal machine, and perhaps a service such as GitHub.

Network access to your RPi4

Currently, TCP port 22 to the cluster is blocked from the public Internet. To SSH to the RPi4 boards, you will need to be connected via the UIS or CL VPN, or tunnel via another lab host on the University or CL network. Instructions for the UIS VPN can be found here:

`https://help.uis.cam.ac.uk/service/network-services/remote-access/uis-vpn`

If you need help with this, please get in touch with us.

Getting started

We have preinstalled FreeBSD on your RPi4 board, and separately provided you with its network details and SSH login credentials. We ask that you not change the root password, nor remove any existing SSH keys we have preinstalled – these allow us to more easily maintain the system, as well as support you if you encounter any difficulties.

In order to run various tracing and instrumentation tools on the kernel itself, you will need to run as the root user. We therefore ask you to take significant care not to damage the system installation or configuration, and also to ensure that your data is carefully backed up. While we are able to reimage boards, that will resolve in loss of all data on the board.

As Jupyter will serve its content on the RPi's loopback interface, you will need to configure SSH port forwarding to access it remotely. You will also need to point SSH at the SSH authentication key that we have provided for your node (you can configure `.ssh/config` so as to avoid specifying this manually on the command line each time you use SSH). The details will depend on the operating system you are using on your notebook or desktop, but will typically resemble the following (replacing `XXX` with your RPi node number):

```
% ssh -i id_rsa_rpi4-XXX -L8080:localhost:8080 root@rpi4-XXX.advopsys.cl.cam.ac.uk
```

We have created a directory for you to place your work in, `/data`, and recommend that you keep any files you create in that directory. Once you've logged in as root, you can run your first DTrace script:

```
# cd /data
# dtrace -qn 'BEGIN { printf("Hello world"); exit(0); }'
```

To write the output of a script to a file, you can redirect standard output to a file:

```
# dtrace -qn 'BEGIN { printf("Hello world"); exit(0); }' > data.out
```

The first lab

You can find the supporting materials for the first lab – its JupyterLab notebook in a compressed tarball on your RPi4 in `/advopsys/labs/2022-2023-advopsys-lab1.tbz`. Future lab tarballs will also include benchmark code that you will need to build. You should un-`tar` the file into your `/data` directory before proceeding to the next step:

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

It will extract a JupyterLab notebook named `2022-2023-advopsys-lab1.ipynb`, which demonstrates how to use JupyterLab to run abenchmark, collect results, analyse the data, and present it. It also demonstrates how to use DTrace from within JupyterLab.

Running the Jupyter notebook

Jupyter Lab notebooks provide a web-based UI able to run Python (and other languages), incorporating code, capturing data, and presenting plots in a unified environment. We provide template notebooks for lab reports and assignments, which include starting-point code to run the appropriate benchmark, as well as demonstrate various instrumentation tools.

Once you've logged in as root (as explained in the Getting started section), you can then start Jupyter by running:

```
# cd /data
# jupyter-lab --allow-root --no-browser --port=8080
```

On starting, the JupyterLab prints the URL on which it can be accessed with a Web browser, which will include an embedded access token. Open this URL on the system where you have run the SSH command to get started. You may need to adjust the URL depending on your port-forwarding setup.

JupyterLab Notebooks are identified by the file extension `.ipynb`. To open a Notebook simply click on its name. The selected Notebook will open in a separate tab from which it can be edited and run. The first lab Notebook is named `2022-2023-advopsys-lab1.ipynb`.

Compiling and running the benchmark

The template lab report includes information on how to compile and run the benchmark from within JupyterLab. The template assumes that your current working directory is `/data` when you start running `jupyter-lab`, and may require modification if that is not the case.

Running DTrace scripts

DTrace scripts can be executed within the JupyterLab notebook with the assistance of the `python-dtrace` module. In order to run the a benchmark and also instrument it with DTrace, scripts are executed within a separate thread by instantiating a `DTraceConsumerThread` object. The `DTraceConsumerThread` constructor takes a Python string specifying D-Language script to execute, as well as several other optional parameters. The lab template contains a worked example using DTrace to capture system-call counts in the benchmark. We recommend using guards (illustrated in the Lab 2 template) to limit DTrace tracing to the main benchmark loop itself, and running with only one iteration when using DTrace.

Plotting performance measurements

Performance measurements can be plotted inside the JupyterLab Notebook using the built-in "magic": `%matplotlib inline`. In addition to displaying inline, matplotlib graphs can be saved to the RPi4 filesystem using the `plt.savefig()` function. Saved files can then be copied from the RPi4 to your home system using SSH for inclusion in L41 lab reports. Further information about matplotlib can be found at the project's website:

`https://matplotlib.org`

Troubleshooting

In the event of the JupyterLab Notebook behaving erratically the first port of call is to simply stop the currently executing cell (this can be done from the JupyterLab Notebook toolbar). For more serious problems (such as systemic unresponsiveness) the executing "kernel" can be reset from the Jupyter toolbar. This resets the Python runtime's state and is an effective for most problems. If resetting the kernel does not resolve the issue it may be to terminate and restart the `jupyter-lab` command.

Unreliable networking

It may be that some students experience unreliable networking, which can be disruptive when working on these labs – especially if you are having trouble completing benchmark runs. Here are some things that may help mitigate that lack of reliability:

1. If your SSH session is disconnected, it's a good idea on reconnect to check that state from the prior session has been discarded. If, for example, your IP address changed after a VPN reconnection, then your system may have noticed that the SSH connection has closed, but the SSH server on the Raspberry Pi may still be waiting for the connection to time out – potentially several minutes. Benchmark runs started in the prior session will affect performance in the new one.

   You can use the UNIX `ps ax` command to list all processes running, after you log in, to see if benchmark programs or JupyterLab are still running. If they are, you can use the UNIX `kill` command to terminate them. See the `ps(1)` and `kill(1)` man pages for further details.

2. You may want to consider running JupyterLab under `tmux`, a terminal multiplexer that supports reconnecting to prior sessions. First run `tmux` to create a new session. If you are disconnected and reconnect, you can use `tmux ls` to list running sessions, and `tmux attach -t` *`session`* to reconnect to the specified session.

As long as the terminal being used by JupyterLab – either the one allocated by SSH when you logged in, or one provided by `tmux` – is still active, JupyterLab will continue to run. You can reconnect to it using the same URL you did for the prior terminal session.