P56 - Lab 4, Kernel Tracing - v2.0

Prof Andrew W. Moore

Lent 2023/24

This lab is intended to familiarise you with kernel tracing, the KUtrace system in particular, and the data exploration system it utilises. We will use a combination of simple examples, following the textbook, and then extend these.

Recall that kernel-tracing is a form of instrumentation like what we have used before to create trace files — except the instrumentation is in the operating-system kernel itself. As a result the KUtrace control program will create its own trace files in addition to those created by any other program you have used to date.

Due to the compact form of logging, the conversion of compact, raw, data to logs and from logs to graphical representation has several more steps; these are also described below.

1 Reference material, code, and directories

While Chapter 19 has plenty of information, the most up-to-date instructions for the KUtrace system are in `KUtrace/docs` in particular the `kutrace_user_guide.pdf`

Recently patched versions of tools and kernels have been created for this and subsequent work. To check you have the latest kernel `uname -a` should return

```
$ uname -a
Linux l51-pi002.nf.cl.cam.ac.uk 5.10.110-v8-KUtrace-awm22+ #11 SMP PREEMPT Thu Feb 23 23:39:47 GMT 2023 aarch64 GNU/Linux
```

The kernel name should include the words KUtrace-awm22 and the build date should be Thursday 23rd or Friday 24th of 2023.

If this isn't the case, reboot the host and check again.

Post processing tools such as `makeself` are located in `~/KUtrace/postproc`

2 KUtrace

Kernel part of KUtrace is composed of a kernel patch and a kernel module.

The kernel patch should already be installed on all machines — you dont need to do it yourself.

The kernel module is located in `~/KUtrace/linux/loadable-module`.

To load the kernel module, when in the `~/KUtrace/linux/loadable-module` directory,

```
sudo insmod kutrace_mod.ko  <optional arguments>
```

`kutrace_mod` takes three arguments: `tracemb`, `pktmask` and `pktmatch`.

`tracemb`, the size of the trace buffer, defaults to 2Mbyte, this is not large enough for our work so a default of at least 20 should be used.

```
sudo insmod kutrace_mod.ko tracemb=20
```

`pktmask` and `pktmatch` are used to identify which network packets are captured into the trace file. By default all RPC packets, defined in Chapter 6, will be captured.

To capture every packet you can use this command, but **beware** this may require an even larger buffer to yield useful information.

```
sudo insmod kutrace_mod.ko pktmask=-1
```

To disable packet capture (the default behaviour)

```
sudo insmod kutrace_mod.ko pktmask=0
```

Parameters can only be changed at kernel module load time. To change the parameters, unload the module first

```
sudo rmmod kutrace_mod
```

If you want to check the module is installed use:

```
lsmod | grep kutrace_mod
```

This will return

```
kutrace_mod 24576  0
```

(or similar) if the module is loaded; nothing at all if the module is missing.
Checking the kernel log with `dmesg` can also be a reassuring way to check kutrace is loading correctly.

3 Reproducing the hello world example

An example is given in book (Chapter 19) with relevant files in the `KUtrace` directory.

To reproduce this example

Firstly, use the previous instructions to load the module. (no packet tracing is required at this stage, and a default buffer size is also sufficient.)

Use two terminal windows logged into the same host.
In one window; start the tracing module

```
  ~/KUtrace/control/control/kutrace_control
  Entering kutrace_control
control>  goipcwrap  <<<- this command starts the trace module,
                  tracks instruction count, and
                   operating in continuous (flight-recorder) mode
```

in the second terminal window, execute the hello world program

```
  ~/KUtrace/bookcode/aw_files/book-user-code/hello_world_trace
```

return to the first window and stop the trace

```
  control> stop
```

The book says that you can supply a filename, but this is not the case.

Note: You must keep `kutrace_control` running while running the program to be traced. The timestamps recorded in the trace are taken from `kutrace_control`, not the kernel module (or the program(s) executed.)

When `kutrace_control` is stopped, it will write the binary-format trace log. Now use the following command to convert. Note the multiple sort commands.

```
./rawtoevent <trace file> | LC_ALL=C sort -n | ./eventtospan3 TITLE > trace.json
cat trace.json | LC_ALL=C sort | ./makeself show_cpu.html > trace.html
```

The `postproc` directory has a number of helper scripts including one called `postproc.sh` This permits you to (more simply) use

```
~/KUtrace/postproc/postproc.sh <trace file> "TITLE".
```

4 Disk and trace

Now lets follow Chapter 25 to explore KUtrace when used against a disk-intensive example.

The trace module will require in excess of 8Mbyte so follow the instructions above to reload the kernel module with `tracemb=20` to capture all events.

Like past examples from the book, the code and executable is in
`~/KUtrace/bookcode/aw_files/book-user-code/`
To run the example (using the SD-card for storage in this case)

```
~/KUtrace/bookcode/aw_files/book-user-code/mystery25 ~/speedTestFile
```

For the example from Section 25.9 the mystery command is

```
~/KUtrace/bookcode/aw_files/book-user-code/mystery25 ~/speedTestFile & \
~/KUtrace/bookcode/aw_files/book-user-code/mystery25 ~/speedTestFile2
```

In each case you need to start and stop `kutrace_control` in another terminal window. This last example may result in an extremely large trace and thus a very large html file.

Complete Exercise 25.1

5 Network

A reminder - this example will require a large trace buffer so make sure you have reloaded the module with `tracemb=20`

Experiment 1

The commands are
On the server host

```
~/KUtrace/bookcode/aw_files/book-user-code/server4
```

On the client host

```
~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piZZZ 12345 \
-rep 20000 sink -key "abcd" -value "vvvv" 4000
```

While only RPC packets are captured, if you are interested, you can use the module argument `pktmask=-1` to capture all packets.

Note that we use the client4 argument `-rep` here instead of `-k`: this is to reproduce the same nanosleep behaviour as shown in the book.

The syscall is named `clock_nanosleep` so it shows up as `clock_n~p`)

Experiment 2

On the server host

```
~/KUtrace/bookcode/aw_files/book-user-code/server4
```

On the client host

```
~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piZZZ 12345 \
-rep 200 sink -key "abcd" -value "vvvv" 1000000
```

Additional exercise One noteworthy thing is that the client (pre-send) processing code spends as much time in user-space memcpy as in page faults. This phenomena also featured in Chapter 6; speculate as to the cause of this with justification of your answer.

Experiment 3 and 4

Commands for experiments 3 and 4 (don't forget to run KUtrace too!)
Server 1

```
~/KUtrace/bookcode/aw_files/book-user-code/server4
```

Server 2

```
~/KUtrace/bookcode/aw_files/book-user-code/server4
```

Server 3

```
~/KUtrace/bookcode/aw_files/book-user-code/server4
```

Client

Run KUtrace on this host

By also capturing packets on this machine you will observe traffic among all servers and this host. the `not port 22` will remove (some) irrelevant traffic.

```
sudo tcpdump -n -s128 -w tcpdump.pcap not port 22 &

~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piXXX 12345 \
-rep 20000 sink -key "abcd" -value "vvvv" 4000 &
~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piXXY 12345 \
-rep 200 sink -key "abcd" -value "vvvv" 1000000 &
~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piXXZ 12345 \
-rep 200 sink -key "abcd" -value "vvvv" 1000000
```

Now stop the packet dump

```
fg <ctrl+C>
```

(For me) Experiment 3 is straightforward. Experiment 4 produced results that didn't exhibit the delay behaviour indicated in the textbook; rather they exhibit near-perfect link sharing.
So.....

Experiment 5

A combined (and simplified) version of 3 and 4; this experiment involves one host running multiple instances of server thereby serving multiple clients. Each client is on a different host.

Commands for experiment 5
Server

```
~/KUtrace/bookcode/aw_files/book-user-code/server4
```

Server has the built-in commands to run KUtrace on this host

By also capturing packets on this machine you will observe traffic among all clients and this host. the `not port 22` will remove (some) irrelevant traffic.
So in another window on the server:

```
sudo tcpdump -n -s128 -w experiment5.tcpdump.pcap not port 22
```

Clients

Use the other three pi hosts to each run a copy of client4, make sure each client4 is accessing the same server and attempting access at the same server port. This changes the point of contention and can change the dynamic. You may wish to simply repeat the earlier client commands (changing the server host). It is tricky to simultaneously launch the clients (particularly if you wish to run kutrace for each client); so you may wish to make the runtime longer (warning the trace might get very long); or use multiple ssh commands for a simultaneous launch of the clients.

```
Client 1

~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piXXX 12345 \
-rep 20000 sink -key "abcd" -value "vvvv" 4000
```

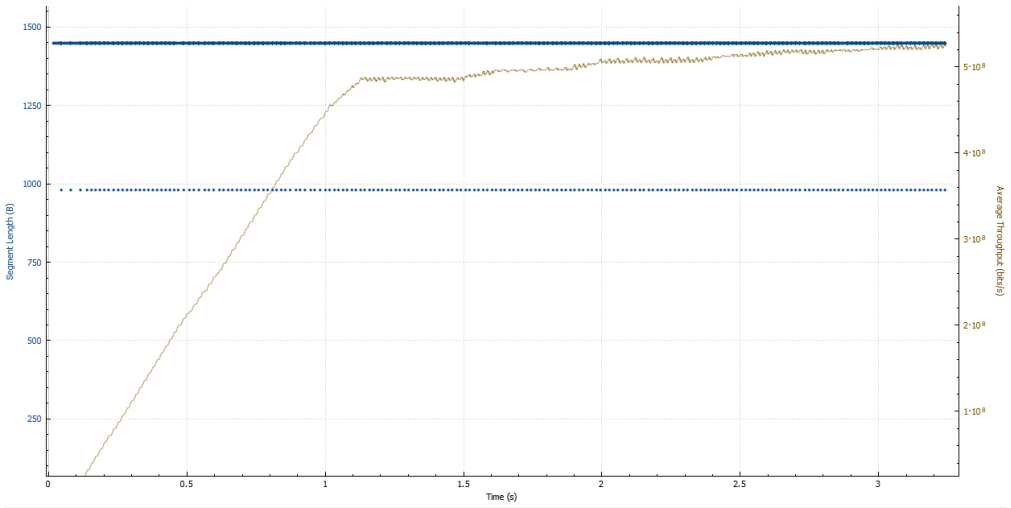

Figure 1: Acknowledgement: Gary Guo. This illustrates both throughput and segment size (in each direction); however, for a submission the labels/axis font is too small.

```
Client 2

~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piXXX 12346 \
-rep 200 sink -key "abcd" -value "vvvv" 1000000

Client 3

~/KUtrace/bookcode/aw_files/book-user-code/client4 l51-piXXX 12347 \
-rep 200 sink -key "abcd" -value "vvvv" 1000000
```

Once again, once the clients stop, stop the packet dump(s).

Examine it with wireshark (or your libpcap personal choice.) wireshark can show the link utilization which will illustrate (fair?) network use among clients. Don't forget a packetdump at the server will include intermingled the traffic from the three different clients.

6 Saving Your Experiments

Make sure to back up your experiments, including (but not limited to) Jupyter notebooks, dump files and scripts. Remember that multiple teams may use the same test machines, so be careful when handling data.

All the measurements are saved under your crsid folder, so backing up the entire folder is a good idea. To copy a remote directory onto your local machine:

`sftp l51@<hostname>.nf.cl.cam.ac.uk` and `get -r <directory>`.
There are also other ways to copy a remote directory, you are welcome to use those as well. You may wish to compress results files in order to save space.
Exporting a Notebook as `.tex` will save graphs as separate files, which you can then include in your lab report.

Please do not push any changes, data or results directly to P56 repository. You can fork the repository to your own user and push changes there. If you would like to suggest a correction or an enhancement to a notebook or a script, please use pull-requests.

7 Understanding Your Measurements

Consider the following prompts to help you understand your results, and encourage you to identify/add supporting evidence for the writeup. However, they are just suggestions - feel free to approach the data differently!

- Discuss the methodology of the measurement tools.
- Explain how the limitations of the methodology are mitigated in this lab.
- Explore the limitations of the experiments conducted in this lab, and explain where the quality of the experiment (e.g., setup, methodology) could have been improved or altered.

You should always look for odd or surprising results (such as our experience with experiment 4), and try to explain them. Note that sometimes exceptional results indicate a problem in your setup or scripts.