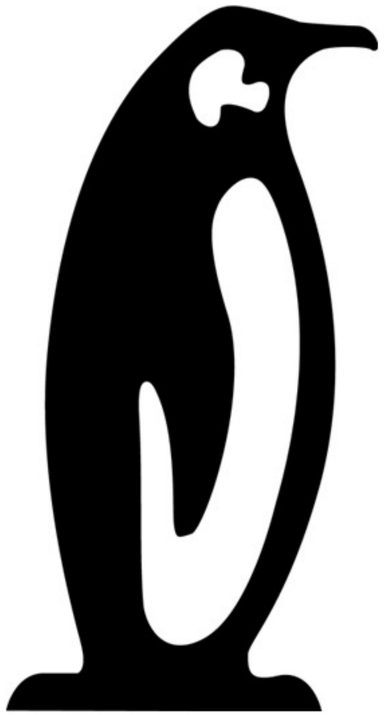

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Tooling for semantic probing based on BPF and kernel tracing

dr. Kris Van Hees
kris.van.hees@oracle.com
Linux Engineering
Oracle

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Agenda

- What are we doing…
- Why we are doing it...
- How are we doing it...
- Opportunities and issues
- Probing kernel functions: issues
- Probing tracepoints: issues
- Where to find more information...

What are we doing...

- Kernel provides great tracing features:
  - kprobes, uprobes
  - ftrace
  - tracepoints
  - BPF
- Great building blocks
- Very specific (some more than others)

Semantic probing is to kernel tracing features (probes)
what high-level libraries are to C library functions

What are we doing... (cont.)

- DTrace provides a documented set of providers with semantic probes
  - proc
  - sched
  - io
  - lockstat
  - ip, tcp, udp
  - profile (and tick)

Why are we doing it...

- Main features:
  - Well-defined names for semantic events
  - Well-defined probe arguments

- Hiding (from the user) any dependency on:
  - Architecture
  - Device features
  - Implementation details
  - Kernel version

And sometimes synthesizing multiple events into one semantic one

How are we doing it...

- Define the set of semantic probes
  - Easy for DTrace: already done when DTrace was first developed
- Figure out how to:
  - Ensure that the semantic event is reported as a probe firing when needed
  - Ensure that we can populate the necessary argument values

How are we doing it... (cont.)

- Probe: io:::done (fires when an I/O request completes)
  - Requires triggering from multiple locations (and depends on kernel):
    - rawtp:block::block_bio_queue
    - rawtp:nfs::nfs_initiate_read (kernel >= 5.6.0)
    - fbt:nfs:nfs_initiate_read:entry (kernel <= 5.5.19)
    - rawtp:nfs::nfs_initiate_write (kernel >= 5.6.0)
    - fbt:nfs:nfs_initiate_write:entry (kernel <= 5.5.9)
  - NFS requests do not have a bio like other I/O requests do
    - we create a fake one.

How are we doing it... (cont.)

- Probe: io:::wait-done (fires when done waiting for an I/O request to complete)
  - Requires multiple probes:
    - fbt::submit_bio_wait:entry
      - Record pointer to bio in a TLS variable
    - fbt::submit_bio_wait:return
      - Retrieve pointer to bio (and delete TLS variable)

    - For XFS, we need a different probe (just one):
      - rawtp:xfs::xfs_buf_iowait_done
        - But that needs a fake bio

Opportunities and issues

- Lots of opportunity for common code
- Perfect scenario for pre-compiled BPF code
  - GCC and binutils BPF support has been crucial for this
  - DTrace also has a built-in compiler (D to BPF)
  - DTrace has a custom linker (link dynamically compiled code with pre-compiled code)

- Sadly, BPF does not provide a good way (that I have found) to share functions between BPF programs
  - Would be very useful here
  - Should be possible if the functions can be verified based on their type data

Opportunities and issues (cont.)

- More generic code can be challenging for BPF
- Makes toolchain use more challenging
  - Balance between convenience and ***what works***

- BPF verifier must be able to guarantee that the code is safe
  - Add hints (conditionals) to ensure the verifier knows what the boundaries are
  - **Compiler optimization may remove those**
  - Loops are still a challenge
  - Stack use can pose unpleasant surprises
    - 1024 bytes is a really low limit!

Opportunities and issues (cont.)

- Custom compiler (self-contained)
  - D as source “language”
  - All compilation done on-the-fly (no objects stored)
- Custom linker
  - Linking on-the-fly compiled code with pre-compiled code (GCC / binutils)
  - Need to do a lot of relocation resolving
    - Various data items cannot be known until the final BPF program is constructed
    - Pre-compiled common code often needs those data items

A lot of *going back to the basics* of toolchain development!

Probing kernel functions: issues

- kprobe/kretprobe:
  - Hard to know which functions can be probed
    - Trial-and-error is very slow
  - Need to retrieve arguments yourself
- fentry/fexit:
  - Much more dependable
  - Faster
  - But still with limitations

Probing kernel functions: issues (cont.)

- Accessing pass-by-value struct arguments
  - Did not work for trampoline-based tracing probes (fentry)
  - Support was added for x86_64, and perhaps now also arm64
  - Still no general support for this
  - If there is no support, the BPF verifier rejects the program
    - Less user friendly than I'd like

- Possible solutions involve getting to the arguments ourselves
  - Implementation (and architecture) dependent
  - May run unto other BPF verifier complaints

Probing tracepoints: issues

- Tracepoints are a great source for semantic tracing
- They tend to move with the source code in new versions (convenient!)
- A few have > 10 arguments, which BPF does not support

- Accessing extra arguments is tricky:
  - Highly dependent on the calling mechanism for tracepoints
  - May be architecture dependent as well

- Unclear on whether it is worth it
  - Only 54 tracepoints out of 1893 have > 10 arguments

Where to find more information...

- DTrace for Linux
  - GitHub: https://github.com/oracle/dtrace-utils.git
  - Mailing list: dtrace@lists.linux.dev

Thank you!

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024