Linux NFS: Using Tracepoints

A new troubleshooting paradigm

Chuck Lever <chuck.lever@oracle.com>

Today’s Take-aways

What are Linux tracepoints?

Why have we replaced the venerable dprintk?

How do I enable and view trace events?

Tracepoints Replace dprintk()

Efficiency

- A single trace event happens entirely in memory and involves no I/O to the console or a log file
  - Therefore, tracepoints are not rate-limited like log messages are
- You can enable one trace event at a time or whole subsystems at once
- You can filter trace events while tracing, or afterwards

Tracepoints Replace dprintk()
Precision

- Each trace event record contains:
  - A microsecond-precision timestamp
  - The CPU ID
  - The pid and command that was running on that CPU
  - IRQ state

Tracepoints Replace dprintk()

Integration

- Tracepoints can work in conjunction with kprobes, eBPF, or SystemTap
- You can enable multiple trace subsystems at once (e.g., nfsd, kmem, and sched)
  - The trace log is interleaved and timestamped
- You can enable tracepoints along with other tracing plug-ins, like function_graph
- You can record a stack trace when every enabled tracepoint fires

Tracepoints Replace dprintk()

NFS and sunrpc related features

- NFS/sunrpc trace events can name a particular RPC task, nfsd thread, XID, or endpoint address
- NFS/sunrpc trace events usually display information symbolically rather than as raw numbers (raw data is still available)
- We've created two categories:
  - Control flow - chatty, report on resource usage or normal events
  - Exceptional - name ends in "_err", fire rarely

Using trace-cmd

How to discover available trace events

- Use “trace-cmd list”
  - Each available event is displayed as “<subsystem>:<event name>”
  - Trace events in modules that are not loaded are not available

Using trace-cmd

Enabling and disabling tracepoints

- Once you have selected the set of tracepoints you want to enable, use:
  - trace-cmd start -e <trace point> [-e <trace point> …]
  - trace-cmd stop
  - trace-cmd reset

Using trace-cmd
Displaying the trace buffer

- Two ways to go:
  - Once tracing has stopped, use “trace-cmd show > <output file>” to save the ring buffer
    - Does not consume the content of the ring buffer
    - Capture is limited to the size of the ring buffer
  - While tracing is enabled, use “trace-cmd show -p” to tail the trace output pipe. This consumes all ring buffer content

Using trace-cmd

Capturing trace activity over time

- “trace-cmd record -e <trace point> [-e <trace point> …] [<command>]” captures long-running activity (like an unbounded network capture)
- Command line options can filter by pid, by CPU, etc.
- To end capture, Ctrl-C the trace-cmd program
  - The signal causes trace-cmd to write the trace buffer into “trace.dat”
  - This command captures continuously, so trace.dat can become very large

Using trace-cmd

Filtering a capture

- To view the captured trace events, use “trace-cmd report”
  - trace.dat carries trace event format specifiers and other metadata, and can be copied to other systems
- Simple text-processing tools like awk, grep, and less can operate on the output of “trace-cmd report”
- These can be used in combination with filtering:
  - “trace-cmd report -F <trace point filter>”
  - “trace-cmd report -R”

Using trace-cmd

Supplemental material

- The trace-cmd man pages: trace-cmd-record, trace-cmd-report, etc.
- Documentation/trace/
- https://lwn.net/Articles/410200/
- https://github.com/rostedt/trace-cmd