Linux Tracing: LTTng vs. SystemTap

Fabian Bornhofen

Software Profiling, Summer 2013
Supervisor: Dr. Peter Tröger
Hasso-Plattner-Institut

2013-05-13

Agenda

Linux Tracing

- Perf: performance counters mainlined in 2009
- Ftrace: static kernel function tracing mainlined ca. 2008
- Strace, OProfile, ...
- **SystemTap**: dynamic kernel & user space tracing, started in 2005
- **LTTng**: efficient, large-scale kernel & user space trace tools, started in 2005

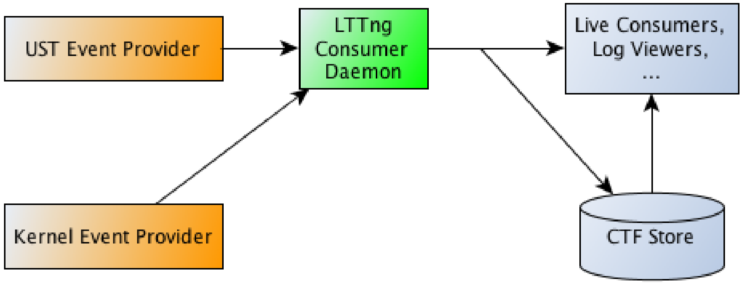
Which one to pick?

LTTng

LTTng: Linux Trace Toolkit – next generation

- Comparable to Event Tracing for Windows
- Efficient tracing tools for debugging and performance analysis
- Static trace points in kernel, user space library available
- Record huge amounts of trace data in common format (CTF)
- Crucial: trace viewer (Eclipse, LTTV (WIP))
- Workflow: post-mortems (flight recorder) or live consumers (e. g. LTTngtop)

LTTng – Architecture



More detail:
http://lttng.org/sites/lttng.org/files/LTTng2_0Architecture_pa3.pdf

LTTng – Kernel Tracing

- Kernel Tracepoint API (mainline) by Mathieu Desnoyers
- Tracepoints present in a number of subsystems (sched, kvm, block, timer, ...)
- `lttng list -k` lists available kernel events
- Add context information, filters, ... to sessions

LTTng - UST (userspace tracer)

- Add tracepoints to userspace programs
- Cheap activation (no traps)
- How to?
  - Write event definition
  - Compile definition to C (`lttng-gen-tp`)
  - Add tracepoints to program
  - Run tracing session
  - Analyze

LTTng – UST (userspace tracer)

```
TRACEPOINT_EVENT(
    provider_name,
     event_name,
     TP_ARGS(arg1_type, arg1_name),
     ...
     TP_FIELDS(
          ctf_type(type, field_name, arg_expr)
     )
)
...
tracepoint(provider_name, event_name,
           arg1, ..., argN);
```

SystemTap

SystemTap

- Similar to DTrace
- User defined or predefined tracing scripts
- SystemTap compiles scripts to C (kernel modules or user space probes)
- Static probes are possible, too
- Text-based traces; TTY, file output or flight recorder
- Workflow: iterative debugging, flight recorder mode for monitoring
- Guru mode: manipulate syscall parameters, ...

SystemTap – Sample Script

Case Study

http://trac.nginx.org/nginx/ticket/53

nginx bug #53 – deadlock after active worker crashes

- Lightweight web server with worker processes, sync via atomic GCC primitives in shared memory region (no syscalls involved)
- Synchronized access to incoming requests
- Active worker := worker holding that lock
- If active worker terminates abnormally, server deadlocks

nginx #53 – post mortem with LTTng

- Trace: run nginx, request files, kill active worker, request files, nothing happens
- Server does not accept connections
- Search trace for sys_accept4
- Last sys_accept4 event appears before a worker was killed
- As developers, we know that nginx protects sys_accept4 with a mutex
- Might conclude that mutex is lost after active worker terminates.
- Possibly introduce UST trace points in C code

nginx #53 – debugging with SystemTap

- Syscall tracing is one option
- Fine-grained analysis: which worker holds the accept mutex at what time?
- Approach 1: entry/exit user space function tracing (lock/unlock) ... which is currently not implemented (used to be, will be?)
- Approach 2: trace statements in code
- Side note: could measure fairness of lock

Conclusion

Why or why not LTTng?

- Low-level tracing, large amounts of data
- Might allow for cheap monitoring
- Switching to Common Trace Format
- Tools are WIP, transition from 1.x to 2.x
- Documentation is WIP as well, everything in flux, lots of dev mailing list traffic
- Depending on distribution, setup may require manual steps

Why or why not SystemTap?

- Flexible tool for debugging and performance analysis
- Get an understanding of 3rd-party code
- Setup requires kernel debug information
- Very detailed and accessible documentation
- Reusable scripts for kernel and user mode software
- fun.

Choosing the right tool

- Syscall tracing is always low-hanging fruit
- Post-mortem analysis, monitoring (flight data recorder): LTTng
- Low-impact performance analysis (dev. drivers etc.): LTTng
- Build with tracing in mind: LTTng
- Understanding control flow: SystemTap
- Customized performance counters: SystemTap
- Tinkering: SystemTap

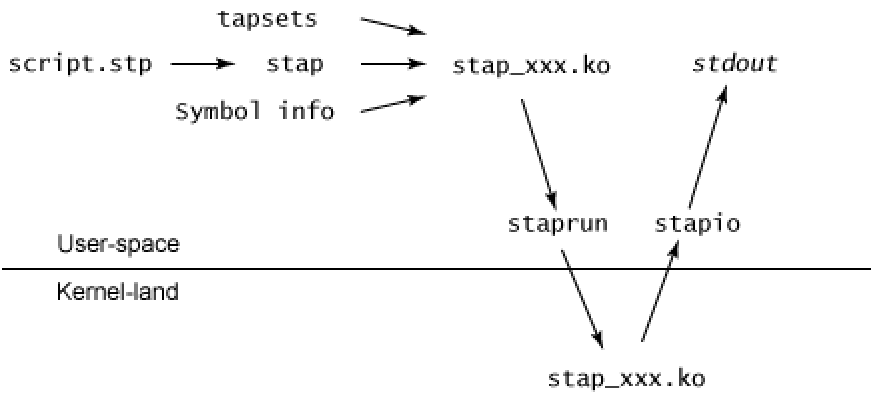
Questions?

References

- Linux Tracing Overview
  - ftrace http://lwn.net/Articles/322666/
  - single buffer https://lwn.net/Articles/388978/
  - perf mainlined http://lwn.net/Articles/339361/
  - utrace http://lwn.net/Articles/224772/, http://lwn.net/Articles/371210/, https://lwn.net/Articles/499190/
- LTTng https://lttng.org/
- LTTng Architecture http://lttng.org/sites/lttng.org/files/LTTng2_0Architecture_-pa3.pdf
- SystemTap http://sourceware.org/systemtap/documentation.html

Backup

SystemTap – Architecture



http://www.ibm.com/developerworks/linux/library/l-systemtap/

nginx #53 – LTTng UST

```
TRACEPOINT_EVENT(
    nginx,
    accept_lock_acquire,
    TP_ARGS(int, pid),
    TP_FIELDS(
        ctf_integer(int, pid, pid)
    )
)
...
tracepoint(nginx, accept_lock_acquire, getpid());
```