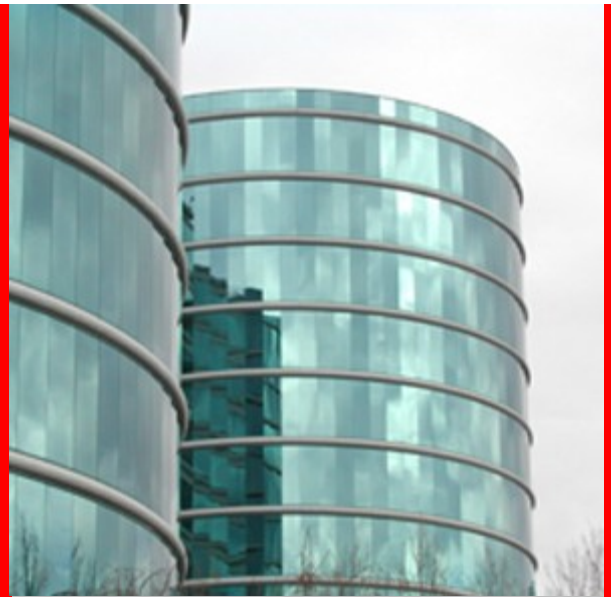

ORACLE®

Tracing on Linux Updates

Elena Zannoni (elena.zannoni@oracle.com)
Linux Engineering, Oracle America

May 30 2013

A Look at the Building Blocks

- Kprobes
- Tracepoints
- Uprobes

Kprobes: Dynamic Kernel Tracing

- Started by IBM in 2004
  http://www.ibm.com/developerworks/library/l-kprobes/index.html
- Merged into the Linux kernel in 2005
- Allow tracing of running kernel
- Must be configured CONFIG_KPROBES=y
- Main concept is similar to debugger breakpoints: place breakpoint instruction at desired code location
- When hit, exception is caused
- Exception handler executes actions associated with kprobe
- Optimizations to kprobes using Jumps instead of exceptions
- Used by Systemtap, ftrace and perf

Events Markers: Static Kernel Tracing

- Static probe points in kernel code
- Independent of users (ftrace, perf, Lttng, systemap,....)
- Many exist in the kernel in various subsystems, and being added
- **`TRACE_EVENT()`** macro with 6 arguments
- Definitions in **`include/linux/tracepoint.h`**
- Define characteristics of tracing actions (probe) using **`TRACE_EVENT()`** in **`include/trace/events/*.h`**
- Mark tracing locations with function call **`trace_my_event_name()`**
  - e.g. **`trace_sched_switch()`** in **`sched.c`** and **`TRACE_EVENT(sched_switch,...)`** defined in **`include/trace/events/sched.h`**
- Read the 3-article series:
  - http://lwn.net/Articles/379903/
  - http://lwn.net/Articles/381064/
  - http://lwn.net/Articles/383362/

Uprobes: Dynamic Userspace Tracing

- Work started in 2007
- In Linux kernel starting from 3.5

- Handle userspace breakpoints in kernel
- Analogous to kprobes
- Uses breakpoint instruction
- No signals / context switches
- Multiple tracers allowed
- Ptrace replacement eventually?

Uprobes: Details

- Implementation based on inodes
- Must be enabled in kernel builds with CONFIG_UPROBES=y
- Uprobes described as: inode (file), offset in file (map), list of associated actions, arch specific info (for instruction handling)
- Probes stored in an rb_tree
- Register a uprobe: add probe to probe tree (if needed), and insert the arch specific BP instruction
- Handle the uprobe by calling the actions
- Resume to userspace
- Multiple consumers per probe allowed (ref count used)
- Conditional execution of actions is possible (filtering)

Uprobes: Execution Out of Line (XOL)

- Replace instruction with breakpoint
- Store original instruction in separate memory
- Execute instruction w/o reinserting it
- Necessary for multithreaded cases: breakpoint remains in place. Any thread can hit it. No need to stop all threads

Uretprobes: Return Probes

- Place probes at exit of functions.
- Done in two steps:
  - Place probe at entry
  - When this is hit, its handler places retprobe at return address.
- Note: retprobe location is after called function ends

Latest Developments:

- Uretprobes just added to mainline in the 3.10 merge window: https://lkml.org/lkml/2013/4/15/442
- SystemTap support for latest version of uretprobes (since March 2013)

Uprobes and Uretprobes: Status

- Uprobes: Supported Architectures: x86, x86_64 (3.5, 3.6 kernels), powerpc (3.7 kernel), ARM (initial patch in October 2012, stalled?)
- Uretprobes: x86, X86_64, Powerpc (3.10 kernel)
- Supported by perf, ftrace, systemtap

In Kernel Tools: Ftrace

Ftrace: Some Background

- Kernel tracer
- Monitor many different areas and activities in the kernel
- Main Maintainer: Steven Rostedt, Started in 2008

- Interface via /sys/kernel/debug/tracing (both control and output)
- GUI for data visualization: KernelShark
  - http://git.kernel.org/cgit/linux/kernel/git/rostedt/trace-cmd.git/
  - http://people.redhat.com/srostedt/kernelshark/HTML/
- Trace-cmd: user space tool with subcommands
- Documentation: in kernel tree (newly updated in 3.10)
  - Documentation/trace/ftrace.txt
  - Documentation/trace/ftrace_design.txt
- See old introductory articles: http://www.lwn.net/Articles/365835/ http://www.lwn.net/Articles/366796/

Ftrace: Build Configuration

Kconfig options: lots of them!

- CONFIG_TRACING=y
- CONFIG_FTRACE=y
- CONFIG_FUNCTION_TRACER=y
- CONFIG_NOP_TRACER=y
- CONFIG_KPROBE_EVENT=y
- CONFIG_UPROBE_EVENT=y
- CONFIG_DYNAMIC_FTRACE=y
- CONFIG_FUNCTION_GRAPH_TRACER=y
- [...]

Most tracers are enabled by default in Fedora kernels.

Ftrace: Control and Output

- /sys/kernel/debug/tracing directory: contains all the control and output files
- current_tracer: which tracer is in effect (could be NOP)
- tracing_on: writing to buffer is enabled
- trace: output buffer (circular, will overwrite)
- trace_pipe: output from live tracing
- available_events: which events (static points in kernel) are available
- available_tracers: which tracers are available (relates to kconfig options, for instance function_graph, function, nop)
- kprobe_events, uprobe_events: written to when a kprobe (uprobe) is placed, empty if none
- options, instances, events, per_cpu, trace_stats: directories
- [....]

Ftrace: Tracers (Plugins)

- *Function*: trace entry of all kernel functions
- *Function-graph*: traces on both entry and exit of the functions. It then provides the ability to draw a graph of function calls
- *Wakeup:* max latency that it takes for the highest priority task to get scheduled after it has been woken up
- *Irqsoff*: areas that disable interrupts and saves the trace with the longest max latency
- *Preemptoff*: amount of time for which preemption is disabled
- *Nop*: trace nothing. Useful to stop output of traces to buffers, while tracing is still enabled
- Controlled by current_tracer file

Ftrace: Dynamic Tracing (Kernel & User)

- Use /sys/kernel/debug/tracing/kprobe_events and /sys/kernel/debug/tracing/uprobe_events to control from command line
- Read more: Documentation/trace/kprobetrace.txt and uprobetracer.txt
- LWN article: http://lwn.net/Articles/343766/

Set kretprobe:

echo 'r:myretprobe do_sys_open $retval' > /sys/kernel/debug/tracing/kprobe_events

Set uprobe:

echo 'p: /bin/bash:0x4245c0' > /sys/kernel/debug/tracing/uprobe_events

Clear them:

echo > /sys/kernel/debug/tracing/kprobe_events

echo > /sys/kernel/debug/tracing/uprobe_events

Ftrace: trace-cmd

- User space tool, many options, very flexible
- Some commands are:
  - *Record*: start collection of events into file trace.dat
    - Can use ftrace plugins (-p) or static kernel events (-e)
  - *Report*: display content of trace.dat
  - *Start*: start tracing but keep data in kernel ring buffer
  - *Stop*: stop collecting data into the kernel ring buffer
  - *Extract*: extract data from ring buffer into trace.dat
  - *List*: list available plugins (-p) or events (-e)
- Version 2.2.1 is latest (March 2013)
- http://git.kernel.org/cgit/linux/kernel/git/rostedt/trace-cmd.git/
- http://lwn.net/Articles/410200/ & http://lwn.net/Articles/341902/

Ftrace: New in 3.10

- Instances
- Tracing triggers
- Finer control of tracing options and parameters
- Additional trace clocks: “perf” (same used by “perf”, useful for possible interleaving), “uptime” (using jiffies)
- Updated documentation

Ftrace: Instances

- New in 3.10 kernel
- Multiple output buffers are now possible
- Instances are basically subdirectories in /sys/kernel/debug/tracing/instances
- Created by user (with mkdir) but populated automatically
- Each instance contains control files and output buffers (similar to main “tracing” directory) but is independent
- Each instance buffer records events enabled in the control file for that instance
- Events only for now. No tracers.
- An instance can be deleted by removing its directory (rmdir)

Ftrace: Tracing Triggers

- Fine grained control of events (static tracepoints) being recorded
- Enable an event conditionally, only when a certain function is entered, instead of being always on or always off. (or only the first N times the specified function is entered)
- Disable an event conditionally, when a function is hit
- Must be one of the events listed in /sys/kernel/debug/tracing/available_events
- Syntax:

```
echo 'add_timer_on:enable_event:timer:timer_start' >
/sys/kernel/debug/tracing/set_ftrace_filter

echo 'add_timer_on:enable_event:timer:timer_start:5' >
/sys/kernel/debug/tracing/set_ftrace_filter

echo 'add_timer_on:disable_event:timer:timer_start' >
/sys/kernel/debug/tracing/set_ftrace_filter
```

Ftrace: More Tracing Triggers/Filters

- Collect stack traces conditionally, only when a function is hit
- Syntax:

```
echo 'foo:stacktrace' > /sys/kernel/debug/tracing/set_ftrace_filter
echo 'foo:stacktrace:5' > /sys/kernel/debug/tracing/set_ftrace_filter
echo '!foo:stacktrace' > /sys/kernel/debug/tracing/set_ftrace_filter
```

- Save a snapshot of the trace buffer when a function is hit
- Syntax:

```
echo 1 > /sys/kernel/debug/tracing/snapshot
cat /sys/kernel/debug/tracing/shapshot
```

In Kernel Tools: Perf

Perf

- In kernel (tools/perf directory) userspace tool
- Started in 2008
- Main contributors : Ingo Molnar, Frederic Weisbecker, Arnaldo Carvalho De Melo, Namhyung Kim, Jiri Olsa and others
- Started as hardware performance counters interface, initially called perf counters.
- Has grown into all encompassing tracing system
- Still very active
- Documentation: tools/perf/Documentation

Perf subcommands

- *Perf stat:* collects and display events data during a command execution
- *Perf record*: run a command, store its profiling (sampling mode) in output file (perf.data)
- *Perf report*: display data previously recorded in output file (perf.data)
- *Perf diff*: diff between perf.data files
- *Perf top*: performance counter profile in real time
- *Perf probe*: define dynamic tracepoints (more on this later)
- *Perf kmem*: show statistics on kernel mem (from perf.data), or record kernel mem usage for a command
- *Perf trace-perl* (*trace-python*): process previously recorded trace data using a perl script (python script)

Perf subcommands (continued)

- *Perf list*: list available symbolic events types (in exadecimal encoding)
- *Perf annotate*: display perf.data with assembly or source code of the command executed
- *Perf lock*: lock statistics
- *Perf sched*: scheduler behaviour
- *Perf kvm*: perform same as above but on a kvm guest environment with subcommands:
  - *Perf kvm top*
  - *Perf kvm record*
  - *Perf kvm report*
  - *Perf kvm diff*

Perf: Types of events

- *Perf list* shows the list of predefined events that are available
- Use one or more of the events in perf stat commands:

  perf stat -e instructions sleep 5
- **Hardware**: cpu cycles, instructions, cache references, cache misses, branch instructions, branch misses, bus cycles. Uses libpfm. CPU model specific.
- **Hardware cache**: for instance L1-dcache-loads, L1-icache-prefetches
- **Hardware breakpoint**
- **Raw**: hexadecimal encoding provided by harware vendor
- **Software**: in kernel events. Such as: cpu clock, task clock, page faults, context switches, cpu migrations, page faults, alignment faults, emulation faults.
- **Static tracepoints**: needs the ftrace event tracer enabled. Events are listed in /sys/kernel/debug/tracing/events/*
- **Dynamic tracepoints** (if any are defined)
- See include/linux/perf_event.h

Other Control Parameters

- Process specific: default mode, perf follows fork() and pthread_create() calls
- Thread specific: use - - no-inherit (or -i) option
- System wide: use -a option
- CPU specific: use -C with list of CPU numbers
- For running process: -p <pid>
- To find what binaries were running, from a perf.data file, use Elf unique identifier inserted by linker:
  - *perf buildid-list*
- User, Kernel, Hypervisor modes: count events when the CPU is in user, kernel and/or hypervisor mode only. Use with *perf stat* command.

Perf: Dynamic Tracing (Kernel & User)

- Probes can be inserted on the fly using *perf probe*
- Defined on the command line only
- Syntax can be simple or complex
- Can use dwarf debuginfo to show source code (--line option) before setting probe (uses elfutils libdw like systemtap), or to use function name, line number, variable name
- Abstraction on ftrace kprobes, alleviates usage problems, supports also same syntax to specify probes
- Some options: --add, --del, --line, --list, --dry-run, --verbose, --force
- Read more: tools/perf/Documentation/perf-probe.txt

- Example: set userspace probe:
  - perf probe -x /lib/libc.so.6 malloc

Perf: Some Recent Improvements

- Perf diff can now diff multiple data files, and can use different diff methods.
  - Delta: %Value1 - %Value2 Difference between percentages of each value
  - Ratio: Value1 / Value2
  - Weighed: Value1*Weight1 – Value2*Weight2
  - If multiple data files are used they are diffed in turn against the baseline (first file given)
- Events can be grouped in perf annotate output (-g option). Will show multiple columns
- Perf mem: sample memory accesses (load and store)

New: Ftrace and Perf Integration

- Add Ftrace as a new Perf command, with subcommands
- Initial work supports function and function graph tracers only
- Still work in progress. Currently at Version 2 of the patchset
- Syntax:
  - Perf ftrace live: start trace and stream output to stdout (uses ftrace trace_pipe output)
  - Perf ftrace record: start trace and record data into per CPU files
  - Perf ftrace show: show the recorded output (equivalent to trace-cmd output)
  - Perf ftrace report: report the recorded output in perf style format
- Author: Namhyung Kim
- http://git.kernel.org/cgit/linux/kernel/git/namhyung/linux-perf.git branch perf/ftrace-v2

Perf ftrace: More Control Parameters

- Perf ftrace live and perf trace record: parameters are either a command during whose execution the trace data is collected or the following options
  - -C (specify a CPU), -p (specify a pid), -t (which tracer), -a (system wide)
- Perf trace report: -s (sort histograms by keys), -D (dump raw data), -I (display additional info on system)
- Perf trace show: no additional parameters

Some Stats

Number of Different Contributors (based on 3.9.3 kernel)		
	Ftrace	**Perf**
2008	38	0
2009	72	59
2010	55	73
2011	34	63
2012	40	71
2013	20	26

Other Tools

- Systemtap
- LTTng
- Ktap
- Dtrace

LTTng

- Started in 2006 (LTT Next Generation)
- http://lttng.org/
- Userspace tracing via markers and tracepoints (static instrumentation). Need to link with special library (UST).
- kprobes/tracepoints support for kernel tracing
- Included in some embedded Linux distributions
- Released Version 2.1 (Dec 2012), getting closer to release 2.2
- Uses common trace format
- Multiple visualization tools
  - Eclipse integration
  - GUI to view events (LTTV)
  - See Babeltrace tool (command line)

SystemTap

- Started in 2005
- Multiple maintainers: Red Hat, IBM, others
- http://sourceware.org/systemtap/wiki
- Kernel tracing using kprobes
- Userspace tracing using uprobes
- Dynamic probes and tracepoints
- Has well defined rich scripting language
- GUI available
- Allows Guru mode for “delicate” operations, such as modifying memory
- Can use debuginfo
- Uses gcc
- Remote operation allowed
- Latest release 2.2.1 (May 2013)

Ktap

- New in kernel tracer
- Project started in January 2013. Announced 0.1 last week.
- Hosted on https://github.com/ktap/ktap.git
- See talk by Jovi Zhang tomorrow about Ktap
- In kernel interpreter
- Scripting language is Lua
- Targeted to embedded community
- Similar design to DTrace

DTrace on Linux: Background

- Porting to Linux, is an Oracle effort
- A Solaris tool, available since 2005
- Want to offer compatibility with existing DTrace scripts for Solaris
- Expertise of Solaris user and administrators can be reused on Linux
- Customer demand
- Initially released on Linux in October 2011, still progressing

DTrace on Linux: Some Details

- Code is here http://oss.oracle.com/git/
  - linux-2.6-dtrace-modules-beta.git
  - linux-2.6-dtrace-unbreakable-beta.git
- Integrated with Oracle Unbreakable Enterprise Kernel:
  - Version 0.3.2 (released December 2012)
  - Available for UEK2 kernel (2.6.39 based)
  - Available as a separate UEK2 based kernel plus userspace portion
  - Available on ULN channel: ol6_x86_64_Dtrace_latest

DTrace on Linux: Current Functionality

- Functionality available in current release (0.3.2):
  - DTrace provider
  - syscall provider
  - SDT (statically defined tracing)
  - Profile provider (partial)
  - Proc provider
  - Test suite ported and extended
- x86_64 only
- Kernel changes are GPL
- Kernel Module is CDDL

DTrace on Linux: Upcoming Features

- Rebased on 3.8 kernel
- New feature: User Space Statically Defined Tracing
- Improved testsuite

Dtrace USDT Example

- Use USDT probes to provide Function Boundary Tracing functionality
- Place USDT probes at entry and exit of function in your program
- Output in call graph format

Example: Instrumented C Program

```
#include <linux/sdt.h>

#define ENTRY()        DTRACE_PROBE(ufbt, __entry);
#define RETURN()       DTRACE_PROBE(ufbt, __return);

void sync(void)
{
ENTRY();
RETURN();
}
void task1(void)
{
ENTRY();
RETURN();
}
void task2(void)
{
ENTRY();
sync();
RETURN();
}

void task3(void)
{
ENTRY();
RETURN();
}
[.....]
```

Example: Output

```
CPU FUNCTION
  1 | :BEGIN
  3  -> main
  3    -> worker
  3      -> subroutine
  3        -> task1
  3        <- task1
  3        -> task2
  3          -> sync
  3          <- sync
  3        <- task2
  3        -> task3
  3        <- task3
  3      <- subroutine
  3      -> sync
  3      <- sync
  3    <- worker
  3    -> sync
  3    <- sync
  3    -> output
  3    <- output
  3  <- main
```

Example: D Script

```
provider ufbt {
probe __entry();
probe __return();
};
```

Conclusion

Tracing: General Open Issues

- The Big One: status of KABI for kernel tracepoints. See this old article: https://lwn.net/Articles/442113/
- Scalability: still not there with the current tools. See this old article: http://lwn.net/Articles/464268/
- Code integration:
  - Infrastructure (probably not happening)
  - Tools (it's actually happening, see ftrace + perf)
- Embedded community and enterprise community are both users (See Ktap)
- Users ask for
  - Low footprint and low overhead
  - No kernel rebuilds
  - High level consolidation of collected data (visualization)
  - Data filtering (e.g. confidential data)

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.