Using the LTTng Tracer for System-Wide Performance Analysis and Debugging (Hands-on Tutorial)

Presentation and files at:
http://www.efficios.com/elc2010

E-mail: mathieu.desnoyers@efficios.com
> Presenter

- Mathieu Desnoyers
- EfficiOS Inc.
  - [http://www.efficios.com](http://www.efficios.com)
- Author/Maintainer of
  - LTTng, LTTV, Userspace RCU
- Ph.D. in computer engineering
  - Low-Impact Operating System Tracing
Plan

- LTTng Installation
- Tracing Strategy
- Trace Analysis (Hands-on Example)
- Questions
> LTTng installation

- http://lttng.org
  - LTTng
  - LTT control
  - LTTV

- Documentation
  - LTTng Kernel Tracer Manual
  - LTTng Compatibility List
Lockless Trace Clock

- `dmesg`, check for LTT warnings
- Cycle counter is used
- For architectures with non-synchronized cycle counters (e.g. some x86):
  - `cpufreq-set -g performance`
  - `idle=poll` (kernel parameter)
- LTTng fully supports power management and frequency scaling on ARM OMAP3
  - funded by Nokia
> Tracing Strategy

- Problem Identification
- Trace Session Setup
- Anchor / Trigger
> Problem identification

- Bug report summary
  - What is going wrong with the system?
  - What is the system configuration affected?
    - Hardware
    - Software
  - Optionally: known good / known bad configurations
Tracing strategy (decision factors)

- Reproducible on development setup or only in production?
- Tracing overhead the system can tolerate
- Frequency of problem occurrence
- Availability of the system
  - Remote/local
  - Controlled by third-party
- Number of tracing iterations available
> Trace Session Setup

- Identify the tracer setup best suited to solve the problem
  - Producer-consumer tracing
  - Flight recorder tracing
  - Per-channel buffer size
  - Activated instrumentation
> Anchor / Trigger

- Traces are hard to analyze
  - Large volume of information collected
  - Hard to identify relevant information
- Add anchor instrumentation to the system
- Use triggers to stop flight recorder tracing
> Anchor

- Starting point for trace analysis
- Identify surrounding of problem occurrence
- Different types
  - Instrumentation anchors
  - Analysis-generated anchors
> Anchor

- **Instrumentation anchors**
  - Userland or kernel instrumentation
    - UST (Userspace Tracer)
    - Write to /debugfs/ltt/write_event
    - Add kernel TRACE_EVENT/markers
  - Events generated from user interaction
    - Input subsystem
  - Instrumentation of program error-handling

- **Analysis-generated anchors**
  - e.g. longest timer interrupt jitter
Producer-consumer tracing

- Writes trace data to the file system
- Whole trace session duration
- Initial state dump: complete state collected
  - (+) Very accurate state representation
  - (-) Consumes disk or network I/O bandwidth

(as root)

ltt-armall

lttctl -C -w /tmp/trace-prod1 trace-prod1

... 

lttctl -D trace-prod1
> Flight recorder tracing

- Gather trace data in circular ring buffers
- Kept in memory, oldest data overwritten
- Last events available when tracing is stopped
- Per-channel size is configurable
  - (+) Very low system throughput overhead
  - (-) Shorter available backlog
  - (-) System state is less accurate
    - Partially unknown
> Triggers

- Instrumentation with side-effect
  - Start/stop tracing when executed
- Particularly useful for flight recorder mode
  - Produces event backlog that lead to execution of “trigger”
- Kernel API
  - \texttt{ltt\_trace\_start(“name”), ltt\_trace\_stop(“name”)}
- From user-space
  - \texttt{lttctl -s name ; lttctl -p name}
Userland trigger example

Userland trigger for flight recorder trace

(as root)
ltt-armall
lttctl -c -o channel.all.overwrite=1 -w /tmp/trace-utrigger1 trace-utrigger1
lttctl -s trace-utrigger1

...(trigger) → lttctl -p trace-utrigger1
lttctl -d -w /tmp/trace-utrigger1 trace-utrigger1
Kernel trigger example

Kernel trigger

(as root)
ltt-armall
lttctl -c -o channel.all.overwrite=1 -w /tmp/trace-ktrigger1 trace-ktrigger1
lttctl -s trace-ltrigger1
...
(trigger in kernel) → ltt_trace_stop(“trace-ktrigger1”);

lttctl -d -w /tmp/trace-ktrigger1 trace-ktrigger1
Trace Analysis (Hands-on Example)

- Identify sources of audio latency
- Scheduler latency
  - `wakeup-latency.c`
    - `write_event` anchor
- With 2.6.33.2 kernel

  maximum latency: 44614.1 µs
  average latency: 3851.4 µs
  missed timer events: 0
Find anchors

Filter expression:

- `<field> <comparator> <value>`
- `<field> <comparator> <value> <logical> <expression>`
  (comparators: =, !=, <, <=, >, >=)
  (logical: &, |, !, ^)

(see lttv-gui "filter" plugin for expression examples)

Find all events in channel "userspace"
(written through write_event):
% lttv -m textDump -e "channel.name=userspace" \
- t /tmp/trace-prod1

Filter by channel.event (list with "find /debugfs/ltt/markers"):
% lttv -m textDump -e "event.name=kernel.sched_schedule" \
- t /tmp/trace-prod1

Filter by PID:
% lttv -m textDump -e "state.pid=1" - t /tmp/trace-prod1

Filter fields

- event.name
- event.subname
- event.category
- event.time
- event.tsc
- event.target_pid
- channel.name
- trace.name
- state.process_name
- state.thread_brand
- state.pid
- state.ppid
- state.creation_time
- state.insertion_time
- state.execution_mode
- state.execution_submode
- state.process_status
- state.cpu
% lttv -m textDump -e "channel.name=userspace" -t /tmp/trace-cfs-2/

... userspace.event: 234.334734395 (/tmp/trace-cfs-2/userspace_0), 10905, 10516, ./wakeup-latency, , 10517, 0x0, SYSCALL { string = "late by: 44614.1 µs" }

...
> Viewing execution patterns
Instrumentation example (with markers):

Index: linux-2.6-lttng.git/kernel/sched_fair.c

--- linux-2.6-lttng.git.orig/kernel/sched_fair.c 2010-04-10
11:47:04.000000000 -0400
+++ linux-2.6-lttng.git/kernel/sched_fair.c 2010-04-10
11:50:59.000000000 -0400

....
@@ -764,12 +768,18 @@
     thresh >>= 1;
     vruntime -= thresh;
+    trace_mark(test, cfs_place_sleeper,
+                "pid %d thresh %lu vruntime %llu",
+                task_of(se)->pid, thresh, vruntime);

    }

....
> Digging into Xorg scheduling
> Looking at alsa

- Adding instrumentation of buffer underrun into aplay as anchor
  - `write_event`

```
./aplay --period-size=128 --buffer-size=512 -D hw:0 test.wav
```

Buffer:
512 samples / 48000 samples/s = 10.6ms

Period:
128 samples / 48000 samples/s = 2.6ms
> View of a buffer underrun

12.8ms