# Linux Foundation Collaboration Summit 2009

## LTTng, Filling the Gap Between Kernel Instrumentation and a Widely Usable Kernel Tracer

# > Plan

- Presenter
- Tracing Infrastructure in Mainline Kernel
- LTTng motivation
- Work done since Kernel Summit and Plumbers Conference
- Conclusion

# > Presenter

- Mathieu Desnoyers
- Author/Maintainer of LTTng and LTTV
- Ph.D. Candidate at École Polytechnique de Montréal
- Fields of interest
  - Tracing
  - Reentrancy, Synchronization, Locking Primitives
  - Multi-core, Real-time

# > Tracing Infrastructure in Mainline Kernel

- Kernel Markers
  - Debug-style event description
    - trace_mark(sched_schedule, “prev %d next %d”, prev->pid, next->pid);
  - Tracer event description (LTTng tree)
    - Exports the markers through debugfs markers subdirectory
    - Connects callbacks to tracepoints automatically

# > Tracing Infrastructure in Mainline Kernel

```
void probe_sched_switch(struct rq *rq, struct task_struct *prev,
		struct task_struct *next);

DEFINE_MARKER_TP(kernel, sched_schedule, sched_switch, probe_sched_switch,
	"prev_pid %d next_pid %d prev_state #2d%ld");

notrace void probe_sched_switch(struct rq *rq, struct task_struct *prev,
		struct task_struct *next)
{
	struct marker *marker;
	struct serialize_int_int_short data;

	data.f1 = prev->pid;
	data.f2 = next->pid;
	data.f3 = prev->state;

	marker = &GET_MARKER(kernel, sched_schedule);
	ltt_specialized_trace(marker, marker->single.probe_private,
		&data, serialize_sizeof(data), sizeof(int));
}
```

# > Tracing Infrastructure in Mainline Kernel

- Tracepoints
  - Infrastructure to provide managed set of kernel events
    - include/trace/sched.h

```
DECLARE_TRACE(sched_switch,
        TPPROTO(struct rq *rq, struct task_struct *prev,
                struct task_struct *next),
                TPARGS(rq, prev, next));
```

    - kernel/sched.c
      - trace_sched_switch(rq, prev, next);

# > LTTng motivation

- Application, library and kernel system-wide performance analysis and debugging
- Heavy HPC multi-core application workloads
- Fit within embedded systems resources limitations
- Run continuously on production systems (flight recorder mode) to provide meaningful bug reports
- Primary target : developers, end-user support

# > LTTng key features

- Very good re-entrancy
  - Supports kernel-wide instrumentation
- Solid monotonic time-base
- Low-overhead
- Architecture agnostic core
- Extensible instrumentation
- Multiple tracing sessions support

# > Users and contributors

- Google, IBM, Ericsson, Fujitsu, Siemens, Nokia, Autodesk, Sony, Montavista, Samsung, Boeing
- Distributions
  - SuSe real-time (Novell)
  - WindRiver Workbench 2.6
  - Montavista Carrier Grade Linux 5.0

# > Work Done since KS2008 and LPC

- Event grouping / ID management
- Event header rework
- Removed “Heartbeat timer”
- Kernel Markers as data source
- Pluggable memory back-ends
- Splice
- DebugFS interface
- Layered buffering system

# > Event grouping / ID management

- Group events under “channels”
  - One channel per tracer
  - Each channel has its own per-CPU buffers
- Allocate event IDs dynamically within the group
  - Allows very compact trace event headers
  - 5 bits typically used for event ID
- Event ID allocation and channel management added to the Linux Kernel Markers.

# > Event Header Rework

- 27-bits for cycle counter
- 5-bits for event ID
  - Ids 29, 30, 31 reserved for “extended headers”
    - 29 : size and timestamp counter
    - 30 : id and size
    - 31 : id
- Optional “payload size” for tracer debugging as extension

# > Removed “Heartbeat timer”

- Assuming a 64-bits time source
  - see trace clock 32 to 64
- Detect 27-bits overflows since the previous event in the current buffer in the tracing site by saving the counter read in a local structure
  - Must carefully consider non-atomic writes on 32-bits architectures. Insures no overflow will be missed, but can generate duplicated extended header.

# > Kernel Markers as data source

- All events meant to be saved in any channel have a description part of the marker section
- All events can be saved either in tracer-specific channel, or used for system-wide tracing
- Events declared are presented to user-space through a debugfs interface and can be enabled individually.

# > Pluggable memory back-ends

- Stop using vmap() to save TLB entries
- Created an API to permit sequential write into an array of page pointers.
  - No event size limitation
  - Space reservation layer does not have to care about memory back-end used
- Allows to be built with a different (potentially contiguous) back-end.
  - Supports, e.g., writing to video card memory (survives hot reboots). Useful for crash dump.

# > Splice()

- Zero-copy from the kernel to the block device or network.
- Does not require extra TLB entries like the vmap() approach.
- Extended NFS to provide splice() write support.

# > Debugfs interface

- /mnt/debugfs/ltt
  - markers
  - setup_trace, destroy_trace, control
  - kprobes

# > Layered buffering system

- Event ID management
- Space reservation
  - Lockless
  - IRQ off
  - IRQ off + spinlock
- Memory backend
  - Allocation
  - write(), read()
  - API shows memory as contiguous

# > Text output (“cat” support)

- Most of the infrastructure present
- Specialized tracers can hook on the buffers through internal API and use the low-level read primitives to print the data following their ascii-art inspiration
- Will provide /mnt/debugfs/ltt/<trace>/ascii
- One single last patch should be reworked to perform integration with the ring buffer. See ltt-ascii.c in the LTTng tree.

# > Performance Considerations

- Optimization phase
  - Turn “pluggability” into a build-time feature
    - Remove costly function calls from the fast path !!
  - Create ltt-type-serializer for custom probes
    - C structures directly written into the buffer, all sizes known statically.
  - Inline all tracer fast-paths, build-time modularization
- Result : worse-case nightmare-ish scenario
  - Localhost tbench 8-cores with tracing enabled
    - 18.2 % slowdown (but typical under 5 %)

# > Conclusion (1)

- Two tracers are not competition if they target different user bases
  - LTTng : targets end-user / developer / tech support
  - Ftrace : targets kernel developer
- Main difference comes from different use-cases and requirements (see motivations)
- Sharing low-level transport infrastructure is not possible if requirements from one party are not considered

# > Conclusion (2)

- Mainlining ?
  - Core kernel code is jealously protected due to large impact of all subsystems
    - Scheduler
    - Kernel instrumentation
      - Kprobes, Markers, Tracepoints, Function Tracer
  - Non-core kernel code “should” easily get to mainline
    - Drivers
    - Tracer infrastructure (trace control, buffering...)

# > Conclusion (3)

- Tracer control and transport are ***not core kernel*** code. Why hasn't it been merged yet ?
  - LTTng is mature
    - Follows K42 and LTT development
    - Started more than 4 years ago
  - LTTng has a large user-base
    - Google, IBM, Ericsson, Fujitsu, Siemens, Nokia, Autodesk, Sony, Montavista, Samsung, Boeing
    - Included in SuSe, Montavista, WindRiver distributions
    - Main complaint : must recompile their kernel

# > Conclusion (4)

- Why ? One ring-buffer to rule them all ?

# > Questions ?

- Information
  - http://www.lttng.org/
  - ltt-dev@lists.casi.polymtl.ca