LinuxCon 2010
Tracing Mini-Summit

A new unified Lockless Ring Buffer library for efficient kernel tracing

Presentation at:
http://www.efficios.com/linuxcon2010-tracingsummit

E-mail:
mathieu.desnoyers@efficios.com

> Presenter

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

> Plan

- History
- Mandate
- Genericity and Flexibility
- Speed and Compactness
- Reliability
- Working together

> History

- May 2005: LTTng implements its ring buffer from scratch
  - Learns lessons from K42, RelayFS and LTT.
- October 2005: LTTng becomes lock-less
  - LTTng gets increasingly used by the industry and shipped with many embedded and RT Linux distributions since then.
- 2008: Ftrace (lock-less in 2009)
- 2010: Perf

> Mandate

- Wish from Linus expressed at the Kernel Summit 2008 to have a common tracer infrastructure in the kernel
- Asked by Steven Rostedt to come up with a unified solution

> Generic Ring Buffer Library

- Input
  - Data received as parameter from ring buffer library clients
- Output
  - Data available through a global or per-CPU file descriptor with splice, mmap or read.
  - Or data available internally to the ring buffer client for reading

> Generic Ring Buffer Library

- Derived from the LTTng ring buffer
  - Exists since 2005
- Goals
  - Generic and flexible
  - Clean API
  - Fast and compact
  - Reliable

> Genericity and Flexibility

- Target Perf, Ftrace, LTTng and drivers
- Not only tracer-specific
  - Ring buffer sits in /lib
- Achieve genericity without hurting performance
  - Ring buffer clients
  - Instantiate client-specific configurations
  - Express configuration into a constant client structure passed as parameter to inline functions

> API: pre-cooked (simple) APIs

- Create/destroy a channel
  - Global buffer
  - Per-CPU buffers
- In-kernel write()
- Read a file descriptor
  - Global iterator
    - The library does fusion merge of per-CPU buffer events based on a heap and quiescent states
  - Per-CPU iterator

> API: pre-cooked APIs

- Mode
  - Overwrite
  - Discard
- Channels
  - Global
  - Per-CPU
    - Global iterators
    - Per-CPU iterators

> Advanced API

- Client configuration
- Client-provided callbacks

> Configuration

- Buffers per-CPU or global
- Overwrite or discard mode
- Natural or packed alignment
- Output
  - splice(), mmap(), read(), iterator, client-specific
- Memory allocation backend
  - page, vmap, static
- OOPS consistency, IPI barrier, wakeup

> Client-provided callbacks

- Clock read
- Event and sub-buffer header size
- Sub-buffer begin/end
- Buffer create/finalize
- Record get
  - For iterators

> Speed and Compactness

- Fast paths
  - Constant configuration structure
  - Compiler removes unused code
- Slow paths
  - Configuration dynamically tested
  - Same code shared amongst all clients

> Performance

- Throughput
- Scalability

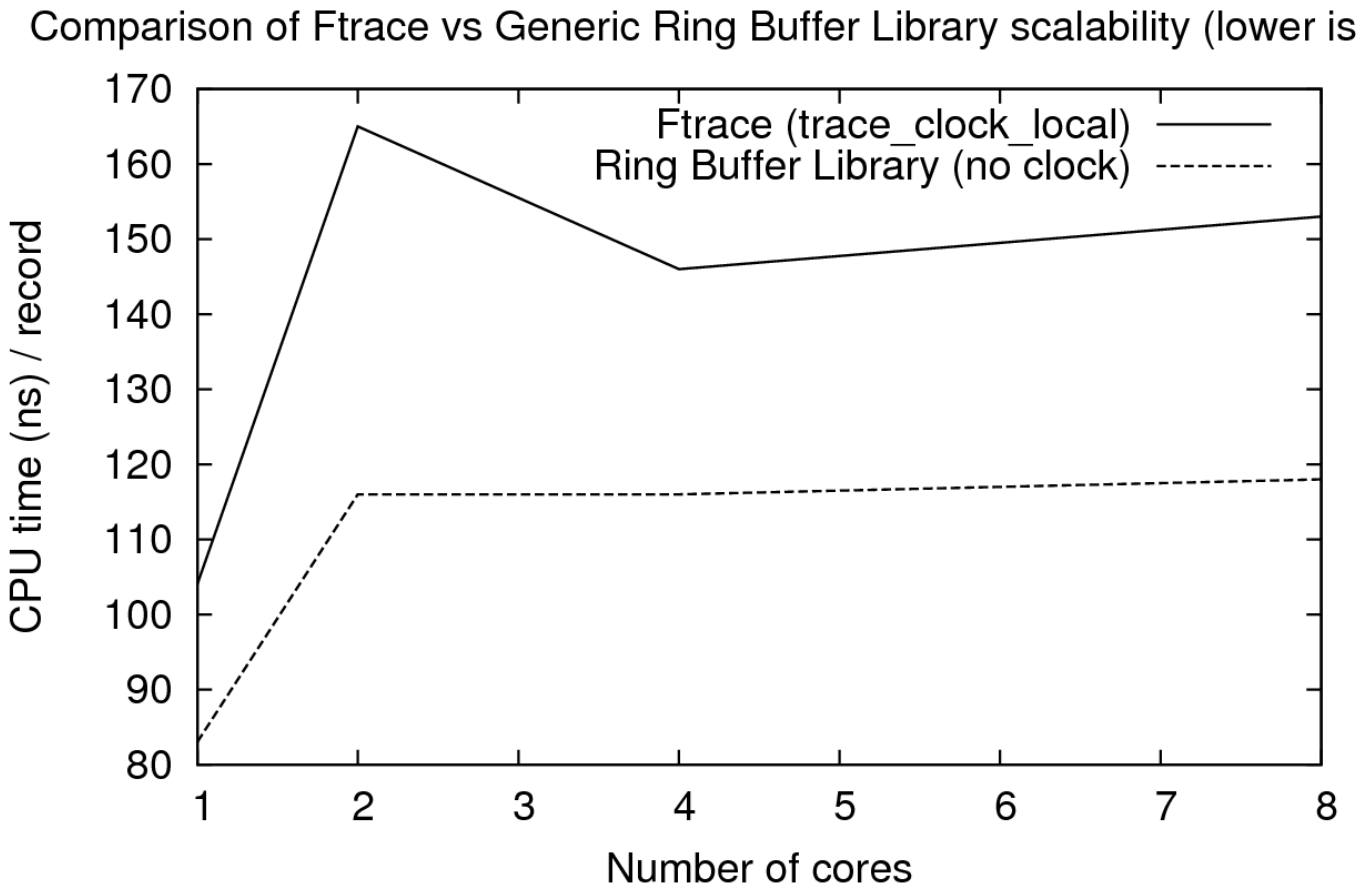
> Throughput (overwrite mode)

- Generic Ring Buffer Library
  - 83-199 ns/entry (depending on configuration)
- Ftrace
  - 103-187 ns/entry
- Perf
  - Mode unavailable

> Throughput (discard mode)

- Generic Ring Buffer Library
  - 257 ns/entry written
- Perf
  - 423 ns/entry written
    - (approximation from Perf output)
- Getting accurate results is hard, influenced by discarded events

> Scalability



> Reliability

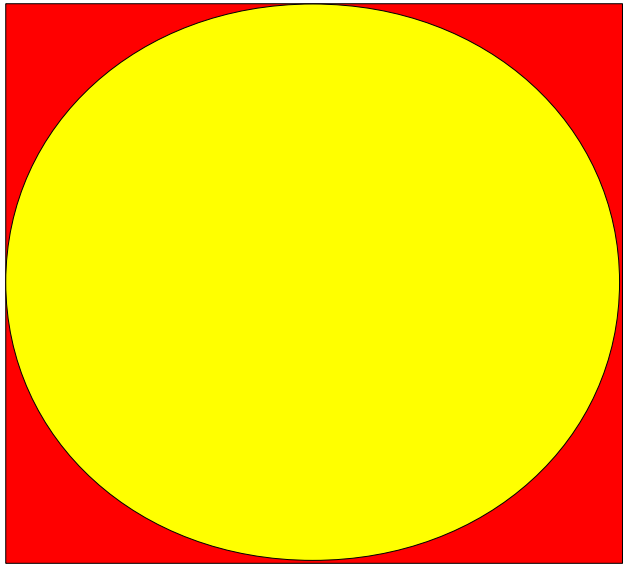
- LTTng
  - Formal verification of the ring buffer algorithm at the architecture level (modeling execution on superscalar processors)
  - Testing on large user-base

> Working together

- Ever had the feeling you were trying to fit something square-shaped into a circle ?

> Working together

- Need to polish off the rough spots



> Working together

- Trying to come up with a clean and flexible API
- Nevertheless, does not always map the current Ftrace and Perf APIs
- Trying very hard not to bloat the API

> Working with Ftrace

- Steven has been very helpful
- I'm about 80% done working on Ftrace transition to the generic ring buffer library

> Ftrace odd-fitting pieces

- Ftrace iteration code
  - Huge set of API functions for iterating on stopped trace buffers without consuming data.
  - Used for:
    - Dumping same output with "cat" many times
    - Peek next item to place brackets in function graph tracer output
  - Could be replaced by "rewind" ability and by modifying the function graph tracer plugin

> Perf

- mmap()-based ABI between kernel and user-space for consuming data.
- No kernel callback invoked when the consumer finishes reading data.
  - Severely limits design choices
- Does not support (and developers don't consider as valid use-case) reading data while writing into a buffer in flight recorder mode.

> Perf

- Does not use padding between sub-buffers
  - No concept of sub-buffers
  - All events are physically contiguous
- Cannot create efficient chunks of data for splice() without copy
- Cannot efficiently index trace without reading all events (increases delay before a large trace can be analyzed)
- Basic data encapsulation principles

> Perf

- Why do they hate sub-buffers so much ?
  - Claim of simplicity
    - False. The fast path ends up being both larger and slower than the generic ring buffer.
- Why is this important ?
  - Shows how low-level Perf design choices prevent contributors from fulfilling end-user basic use-cases.
  - Shows Perf developers unwillingness to support use-cases other than kernel developers own needs.

> Funding

- Thanks to Ericsson for funding parts of this work.

> Questions ?

EfficiOS

- http://www.efficios.com

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

> API (per-CPU discard)

```
extern struct channel *
ring_buffer_percpu_discard_create(size_t buf_size);


extern void
ring_buffer_percpu_discard_destroy(struct channel *chan);


extern int
ring_buffer_percpu_discard_write(struct channel *chan,
                                 const void *src,
                                 size_t len);
```

And map file operation "channel_payload_file_operations" from iterator.h to file descriptor.