Interoperability Between Tracing Tools with the Common Trace Format (CTF)

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

- Mathieu Desnoyers
- EfficiOS Inc.
  - http://www.efficios.com
- Author/Maintainer of
  - LTTng, LTTng-UST, Babeltrace, Userspace RCU
> Content

- Common Trace Format introduction & goals
- Trace Stream Description Language
- Overview of trace layout
- Collaboration
- Reference implementations
- Other tools based on CTF
- Areas to improve
- Conclusion
> Common Trace Format

- Targets system-wide and multi-system trace representation in a common format, for integrated analysis:
  - **Software traces**
    - Across multiple CPUs
    - Across the software stack (Hypervisor, kernel, library, applications)
  - **Hardware traces**
    - DSPs, device-specific tracing components.
    - GPUs.
Goals of the Common Trace Format (CTF)

- Portable,
- Compact,
- Configurable per-architecture to express layout required for speed,
- Transport independent: disk, network, serial port, memory,
- Usable on minimalistic DSPs as well as full-featured OS,
- Availability of flight recorder,
Goals of the Common Trace Format (CTF) (continued)

- Buffers retrievable after crash,
- Support dynamically inserted instrumentation while tracing,
- Support per-cpu buffers, and many configurable streams.
What is CTF?

- Self-described binary trace format
- Domain-specific language (DSL) for description of stream layout: TSDL (Trace Stream Description Language)
- Trace embeds its own description
• TSDL trace description entry:

```c
trace {
    major = 1; minor = 8; uuid = "a116db0a-ad45-40a0-9f66-b195d79432a0";
    byte_order = le;
    packet_header := struct {
        uint32_t magic; uint8_t uuid[16]; uint32_t stream_id;
    };
};
```
> TSDL Clock Description

• TSDL clock description entry:

```c
clock {
    name = monotonic;
    uuid = "1fece6ff-a288-4a59-b750-07bef0d296f0";
    description = "Monotonic Clock";
    freq = 1000000000; /* Frequency, in Hz */
    /* clock value offset from Epoch is: offset * (1/freq) */
    offset = 1338755739325858212;
}
```

```c
typealias integer {
    size = 64; align = 8; signed = false;
    map = clock.monotonic.value;
} := uint64_clock_monotonic_t;
```
TSDL Types

- TSDL type descriptions:

```c
typedef integer { size = 64; align = 8; signed = false; } := uint64_t;
[...]
typedef integer { size = 27; align = 1; signed = false; } := uint27_t;
struct packet_context {
    uint64_clock_monotonic_t timestamp_begin;
    uint64_clock_monotonic_t timestamp_end;
    uint32_t events_discarded; uint32_t content_size; uint32_t packet_size;
    uint32_t cpu_id;
};
struct event_header {
    uint64_t timestamp;
    uint32_t id;
} align(8);
```
TSDL Stream and Event

• TSDL stream and event descriptions:

```
stream {
    id = 0;
    event.header := struct event_header;
    packet.context := struct packet_context;
};

event {
    name = "ust_tests_hello:tptest"; id = 0; stream_id = 0; loglevel = 13;
    fields := struct { uint27_t _intfield; [...] };
};
```
> CTF Diagram: Field Types

- Integer
- Array
- Structure
- String

- Enumeration
- Float
- Sequence

- Variant

Depends on
CTF Diagram: Event Structure Example

- **Event**
  - **Header**
    - id
    - timestamp
  - **Context**
    - vpid
    - procname
  - **Payload**
    - field0
    - field1
CTF Diagram: Trace Structure

<table>
<thead>
<tr>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedefias ...;</code></td>
</tr>
<tr>
<td><code>trace { ... };</code></td>
</tr>
<tr>
<td><code>clock { ... };</code></td>
</tr>
<tr>
<td><code>stream { ... };</code></td>
</tr>
<tr>
<td><code>event { ... };</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stream 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Packet 0</strong></td>
</tr>
<tr>
<td>header</td>
</tr>
<tr>
<td>context</td>
</tr>
<tr>
<td>event 0</td>
</tr>
<tr>
<td>event 1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stream 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Packet 0</strong></td>
</tr>
<tr>
<td>header</td>
</tr>
<tr>
<td>context</td>
</tr>
<tr>
<td>event 0</td>
</tr>
<tr>
<td>event 1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stream 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Packet 1</strong></td>
</tr>
<tr>
<td>header</td>
</tr>
<tr>
<td>context</td>
</tr>
<tr>
<td>event 0</td>
</tr>
<tr>
<td>event 1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stream 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Packet 1</strong></td>
</tr>
<tr>
<td>header</td>
</tr>
<tr>
<td>context</td>
</tr>
<tr>
<td>event 0</td>
</tr>
<tr>
<td>event 1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

...
> CTF Diagram: Trace Structure

Trace directory hierarchy:

```
/  
metadata
   
stream_0
   
stream_1
   ...
```
CTF Diagram: Trace Collection

Trace collection directory hierarchy:

/  TraceA
   /  TraceB
      /  ...

metadata
  stream_0
  stream_1
  ...
metadata
  stream_0
  stream_1
  ...

Check if clock UUID match for trace correlation.
> Advanced Usage: Variant Type

```c
struct event_header_compact {
  enum : uint5_t { compact = 0 ... 30, extended = 31 } id;
  variant <id> {
    struct {
      uint27_clock_monotonic_t timestamp;
    } compact;
    struct {
      uint32_t id;
      uint64_clock_monotonic_t timestamp;
    } extended;
  } v;
} align(8);
```
5-bit: values 0-30 select “compact” variant.

5-bit: value 31 selects “extended” variant.

3-bit padding: on this architecture, 32-bit and 64-bit integers are aligned on 8-bit.

<table>
<thead>
<tr>
<th>0 ... 7 8 ... 1516...2324...31</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>vpid (32-bit)</td>
</tr>
<tr>
<td>event fields...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 ... 7 8 ... 1516...2324...31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td>timestamp (64-bit)</td>
</tr>
<tr>
<td>vpid (32-bit)</td>
</tr>
<tr>
<td>event fields...</td>
</tr>
</tbody>
</table>
> Environment Description

```bash
env {
    hostname = "thinkos";
    domain = "kernel";
    sysname = "Linux";
    kernel_release = "3.4-trunk-amd64";
    kernel_version = "#1 SMP Tue Jun 26 17:23:03 UTC 2012";
    tracer_name = "lttng-modules";
    tracer_major = 2;
    tracer_minor = 0;
    tracer_patchlevel = 1;
};
```
> Collaboration

- Trace format specification
  - Funded by
    - Linux Foundation CE Linux Forum and Ericsson
  - In collaboration with Multi-Core Association Tool Infrastructure Workgroup
    - Freescale, Mentor Graphics, IBM, IMEC, National Instruments, Nokia Siemens Networks, Samsung, Texas Instruments, Tilera, Wind River, University of Houston, Polytechnique Montréal, University of Utah.
  - Gathered feedback from Linux kernel developers and SystemTAP communities.
> Reference Implementations

• Babeltrace
  • Reference implementation trace conversion tool and read/seek API for trace collections.
  • Initially converts
    – From CTF to text
    – From dmesg text log to CTF

• LTTng kernel 2.0 and LTTng-UST 2.0
  • Native CTF producer reference implementation.

• Eclipse Tracing and Monitoring Framework
Other tools based on CTF

- GDB (coming in Q4 2012)
- Javeltrace (CTF generator)
- Proprietary converters (derived from Babeltrace)
- LTTngTop
- LTTTV
- LTTng Studio
Areas to Improve

- Support for clocks with varying frequency,
- Mandate some of the currently “suggested” fields,
- Extend CTF to include state change description along with events,
- Extend CTF to include categorization of events,
- Should we keep CTF minimalistic (limited to description of binary layout and clocks), or include high-level semantic information?
Questions?

- CTF specification available at:
  http://www.efficios.com/ctf

- LTTng Information
  - http://lttng.org
  - lttng-dev@lists.lttng.org