LTTng 2.0 : Tracing, Analysis and Views for Performance and Debugging.

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

- Tracing benefits,
- LTTng 2.0 Linux kernel and user-space tracers,
- LTTng 2.0 usage scenarios & viewers,
- New features ready for LTTng 2.1,
- Conclusion
> Benefits of low-impact tracing in a multi-core world

- Understanding interaction between
  - Kernel
  - Libraries
  - Applications
  - Virtual Machines
- Debugging
- Performance tuning
- Monitoring
> Tracing use-cases

- Telecom
  - Operator, engineer tracing systems concurrently with different instrumentation sets.
  - In development and production phases.
- High-availability, high-throughput servers
  - Development and production: ensure high performance, low-latency in production.
- Embedded
  - System development and production stages.
> LTTng 2.0

- Rich ecosystem of projects,
- Key characteristics of LTTng 2.0:
  - Small impact on the traced system, fast, user-oriented features.
- Interfacing with: Common Trace Format (CTF)

Interoperability Between Tracing Tools with the Common Trace Format (CTF),
Mathieu Desnoyers, EfficiOS,
Tracing Summit,
Aug. 30, Room Nautilus 3, 14h45.

Tracing Well With Others: Integration of GDB Tracepoints Into Trace Tools,
Stan Shebs, Mentor Graphics,
Tracing Summit,
Aug. 30, Room Nautilus 3, 11h15.
LTTng 2.0 Tracers

- Controlled by lttng-sessiond(8),
- User interact with single command line UI, “lttng(1),
- LTTng modules: Linux kernel tracing,
- LTTng-UST library: user-space tracing,
LTTng 2.0 Low-Overhead Tracing Architecture

**LTTng Command Line Interface (GPLv2)**
- liblttngctl (LGPLv2.1)

**LTTng Session Daemon (GPLv2)**
- Control multiple tracing sessions
- Centralized tracing status management
  - liburcu (LGPLv2.1)
  - liblttngctl (LGPLv2.1)
  - liblttng-ust-ctl (GPLv2)

**Custom Control Software**
- Interface with proprietary cluster management infrastructures
  - liblttngctl (LGPLv2.1)

**LTTng Relay Daemon (GPLv2)**

**Local storage CTF†**

**CTF† over TCP/UDP/SSH**

**LTTng Consumer Daemon (GPLv2)**
- Zero-copy data transport or aggregator
- Export raw trace data, statistics and summary data
- Snapshots from in-memory flight recorder mode
- Store all trace data, discard on overrun
  - liburcu (LGPLv2.1)
  - liblttng-ust-ctl (GPLv2)
  - liblttng-consumer (GPLv2)

**C/C++ Application**
- Tracepoint*
- Tracepoint Probes*
  - liburcu (LGPLv2.1)
  - liblttng-ust (LGPLv2.1)

**Java/Erlang Application**
- Tracepoint*
  - LTTng VM adaptor
  - Tracepoint Probes*
  - liburcu (LGPLv2.1)
  - liblttng-ust (LGPLv2.1)

**Linux kernel**
- Tracepoint*
- Dynamic probes (kprobes)

**LTTng modules (GPLv2/LGPLv2.1)**
- Tracepoint Probes*
  - liburcu (LGPLv2.1)
  - liblttng-ust (LGPLv2.1)

**LTTng VM adaptor**
- Tracepoint*

**Java/Erlang Application**
- Tracepoint

**Instrumentation**

**Control**

**Trace Data**

**Libraries**

---

### Instrumentation
- Low overhead, no trap, no system call,
- Re-entrant: Signal, thread and NMI-safe,
- Wait-free read-copy update,
- Can be used in real-time systems,
- Use GCC asm goto and Linux kernel static jumps,
- Cycle-level time-stamp,
- Runtime activation of statically and dynamically inserted instrumentation,
- Non-blocking atomic operations,
- Allow tracing of proprietary applications and proprietary control software (LGPLv2.1 license).

### Control
- Signal, thread, and NMI-safe

### Trace Data
- Compact binary format,
- Self-described,
- Handles HW&SW tracing,
- TCP and UDP network streaming,
- Flexible data layouts for expressiveness and highest throughput,
- Layout allows fast seek and processing of very large traces (> 10GB).

### Libraries
- lib廖ngctl (LGPLv2.1)
LTTng 2.0 Common Trace Format Viewers

Host-Side User Interfaces

- **Babeltrace** (MIT/BSD)
  - Trace converter
  - Trace pretty printer
  - Allow open source and proprietary plugins
  - libbabeltrace (MIT/BSD)

- **LTTngTop** (GPLv2)
  - Top-alike resource usage view
  - Read live from buffers
  - libbabeltrace (MIT/BSD)

- **Eclipse Tracing and Monitoring Framework** (EPL)
  - Trace display and analysis
  - Trace control
  - Allow open source and proprietary plugins

Local storage

- **CTF†**

Memory-mapped buffers or splice, poll, ioctl
Trace Control Libraries and Bindings

- liblttng-ctl and lttng/lttng.h expose a C/C++ LGPL v2.1 API to control tracing.

- LTTng tracing control Python bindings planned to be merged into lttng-tools 2.2

- Noteworthy: GDB is also using Python.
LTTng 2.0 Linux Kernel Tracer

- Build against a vanilla or distribution kernel 2.6.38+, without need for additional patches, and back to 2.6.32 with backport of 3 upstream kernel patches,
- Instrumentation sources: Tracepoints, System calls, Function tracer, kprobes, and kretprobes,
- Supports multiple concurrent tracing sessions,
- Flight recorder mode, snapshots, supported at the tracer level, not supported by lttng-tools 2.0 yet (planned for 2.2).
LTTng 2.0 Kernel Tracer

- Supports dynamically selectable “context” information to augment event payload
  - Any Perf Performance Monitoring Unit counter
  - PID, PPID, TID, process name, VPID, VTID, …
  - Dynamic Priority, nice value
> LTTng 2.0 User-Space Tracer

- Supports: Linux, FreeBSD, OpenBSD, NetBSD,
- Tracing performed directly in user-space through shared memory map, without calling the kernel (for speed),
- Support TRACEPOINT_EVENT instrumentation: dynamically enabled, statically defined, user-space instrumentation.
- Supports multiple concurrent tracing sessions.
LTTng-UST 2.0
User-space Tracer Features

● TRACEPOINT_EVENT() API for application/library static instrumentation with sdt.h gdb/systemtap integration.

● Per-user tracing.

● System-wide tracing.
  • “tracing” group: no need to be root to perform system-wide tracing.
In header:

TRACEPOINT_EVENT(ust_tests_hello, tptest,
    TP_ARGS(int, anint, long *, values,
        char *, text, size_t, textlen,
        double, doublearg, float, floatarg),
    TP_FIELDS(
        ctf_integer(int, intfield, anint)
        ctf_integer_hex(int, intfield2, anint)
        ctf_array(long, arrfield1, values, 3)
        ctf_sequence(char, seqfield1, text,
            size_t, textlen)
        ctf_string(stringfield, text)
        ctf_float(float, floatfield, floatarg)
        ctf_float(double, doublefield, doublearg)
    )
)
> User-level Tracepoint

Name convention

< [com_company_]project[_component] >, < event >

Where "company" is the name of the company,
   "project" is the name of the project,
   "component" is the name of the project component (which may
   include several levels of sub-components, e.g. ...
   ...component_subcomponent_...)) where the tracepoint is located
   (optional),
   "event" is the name of the tracepoint event.

Tracepoint invocation within the code:

```c
void fct(void)
{
    tracepoint(ust_tests_hello, tptest, i, values,
                text, strlen(text), dbl, flt);
}
```
> Usage Scenarios

- Tracing,
- Analyzing trace data,
- Tracing across kernel and user-space,
- Tracing across multiple nodes.
> Tracing: record kernel trace

```
$ lttng create
Session auto-20120827-120832 created.
Traces will be written in /home/test4/lttng-traces/auto-20120827-120832
$ lttng enable-event -k -a
All kernel events are enabled in channel channel0
$ lttng start
Tracing started for session auto-20120827-120832
$ lttng stop
Tracing stopped for session auto-20120827-120832
```

> Tracing: view trace

test4@thinkos:$ lttng view |head
[12:09:27.319016156] (+.???????????) thinkos exit_syscall: { cpu_id = 0 }, { ret = 4400 }
[12:09:27.319019724] (+0.000003568) thinkos sys_poll: { cpu_id = 0 }, { ufds = 0x7F799E791CB0, nfds = 2, timeout_msecs = -1 }
[12:09:27.319021368] (+0.000001644) thinkos exit_syscall: { cpu_id = 0 }, { ret = 1 }
[12:09:27.319022644] (+0.000001276) thinkos sys_recvmsg: { cpu_id = 0 }, { fd = 18, msg = 0x7F799E7909C0, flags = 0 }
[12:09:27.319027091] (+0.000004447) thinkos exit_syscall: { cpu_id = 0 }, { ret = 1 }
[12:09:27.319037949] (+0.000010858) thinkos sys_geteuid: { cpu_id = 0 }, { }
[12:09:27.319038734] (+0.000000785) thinkos exit_syscall: { cpu_id = 0 }, { ret = 0 }
[12:09:27.319040053] (+0.000001319) thinkos sys_pipe: { cpu_id = 0 }, { fildes = 0x7F799E790920 }
[12:09:27.319046308] (+0.000006255) thinkos exit_syscall: { cpu_id = 0 }, { ret = 0 }
[12:09:27.319047858] (+0.000001550) thinkos sys_mmap: { cpu_id = 0 }, { addr = 0x0, len = 10485760, prot = 3, flags = 131362, fd = -1, offset = 0 }
test4@thinkos:$
> LTTng 2.0 high-speed “strace”

lttng enable-event --syscall -a

```
compudj@squeeze-amd64:~
```

```bash
name = sys_brk, stream.packet.context = { cpu_id = 1 }, event.fields = { brk = 28622848 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 28622848 }
name = sys_read, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3, buf = 0x1B48008, count = 9645 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 9645 }
name = sys_close, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = sys_open, stream.packet.context = { cpu_id = 1 }, event.fields = { filename = "/root/.bash_history", flags = 513, mode = 0600 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 3 }
name = sys_write, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3, buf = 0x1B48081, count = 9524 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 9524 }
name = sys_close, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = sys_rt_sigprocmask, stream.packet.context = { cpu_id = 1 }, event.fields = { how = 0, nset = 0x7FFF28A2A040, oset = 0x7FFF28A2A048 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = ioctl, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 255, cmd = 21520, arg = 140733875134380 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = sys_rt_sigprocmask, stream.packet.context = { cpu_id = 1 }, event.fields = { how = 2, nset = 0x7FFF28A29FC0, oset = 0x7FFF28A29FC8 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = setpgid, stream.packet.context = { cpu_id = 1 }, event.fields = { pid = 0, pgid = 4235 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = sys_exit_group, stream.packet.context = { cpu_id = 1 }, event.fields = { error_code = 0 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 1 }
name = sys_gettimeofday, stream.packet.context = { cpu_id = 1 }, event.fields = { tv = 0x7FFF0E61DC10, tz = 0x0 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
```
<table>
<thead>
<tr>
<th>CPU Top</th>
<th>PID</th>
<th>TID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.00</td>
<td>2580</td>
<td>2580</td>
<td>Xorg</td>
</tr>
<tr>
<td>3.50</td>
<td>15021</td>
<td>15021</td>
<td>lttnngtop</td>
</tr>
<tr>
<td>2.39</td>
<td>5957</td>
<td>5957</td>
<td>awesome</td>
</tr>
<tr>
<td>2.35</td>
<td>15035</td>
<td>15035</td>
<td>gimp</td>
</tr>
<tr>
<td>0.15</td>
<td>14523</td>
<td>14523</td>
<td>kworker/0:2</td>
</tr>
<tr>
<td>0.12</td>
<td>14762</td>
<td>14762</td>
<td>kworker/1:2</td>
</tr>
<tr>
<td>0.09</td>
<td>14887</td>
<td>14887</td>
<td>/usr/bin/x-term</td>
</tr>
<tr>
<td>0.08</td>
<td>6021</td>
<td>6021</td>
<td>xscreensaver</td>
</tr>
<tr>
<td>0.07</td>
<td>18470</td>
<td>18470</td>
<td>icedove-bin</td>
</tr>
<tr>
<td>0.03</td>
<td>2498</td>
<td>2498</td>
<td>acpid</td>
</tr>
<tr>
<td>0.03</td>
<td>3</td>
<td>3</td>
<td>ksoftirqd/0</td>
</tr>
<tr>
<td>0.03</td>
<td>22469</td>
<td>22469</td>
<td>/usr/bin/x-term</td>
</tr>
<tr>
<td>0.02</td>
<td>6019</td>
<td>6019</td>
<td>bluetooth-apple</td>
</tr>
<tr>
<td>0.02</td>
<td>15028</td>
<td>15028</td>
<td>/usr/bin/x-term</td>
</tr>
<tr>
<td>0.02</td>
<td>30114</td>
<td>30114</td>
<td>firefox-bin</td>
</tr>
<tr>
<td>0.01</td>
<td>14954</td>
<td>14954</td>
<td>kworker/u:1</td>
</tr>
<tr>
<td>0.01</td>
<td>14504</td>
<td>14504</td>
<td>kworker/u:2</td>
</tr>
<tr>
<td>0.01</td>
<td>22342</td>
<td>22342</td>
<td>udisks-daemon</td>
</tr>
<tr>
<td>0.01</td>
<td>171</td>
<td>171</td>
<td>scsi_eh_1</td>
</tr>
<tr>
<td>0.01</td>
<td>14736</td>
<td>14736</td>
<td>evince</td>
</tr>
<tr>
<td>0.01</td>
<td>22360</td>
<td>22360</td>
<td>gnome-settings-</td>
</tr>
<tr>
<td>0.01</td>
<td>1</td>
<td>1</td>
<td>init</td>
</tr>
<tr>
<td>0.00</td>
<td>4114</td>
<td>4114</td>
<td>uml_switch</td>
</tr>
<tr>
<td>0.00</td>
<td>14506</td>
<td>14506</td>
<td>flush-254:2</td>
</tr>
</tbody>
</table>

Eclipse Linux Tools Project: LTTng support

- http://wiki.eclipse.org/Linux_Tools_Project/LTTng

Extensible trace analysis using the Tracing and Monitoring Framework, Alexandre Montplaisir, Ericsson, Tracing Summit, Aug. 30, Room Nautilus 3, 16h10.
> LTTng-Graph

http://git.dorsal.polymtl.ca/~jdesfossez?p=lttng-graph
> Viewer Libraries and Bindings

• Lib Babeltrace: a C/C++ library for reading CTF traces (MIT BSD-style license)

• Python bindings over Lib Babeltrace planned to be merged into Babeltrace 1.1
  – http://git.efficios.com/?p=babeltrace.git
  bindings/python branch.
Tracing across kernel and user-space

```
$ lttng create
Session auto-20120827-133620 created.
Traces will be written in /home/test4/lttng-traces/auto-20120827-133620
$ lttng enable-event -u -a
All UST events are enabled in channel channel0
$ lttng enable-event -k -a
All kernel events are enabled in channel channel0
$ lttng start
Tracing started for session auto-20120827-133620
$ lttng destroy
Session auto-20120827-133620 destroyed
$ 
```
> Tracing across kernel and user-space (2)

```bash
test4@thinkos:~ $ babeltrace -f trace:domain /home/test4/lttng-traces/auto-20120827-133620 | grep "\{ cpu_id = 0 \}" | less
```

![Screenshot of a terminal window showing output from `babeltrace` command]

**EffiциOS**

Mathieu Desnoyers

August 29th, 2012

26
> Tracing across nodes

babeltrace --clock-force-correlate trace1 trace2 |grep -e sys_read -e sys_write |less
Approx time by event – 1 thread (nanoseconds)

In the same elapsed time, the number of events that can be traced can be multiplied by, respectively, 8.5 and 21.4, compared to Dtrace and SystemTap.
Approx time by event – 8 threads (nanoseconds)

In the same elapsed time, the number of events that can be traced can be multiplied by, respectively, 69 and 200, compared to Dtrace and SystemTap.
> Strace vs LTTng Tracing

Timing of a find of 100000 files
(seconds)

- **find**: 0.54 seconds
- **find + lttng**: 1.4 seconds
- **find + strace**: 38.8 seconds

LTTng speedup
44.49:1 vs strace
> New Features (LTTng 2.1)

- Network Streaming over TCP
- LTTng-UST filtering before data collection
- lttng-sessiond(8) health monitoring API.
> Network Streaming over TCP

```
root@squeeze-i386:~# lttng create -U net://thinkos
Session auto-20120827-141834 created.
Traces will be written in net://thinkos
root@squeeze-i386:~# lttng enable-event -k -a
All kernel events are enabled in channel channel0
root@squeeze-i386:~# lttng start
Tracing started for session auto-20120827-141834
root@squeeze-i386:~# lttng destroy
Session auto-20120827-141834 destroyed
root@squeeze-i386:~# lttng-traces/squeeze-i386$ find .
.
./auto-20120827-141834
./auto-20120827-141834/kernel
./auto-20120827-141834/kernel/metadata
./auto-20120827-141834/kernel/channel0_1
./auto-20120827-141834/kernel/channel0_0
```
> LTTng-UST Filtering

```
test4@thinkos:~$ lttng create
Session auto-20120827-142450 created.
Traces will be written in /home/test4/lttng-traces/auto-20120827-142450

test4@thinkos:~$ lttng enable-event -u -a --filter "(intfield > 42 && intfield <= 44) || longfield == 1"
All UST events are enabled in channel channel0

test4@thinkos:~$ lttng start
Tracing started for session auto-20120827-142450

test4@thinkos:~$ lttng destroy
Session auto-20120827-142450 destroyed

```
> LTTng-UST Filtering (2)
Distributions / Integration

- LTTng 0.x
  - Wind River Linux, Montavista, STlinux, Linaro, Yocto, Mentor Embedded Linux, ELinOS.

- LTTng 2.0
  - Ubuntu 12.04 LTS
  - Debian
  - SuSE Enterprise RT Linux
  - Linux Foundation LTSI
  - Fedora : process ongoing
  - Etc...
> Collaborations

- Interfacing: **GDB** tracepoints can interoperate with LTTng UST tracepoints, the **Eclipse Tracing Monitoring Framework** supports LTTng CTF traces, **Perf** PMU counters are used by LTTng, The Multi Core Association is defining a **Common Trace Format** (CTF), for which LTTng 2.0 is a reference implementation.

Tracing Summit 2012

The Tracing Summit 2012 will be held in San Diego, on August 30th, 2012, as part of the Linux Plumbers Conference 2012.

8h30 Can mainstream tracing meet embedded needs?, Frank Rowand, Sony
8h55 What We Want in Our Toolkit: Thoughts From a Mission Critical Low Latency Environment, Vinod Kutty, CME Group
9h20 Troubleshooting complex problems with built-in diagnostic, Dominique Toupin, Ericsson
9h45 The Linux Perf Tools: Overview and Current Developments, Arnaldo Carvahlo de Melo, Red Hat
10h10 - 10h25 break
10h25 Tracing the Guest OS from Host via Shared Memory, Masami Hiramatsu, Hitachi
10h50 Shrinking core dump on the fly, Thomas Gleixner, Linutronix
11h15 Tracing Well With Others: Integration of GDB Tracepoints Into Trace Tools, Stan Shebs, Mentor Graphics
11h40 Ftrace and Multiple buffers, Steven Rostedt, Red Hat
12h05 - 13h30 lunch
13h30 The Road ahead of Uprobes: Plans and features in pipeline, Srikar Dronamraju, IBM
13h45 Systemtap and new connections: dyinst, pcp, uprobes, David Smith and Josh Stone, Redhat
14h20 LTTng and Nexus Trace for Freescale QoriQ Devices, Ed Martinez, Freescale
14h35 - 14h45 break
14h45 Interoperability Between Tracing Tools with the Common Trace Format (CTF), Mathieu Desnoyers, EfficiOS
15h10 Making linsched useful, Dhaval Giani, University of Toronto
15h35 LTTngTop: Human Readable Trace Viewer, Julien Desfossez, EfficiOS
16h00 - 16h10 break
16h10 Extensible trace analysis using the Tracing and Monitoring Framework, Alexandre Montplaisir, Ericsson
16h45 - 18h00 Open Discussion, Where do we go from here?
Questions?

LTTng 2.0 available at http://lttng.org

• EfficiOS
  • http://www.efficios.com
  • LTTng Information
    • http://lttng.org
    • lttng-dev@lists.lttng.org