LTTng 2.0 : Kernel and Application tracing for the Enterprise.

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

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

- Tracing overview
- LTTng 2.0 features
- LTTng 2.0 UI examples
- Benchmarks
- Trace viewer & analysis tools
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 maintenance phases.

- **High-availability, high-throughput servers**
  - Development and maintenance: ensure high performance, low-latency in production.

- **Embedded**
  - System development, maintenance of deployed systems.
> Tracers timeline

- Tracing commonly used for embedded real-time systems (small traces) and for early high performance SMP servers (SGI, IBM). Then comes Linux...
  
1. 1999: LTT

2. 2005: LTTng

3. 2005: Dtrace (Solaris/xBSD)

4. 2005: SystemTap (RedHat)

5. 2008: Ftrace

6. 2009: Perf

7. 2012: LTTng 2.0
Why do we need a LTTng 2.0?

- Need more flexible trace data layout format
  - Introduce Common Trace Format (CTF)
- Introduction of user-space tracing (UST)
  - Leverage common control infrastructure for kernel and user-space tracing
  - Simplification of the kernel-level infrastructure
- Need more flexible ring buffer
  - Snapshot, mmap and splice, global and per-cpu, kernel and user-space, configurable crash dump support.
Combined kernel and user-space tracing
Multi-user concurrent sessions

```
client

Alice
normal user

Alice App 1
command
create shm

buffers 1
write

sessiond

ust-consumerd

consume

client

Bob
normal user

Bob Apps

sessiond

ust-consumerd

consume

buffers
write

Alice App N
command
create shm

buffers n
write
```

EfficiOS
Mathieu Desnoyers
April 30th, 2012

9
> LTTng 2.0 Tracing Session

• Multiple domains:
  • Kernel, User-space
  • Eventually: Hypervisor, multiple hosts

• Controlled through same UI/API:
  • lttng -k ...
  • lttng -u ...

• Correlation across domains (common time-line)
• Viewed by pointing trace viewer to the top-level trace collection directory
> Session, domain, channel and event

**Session hash table (indexed by name)**

- Tracing session 1
- Tracing session 2
- Tracing session n

**Domains**

- Kernel tracer
- User-space tracer

- KERNEL
  - Channels
  - Events

- UST
  - Channels
  - Events

- UST PID
  - Channels
  - Events

...
LTTng 2.0 Low-Overhead Tracing Architecture

**Host-Side User Interfaces**
- **Babeltrace** (MIT/BSD)
  - Trace converter
  - Trace pretty printer
  - Allow open source and proprietary plugins
- **LTTV** (GPLv2)
  - Trace display and analysis
  - Trace control
  - Allow open-source plugins
- **Eclipse Tracing and Monitoring Framework** (EPL)
  - Trace display and analysis
  - Trace control
  - Allow open-source and proprietary plugins

**Host**
- SSH connexion

**Target**

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

**LTTng Session Daemon** (GPLv2)
- liblttngctl (LGPLv2.1)
- liblttng-ust (LGPLv2.1)
- liblttng-ust-ctl (GPLv2)

**LTTng Consumer Daemon** (GPLv2)
- Zer-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

**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)

**Tracepoint and Probes Characteristics**
- 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)

**CTF† over TCP/UDP/SSH**

**Local storage**
- Common Trace Format (CTF)
  - 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)

**Visibility**
- 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)

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

**Host**
- SSH connexion

**Target**
- SSH connexion
> LTTng 2.0 Kernel Tracer

- Build against a vanilla or distribution kernel, without need for additional patches,
- Tracepoints, System calls, Function tracer, Perf CPU Performance Monitoring Unit (PMU) counters, kprobes, and kretprobes support,
- Supports multiple tracing sessions,
- Flight recorder mode, snapshots, supported at the tracer level, not supported by lttng-tools 2.0 yet (coming in 2.1).
> 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-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.
> TRACEPOINT_EVENT

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)
    )
)
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);
}
```
> LTTng-UST 2.0 Buffering

- Port of the lib ring buffer to user-space.
- Supports buffering between processes through POSIX shared memory maps.
- Fast-paths stay in user-space (no system call).
- Wake-up though pipes.
- Buffers per process (for security), shared with consumer. Faster/lower memory consumption per-user global buffers feature planned for 2.1.
LTTng Tracing Session Daemon

- Central (system-wide) and per-user instances.
- Controls
  - LTTng kernel tracer
  - LTTng-UST application/library tracer
  - Right management by UNIX socket file access rights.
  - System-wide tracing controlled by tracing group.
  - File descriptors passed through UNIX sockets
- Presents a unified notion of system-wide tracing session, with multiple “domains”.
LTTng UI examples

```bash
lttng list -k                      # list available kernel tracepoints
lttng create mysession            # create session “mysession”
lttng enable-event -k -a           # enable all syscalls/tracepoints
lttng enable-event -k --syscall -a # trace system calls
lttng enable-event sched_switch,sched_wakeup -k
lttng enable-event aname -k --probe symbol+0x3
lttng enable-event aname -k --function <symbol_name>
lttng add-context -k -e sched_switch -t pid   # add PID context
lttng add-context -k -e sched_switch -t perf:cpu-cycles
lttng start                        # start tracing

...                              # stop tracing
lttng stop
lttng destroy                    # teardown session
# text output
babeltrace -n $HOME/lttng-traces/mysession-<date>-<time>
```
> LTTng 2.0 high-speed "strace"

lttng enable-event --syscall -a

```
compudj@squeeze-amd64: ~

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 = 0xB48008, 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 = 0x7935 }
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 = 0xB48081, 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 = 0x7FFFF8A2A040, oset = 0 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = sys_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 = 0x7FFFF8A29FC0, oset = 0 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
name = sys_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 = 0x7FFFFD610C10, tz = 0x0 }
name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }
```
Common Trace Format

- 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.
> 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.
> Common Trace Format

- 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.
Approx time by event – 1 thread
(nanoseconds)

- LTTng UST: 280
- UST (W/O OPT): 500
- Dtrace: 2400
- SystemTap: 6000
> Userspace Tracing Benchmark

Approx time by event – 8 threads (nanoseconds)

- UST: 280
- Dtrace: 19400
- SystemTap: 56000

EfficiOS
Timing of a find of 100000 files (seconds)

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

> Strace vs LTTng Tracing
Top vs LTTngTop

- Top:
  - CPU time: 1.3%
  - Syscalls/s: 1344

- LTTngTop:
  - CPU time: 0.5%
  - Syscalls/s: 78
> Distributions / Integration

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

• LTTng 2.0
  • Ubuntu 12.04 LTS
  • Debian
  • Novell SuSE Enterprise RT Linux
  • Linux Foundation LTSI
  • Fedora / RedHat : process ongoing
> Collaborations

- Financial support: CAE, DRDC, Ericsson, Google, Opal-RT, Revolution Linux, with matching contributions from CRIAQ, NSERC, Prompt.

- Maintainer and main developer: Mathieu Desnoyers, EfficiOS

- Integrating and redistributing LTTng: Wind River, MontaVista, Linaro, LTSI, Debian, Ubuntu, Suse, Fedora/Red Hat (pending).

- Interfacing: GDB tracepoints can interoperate with LTTng UST tracepoints, The Multi Core Association is defining a Common Trace Format (CTF), for which LTTng 2.0 is a reference implementation.

- Code contributions: over 70 individuals from more than 20 companies including Ericsson, Google, IBM, Red Hat...
> Babeltrace

```
[13:58:29.128909723] (+0.0000002475) sys_read: { 0 }, { "firefox-bin", 3363 }, { fd = 5, buf = count = 16 }
[13:58:29.128911513] (+0.0000001790) exit_syscall: { 0 }, { "firefox-bin", 3363 }, { ret = -11
[13:58:29.128919672] (+0.0000008159) sys_write: { 0 }, { "firefox-bin", 3363 }, { fd = 5, buf
, count = 8 }
[13:58:29.128921404] (+0.0000001732) exit_syscall: { 0 }, { "firefox-bin", 3363 }, { ret = 8 }
[13:58:29.128922884] (+0.0000001480) sys_read: { 0 }, { "firefox-bin", 3363 }, { fd = 19, buf
, count = 1 }
[13:58:29.128925765] (+0.0000002881) exit_syscall: { 0 }, { "firefox-bin", 3363 }, { ret = 1 }
[13:58:29.128928120] (+0.0000002355) sys_write: { 0 }, { "firefox-bin", 3363 }, { fd = 5, buf
, count = 8 }
[13:58:29.128929552] (+0.0000001432) exit_syscall: { 0 }, { "firefox-bin", 3363 }, { ret = 8 }
[13:58:29.129020005] (+0.00000090453) exit_syscall: { 0 }, { "acpid", 1536 }, { ret = 1 }
[13:58:29.129025587] (+0.0000005582) sys_rt_sigprocmask: { 0 }, { "acpid", 1536 }, { how = 0, oset = 0x0, sigsetsize = 8 }
[13:58:29.129027993] (+0.0000002406) exit_syscall: { 0 }, { "acpid", 1536 }, { ret = 0 }
[13:58:29.129030188] (+0.0000002195) sys_poll: { 0 }, { "acpid", 1536 }, { ufds = 0x7FFF2A055D meout_msecs = 0 }
[13:58:29.129032570] (+0.0000002382) exit_syscall: { 0 }, { "acpid", 1536 }, { ret = 0 }
[13:58:29.129033929] (+0.0000001359) sys_rt_sigprocmask: { 0 }, { "acpid", 1536 }, { how = 1, oset = 0x0, sigsetsize = 8 }
[13:58:29.129035144] (+0.0000001215) exit_syscall: { 0 }, { "acpid", 1536 }, { ret = 0 }
[13:58:29.129037520] (+0.0000002376) sys_read: { 0 }, { "acpid", 1536 }, { fd = 4, buf = 0x7FF
 = 24 }
```
<table>
<thead>
<tr>
<th>CPU Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU(%)</td>
</tr>
<tr>
<td>10.00</td>
</tr>
<tr>
<td>5.50</td>
</tr>
<tr>
<td>0.93</td>
</tr>
<tr>
<td>0.29</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.04</td>
</tr>
<tr>
<td>0.04</td>
</tr>
<tr>
<td>0.02</td>
</tr>
<tr>
<td>0.02</td>
</tr>
<tr>
<td>0.02</td>
</tr>
<tr>
<td>0.02</td>
</tr>
<tr>
<td>0.01</td>
</tr>
<tr>
<td>0.01</td>
</tr>
<tr>
<td>0.01</td>
</tr>
<tr>
<td>0.01</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
</tr>
</tbody>
</table>

Status:
Starting display
Pause
> Eclipse Linux Tools Project: LTTng support

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

LTTng 2.0 support planned for Juno release.
• Will be ported to LTTng 2.0 soon.
> LTTng-Graph
LTTng studio

April 30th, 2012
> The road ahead

- New LTTV and TMF with Common Trace Format and State System
- Live tracing
- Other language bindings
- Remote tracing
- Filtering
- Dependency analysis
- Event abstraction
> Questions?

LTTng 2.0 available at http://lttng.org

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