LTTng-UST: Efficient System-Wide User-Space Tracing
Christian Babeux, Software Developer, EfficiOS,

Background in embedded and ASIC tools,

Active contributor to the LTTng projects:
  - lttng-tools & lttng-ust,
  - CI infra, Website, Twitter.

AUR package maintainer for Arch Linux.
Overview of LTTng 2.x and UST,
User-space instrumentation sources,
Trace format standardisation efforts,
Tales from a user-space tracer,
Recent features & future work.
Overview of LTTng 2.x
Overview of LTTng 2.x

- Unified user interface, API, kernel and user space tracers,

- Trace output in a unified format,

- Low overhead,

- Shipped in distros: Ubuntu, Debian, Suse, Fedora, Linaro, Wind River, etc.
Project overview

- Tracers
- Utilities
- Viewers
Tracers

- **lttng-modules**: kernel tracer module, compatible with kernels from 2.6.38* to 3.9,
- **lttng-ust**: user-space tracer, in-process library.

* Kernel tracing is now possible on 2.6.32 to 2.6.37 by backport of 3 Linux Kernel patches [1].
Utilities

- lttng-tools: cli utilities and daemons for trace control,
  - lttng: cli utility for tracing control,
  - lttng-sessiond: tracing registry daemon,
  - lttng-consumerd: consume trace data,
  - lttng-relayd: network streaming daemon.
• babeltrace: cli text viewer, trace converter, plugin system,
• lttngetop: ncursor top-like viewer,
• Eclipse lttnget plugin: front-end for lttnget, collect, visualize and analyze traces, highly extensible.
Overview of LTTng-UST
LTTng-UST – Features

Pure userspace implementation,
- Shared memory map between apps and trace consumers,
- Portable to other OS: BSDs, Cygwin (experimental).

Optimized for low-overhead, high-throughput [2],
- Generic kernel ringbuffer ported to userspace,
- Efficient concurrent data structures for trace control.
Dynamically enabled, statically defined instrumentation,

Per user tracing and system-wide tracing,
  - Tracing group for system-wide tracing.

Traces recoverable even after application crash.
Users instrument their applications with static tracepoints,

liblttng-ust, in-process library, dynamically linked with application,

Session setup, etc.,

Run app, collect traces,

Post analysis with viewers.
Tracing session - Setup

- Session setup: `$lttng create`
- User-space event enabling: `$lttng enable-event -u -a`
- Start tracing: `$lttng start`
Tracing session - A wild app appears

Instrumented application

- Listener thread spawned via constructor (GCC extension),
- App registration,
- Send SHM and wait fd.

UST listener thread

sessiond

consumerd

UNIX Socket

SHM

Pipe
Tracing session – App. execution & teardown

- App running,
- Events written to ringbuffer,
- Notification of data availability via pipe,
- App unregistered via destructor.

Instrumented application

UST listener thread

sessiond

consumerd

$viewer

Tracepoint 1
...
Tracepoint N

UNIX Socket

SHM

Pipe
User-space instrumentation sources
TRACEPOINT_EVENT(
    /* Provider name */
    ust_tests_hello,

    /* Tracepoint name */
    tptest,

    /* Type, variable name */
    TPARGS(int, anint,
        long *, values,
        float, floatarg),

    /* Type, field name, expression */
    TPFIELDS(ctf_integer(int, intfield, anint),
        ctf_array(long, arrfield1, values, 3),
        ctf_float(float, floatfield, floatarg))
)
void function(void)
{
    int i = 0;
    long vals[3] = { 0x42, 0xCC, 0xC001CAFE };  
    float flt = M_PI;

    [...] 
    tracepoint(ust_tests_hello, 
        tptest, 
        i, 
        &vals, 
        flt);

    [...] 
}
Integration result of Collaboration Summit 2011 discussions,

Compatibility with SystemTAP SDT,

- Users can use SystemTAP with tracepoint() instrumented code.
Kernel patchset merged in 3.5,

LTTng integration:
- Initial lttng-modules patchset proposed [4],
- Need usability improvement
- Interface not exported by kernel
Trace format standardisation efforts
Trace format standardisation efforts

How standards proliferate:

(See: A/C chargers, character encodings, instant messaging, etc.)

Situation: There are 14 competing standards.

14?! Ridiculous! We need to develop one universal standard that covers everyone’s use cases. Yeah!

Soon:

Situation: There are 15 competing standards.

Source: xkcd.com/927
**Trace format standardisation efforts**

*Joking aside:* We need a common open format,

*Collaboration:* Multicore Association, Ericsson,

**Goals of the Common Trace Format (CTF):**
- Common format for SW and HW traces,
- Portable,
- Compact,

**Tools based on CTF:**
- LTTng 2.x, Babeltrace, Eclipse LTTng plugin
- GDB (save trace to CTF) [3],
- Javeltrace
Common Trace Format

Self-described, packet-based format.

typealias ... ;
trace { ... };
clock { ... };
stream { ... };
event { ... };

Metadata

Stream 0
Packet 0
Header
Context
Event 0
...

Stream n
Packet 0
Header
Context
Event 0
...

Packet m

Packet m
"Interoperability Between Tracing Tools with the Common Trace Format",
- Mathieu Desnoyers at Linux Plumbers 2012 [5]

Common Trace Format (CTF) Specification [6],

Common Trace Format compliance testsuite [7].
Tales from a user-space tracer
Ringbuffer delivery notification use a pipe,
– Traced applications can receive SIGPIPE if consumer end dies abruptly.

Suppress SIGPIPE only in our lib without affecting signal handling for the rest of the process [8].
• Thread Local Storage (TLS) variable storage in dynamically libs. allocated when first used [9],

• Rely on internal glibc mutex to protect against dynamic linker,

• Same mutex is held while running ctor/dtor,
TLS & constructors (cont.)

liblttng-ust constructor

```
dl_load_lock (ok)
```

```
ust_lock (fail)
```

```
UST listener thread
```

```
ust_lock (ok)
```

```
TLS lookup
```

```
dl_load_lock (fail)
```

Spawn
• Take mutex within constructors while TLS fixup performed,

⚠️ Deadlock!

• Workaround: Force TLS fixup within lib ctor.
Tracing of apps closing all fds
Close all the things

- When daemonizing, some apps close all available fds,
Tracing of apps closing all fds

- When daemonizing, some apps close all available fds,

⚠️ No communication == No tracing.

- Fix: None for the moment.
Recent features & future work
Recent features

2.1 (Basse Messe)

Network streaming over TCP,
- Introduce lttng-relayd, receive traces from remote consumers.

Filtering before data collection,
- C-like syntax, bytecode interpreter.
- UST only for the moment.

Session daemon health monitoring API.
Network streaming over TCP

```bash
test4@thinkos:~$ lttng-relayd
```

```bash
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:~#
```

```bash
test4@thinkos:~/$ 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
```

37
Filtering (1)

Filter:

“(intfield > 42 && intfield <= 44) || longfield == 1”
Filtering (2)

“(intfield > 42 && intfield <= 44) || longfield == 1”
2.2 (Cuda, Currently in RC)

- Per-uid buffers in UST,

- Context filtering,
  - "$ctx.procname == "demo*"",
  - "$ctx.vpid > 9000".

- Trace file size limits,
Future work

Flight recorder mode tracing (2.3),

Trace data extracted on core dump (2.3),

Java tracing.
Future work (cont.)

- Tracer triggers actions on specific events & filters
- Compressed, encrypted streaming and storage,
- LTTng accepted in Google Summer of Code [10].
  - Dynamic instrumentation support in UST,
  - Android port.
Conclusion

✔️ Usability of user space tracing in production
References

- [2] – lttnng-modules README -
- [7] - Common Trace Format compliance testsuite
- [8] – LTTng-UST – 2C44F5B9 - Fix UST SIGPIPE handling
- [10] – LTTng GSoC 2013 Ideas list