Tailor-made trace analyses
Jérémie Galarneau

EfficiOS Inc.
- Vice President
- http://www.efficios.com

Maintainer of
- LTTng-Tools
- Babeltrace
Babeltrace 1.x

- MIT licensed Common Trace Format (CTF) reference implementation (2011)
  - Served as the *de facto* LTTng command-line trace reader

- Introduced Python bindings to read traces (2013)
  - Provide users a way to prototype an analysis rapidly without using text-based tools (awk, sed, grep, etc.)
    - Debugging
    - Testing
  - Scripts maintained as internal tools by some users
  - Basis of LTTng analyses

- Provides a CTF production library (CTF writer) (2014)
  - Used by perf to convert traces to CTF
Limitations of Babeltrace 1.x

- Only supports in-tree plugins
  - Does not expose a stable ABI to plugins
  - No solution to support proprietary (or niche) trace formats out of tree

- Design makes it impossible to implement “LTTng-live” support as a plugin

- API quirks
  - Does not allow caching of events
  - One iterator per trace
  - Hard/impossible to work with multiple clock sources
Redefining Babeltrace’s scope

- Realized that trace analysis tools were not keeping up with the tracers
  - Capturing trace is only half of the battle
  - Use in production shows that working with huge traces is challenging
  - Text-based analyses do not scale and are hard to maintain... but they are useful!

- Project scope changed from a trace *converter* to a trace *manipulation* tool
  - Support more input and output formats
  - Trim, filter, and add information to traces
  - Make it easy to assemble “blocks” to build a custom trace analysis pipeline
Redefining Babeltrace’s scope

- The babeltrace client becomes a “host” application for trace processing graphs

- User-defined processing graphs
  - Standardize trace processing (shareable graph configurations)
  - Can be assembled programatically
    - Usable from external tools, such as viewers
  - Can be used to process trace “chunks” independently
Building a graph

- Building a graph on the command line
  
  
  $ babeltrace run
  
  --component=my-source:src.plugin-name.my-src
  
  --component=my-sink:sink.plug.my-sink
  
  --connect=my-source:my-sink

  Don't worry, there are helpers for common scenarios...

- You can build graphs using the C and Python APIs
Redefining Babeltrace’s scope

- Cross-platform
  - Linux
  - Windows (native and Cygwin)
  - Solaris
  - BSDs
  - macOS

- Preserve the current Babeltrace 1.x Python and CTF writer interface
Babeltrace 2.0

- Provides components which allow everything Babeltrace 1.x could do
  - CTF file system source
  - CTF file system sink
  - LTTng-live source
  - dmesg source
  - Muxer
  - Trimmer
  - Debugging information injector

- Components can be written in C, C++, and Python
  - Stable ABI allows out-of-tree components
Scenarios – Trimming a trace

$ babeltrace /path/to/trace --begin=22:14:38
--end=22:15:07
--output-format=ctf --output=/path/to/trace2
Scenarios – Record part of a live trace

$ babeltrace --input-format=lttng-live
  net://localhost/host/my_host/my_session
  --output-format=ctf --output=/path/to/trace

Diagram:

```
CTF trace
  source.ctf.lttng-live
    bt-plugin-ctf.so
  filter.utils.muxer
    bt-plugin-utils.so
  sink.ctf.fs
    bt-plugin-ctf.so
```

Command:
babeltrace convert
Scenarios – Print debugging information

$ babeltrace --debug-info /path/to/trace
Scenarios – Mux multiple formats

```
$ babeltrace
/path/to/trace1 /path/to/trace2
--component source.custom.src --path /path/to/trace
```
Easy to prototype new components

• You can write components in Python and use them from command line just like built-in components
  – Insert only the logic you need to build your analysis

• Quick example: a callstack view inspired by *uftrace* in ~50 lines
  – https://github.com/jgalar/TracingSummit2017
User space callstack view

- Build binary using `-finstrument-functions`
- Enable userspace tracing with LTTng (see GitHub link)
- `LD_PRELOAD="liblttng-ust-cyg-profile-fast.so" ./my_binary`
Demo
Future Work

- Currently at v2.0.0-pre4
  - APIs are not frozen yet
  - Works on all supported platforms
- Targetting the first Release Candidate for November
  - Optimizations which may affect the APIs
  - Documentation
- Support for CTF 2 (v2.1, if CTF 2 is specified soon)
- Future components
  - Filtering (v2.1)
  - State tracker, Period/span tracking, ideas?
Questions?

Babeltrace

- diamond.org/babeltrace
- github.com/efficios/babeltrace
- lttng-dev@lists.lttng.org

EfficiOS

- www.efficios.com
- @efficios