Google

Tracing in Google Chrome : Overview, challenges and userspace-kernel interaction

Georges Khalil, Google

Bruce Dawson, Google

Plan

- Chrome architectural overview
  - Multi-process Architecture
  - Overview of threads in the browser
  - Inter-thread communication
- Built-in tracer (about:tracing)
- Integration between ETW and Chrome
  - Motivation
  - Implementation details
  - Overview of UIForETW
- Conclusion

Chrome architectural overview

Multi-process Architecture

- Motivation for using multiple processes
- Types of processes
- Processes have multiple threads

Chrome architectural overview

Overview of some threads in the browser (main process)

- **ui_thread**: Main thread where the application starts up
- **io_thread**: Dispatcher thread that handles communication between the browser process and all the sub-processes
- **file_thread**: A general process thread for file operations
- **db_thread**: A thread for database operations

Chrome architectural overview

Inter-thread communication

- Posting tasks
- Callbacks

Inter-process communication

- IPC directives

⇒ Harder to profile and debug

Built-in tracer (about:tracing)

- Open source project called trace-viewer
- Part of the Catapult project
- Accessed in Chrome by navigating to about:tracing
- Works on all non-mobile platforms
- Records method signatures in a hierarchical view

Built-in tracer (about:tracing)

Advantages

- Platform independant
- Easy to use interface
  - Chrome developers
  - Web developers
  - Extension developers
  - Curious users
- Uses JSON as file format

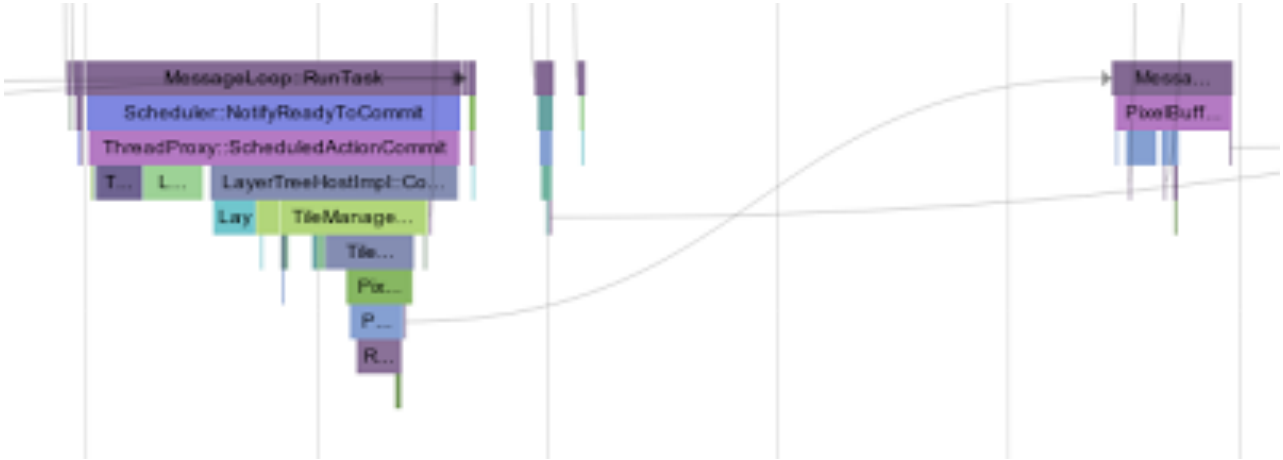
Built-in tracer (about:tracing)

Overview of UI (Demo)

Frame viewer data (Demo)

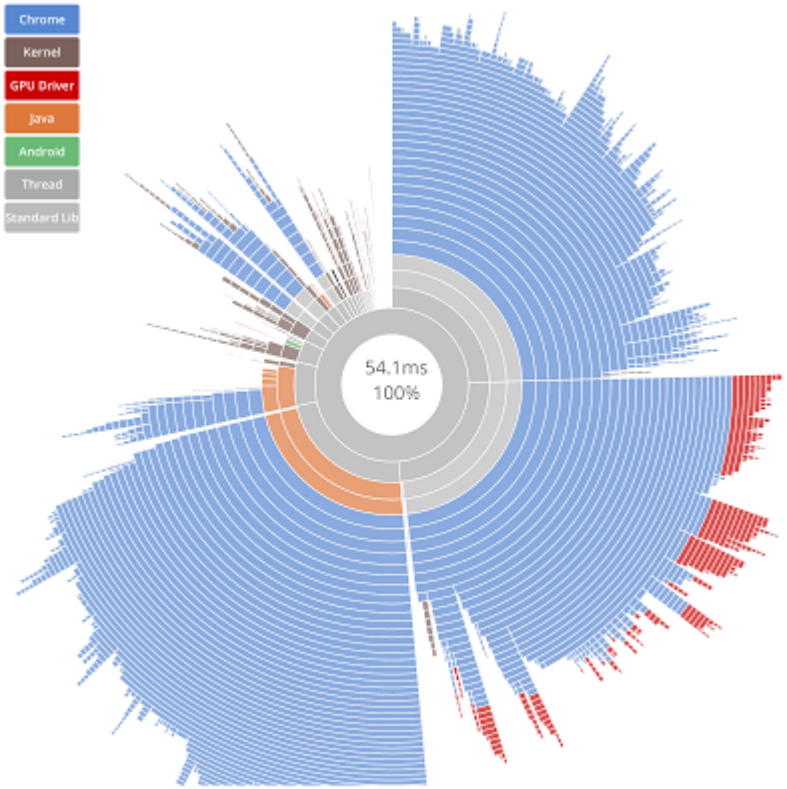
Built-in tracer (about:tracing)

- IPC messages



Built-in tracer (about:tracing)

- Perf profiling data



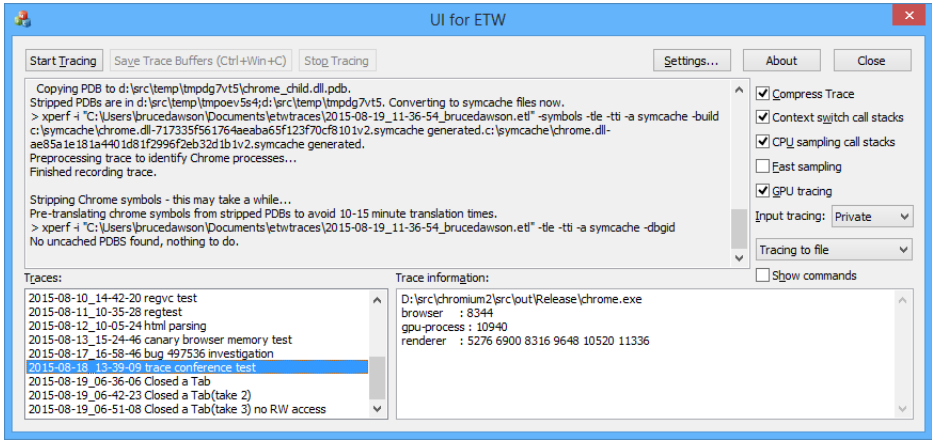
Integration between ETW and Chrome

Motivation

- Get system traces alongside Chrome events
- ETW is very lightweight and has little overhead
- WPA offers interesting features

Integration between ETW and Chrome

- ETW: Event Tracing for Windows
  - Can record context switches, file I/O, disk I/O, GPU activity, CPU samples with stacks, custom data, etc.
- UIforETW: Open source tool to record ETW traces, including Chrome tracing events



Conclusion

- Chrome shares many characteristics with operating systems
  - Tracing and profiling is very challenging
  - Performance is of utter importance
- Features a built-in tracer that is platform-independant
  - Built-in tracer is a separate open source project
  - Built-in tracer is very easy to use
- Recently added integration with ETW tracing on Windows
  - Allows a trace to include both userspace (Chrome) events and kernel events
- UIForETW is a frontend to easily capture ETW traces
  - Simple to use interface
  - Allows a user to quickly and easily capture a trace using ETW

References and links

https://github.com/catapult-project/catapult

https://www.chromium.org/developers/how-tos/trace-event-profiling-tool

https://github.com/google/UIforETW

Questions? Comments?

Emitting traces

- TRACE_EVENT_INSTANT**X** (with **X**=number of arguments, between 0 and 2)

```
529 bool GpuProcessHost::Init() {
530   init_start_time_ = base::TimeTicks::Now();
531
532   TRACE_EVENT_INSTANT0("gpu", "LaunchGpuProcess", TRACE_EVENT_SCOPE_THREAD);
533
534   std::string channel_id = process_->GetHost()->CreateChannel();
535   if (channel_id.empty())
536     return false;
537
```

Emitting traces

- TRACE_EVENT_BEGIN**X** / TRACE_EVENT_END**X**

```
795   if (command_line.HasSwitch(switches::kHostRules)) {
796     TRACE_EVENT_BEGIN0("startup", "IOThread::InitAsync:SetRulesFromString");
797     globals_->host_mapping_rules->SetRulesFromString(
798         command_line.GetSwitchValueASCII(switches::kHostRules));
799     TRACE_EVENT_END0("startup", "IOThread::InitAsync:SetRulesFromString");
800   }
```

Emitting traces

- TRACE_EVENT**X**

history_service.cc

```
873
874 bool HistoryService::Init(
875     bool no_db,
876     const std::string& languages,
877     const HistoryDatabaseParams& history_database_params) {
878   TRACE_EVENT0("browser,startup", "HistoryService::Init")
879   SCOPED_UMA_HISTOGRAM_TIMER("History.HistoryServiceInitTime");
880   DCHECK(thread_) << "History service being called after cleanup";
881   DCHECK(thread_checker_.CalledOnValidThread());
882
```

Chrome architectural overview