Profiling Low-End Platforms using HawkTracer Profiler

Marcin Kolny

Amazon

FOSDEM 2019

February 03, 2019

Agenda

- Profilers - introduction
- Why do we have another profiler?
- HawkTracer features
- Demo
- Q/A

Introduction to profilers

Performance profiling - analyzing performance of the application

- sample-based - regular samples to see which methods/resources are used (e.g. perf)

```
while (do_profile)
{
   read_process_callstack(pid)
   sleep(1s)
}
print_statistics()
```

- instrumentation-based - developer *instruments* the code to get specific information (e.g. HawkTracer)

```
start = now();
foo();
stop = now();
printf("foo() time: ", stop - start);
```

HawkTracer - genesis

Environment:

- C/C++ (Native) & LUA/JavaScript (Scripted)
- Low-end platform
- Limited access to the device
- No well-known tools available

HawkTracer - genesis

Environment:

- C/C++ (Native) & LUA/JavaScript (Scripted)
- Low-end platform
- Limited access to the device
- No well-known tools available

Let's build a profiler!

HawkTracer - genesis

Environment:

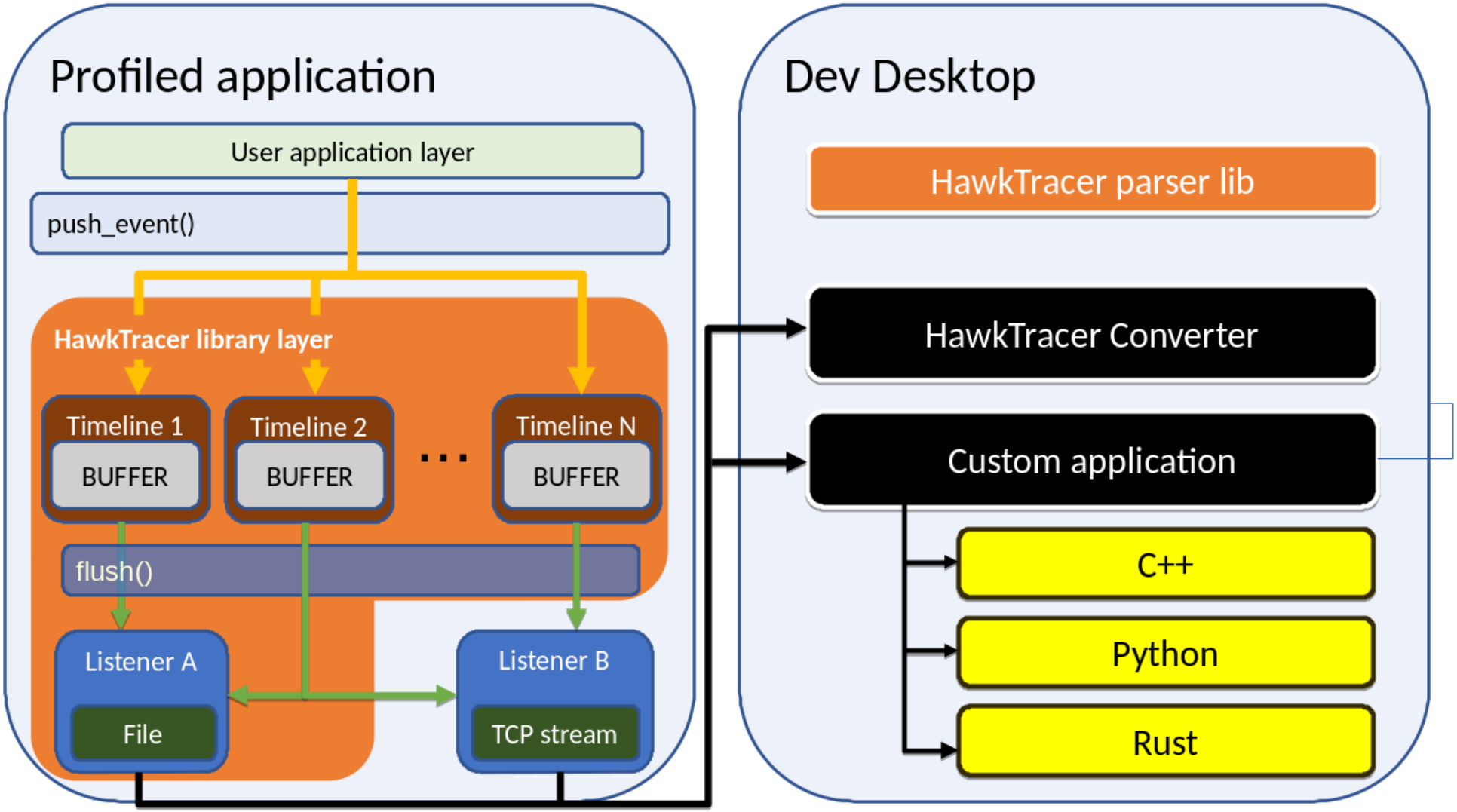
- C/C++ (Native) & LUA/JavaScript (Scripted)
- Low-end platform
- Limited access to the device
- No well-known tools available

Let's build a profiler!

Requirements:

- User-space only
- Built as a library (no need to run a new process)
- Low-overhead
- Persistent storage not required
- Easily portable to other platforms
- Easy to use

HawkTracer - high level architecture



HawkTracer - base components

Events

- predefined/user-defined event types
- support inheritance
- ability to introspect the structure at runtime (MKCREFLECT library)

Timelines

- Lock-free / locking (thread safe)
- Internal buffers

Timeline listeners

- User-defined callbacks

Defining new HawkTracer event class

Declare event klass:

```
HT_DECLARE_EVENT_KLASS(
  ResourceUsageEvent,           // event name
  HT_Event,                     // base event klass
  (INTEGER, uint64_t, cpu),     // field definition (type, C type, field name),
  (INTEGER, uint64_t, memory) // field definition (type, C type, field name)
  /* , (ANOTHER FIELD)... */
)
```

Generated code:

```
typedef struct {
    HT_Event base;
    uint64_t cpu;
    uint64_t memory;
} ResourceUsageEvent;
```

```
typedef struct {
    HT_EventKlass* klass;
    HT_TimestampNs timestamp;
    HT_EventId id;
} HT_Event;
```

```
/* serialization function */
size_t ht_ResourceUsageEvent_fnc_serialize(HT_Event* event, HT_Byte* buffer);

/* runtime type info, klass ID */
HT_EventKlass* ht_ResourceUsageEvent_get_event_klass_instance();

/* register type in a system */
HT_EventKlassId ht_ResourceUsageEvent_register_event_klass();
```

Event stream

- Metadata stream

```
[
  {
    class_id = 987,
    klass_name = "ResourceUsageEvent",
    fields = [
      {
        name: "cpu_usage",
        type_name: "uint64_t",
        type: INTEGER,
        sizeof: 8
      },
      // ...
    ]
  }
]
```

- Event stream (36 bytes):

```
0x00 0x00 0x03 0xDB                                     // class_id (987)
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x08   // timestamp (8)
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x10   // event sequence number (16)
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x27   // cpu_usage (39)
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x7C   // mem_usage (124)
```

Timeline

- User-created timeline

```
HT_Timeline* ht_timeline_create(size_t buffer_capacity,
                                HT_Boolean thread_safe,
                                HT_Boolean serialize_events,
                                const char* listeners,
                                HT_ErrorCode* out_error_code);
```

- Global timeline
  - per-thread instance
  - shared listeners
  - accessor method:

```
HT_Timeline* ht_global_timeline_get(void);
```

Time measurement

- Measure scope (C++ and GNU C only)

```
void foo()
{
  HT_TP_FUNCTION(timeline); // the same as HT_TP_SCOPED_STRING(timeline, "foo");
  // ...
  { // non-function scope
    HT_TP_SCOPED_STRING(timeline, "internal"); // or HT_TP_STRACEPOINT
    // ...
  }
}
```

- Measure arbitrary code

```
ht_feature_callstack_start_string(timeline, "label");
// ...
ht_feature_callstack_stop(timeline);
```

- Output

```
{
  // from HT_Event:
  klass_id, timestamp, event_id,

  duration, // nanoseconds
  thread_id,
  label
}
```

Integration with your project

- pkg-config

```
gcc your_file.c $(pkg-config --cflags --libs hawktracer)
```

Integration with your project

- pkg-config

```
gcc your_file.c $(pkg-config --cflags --libs hawktracer)
```

- CMake
  - External Project

```
# copy hawktracer.cmake from HawkTracer repository to your project
# examples/integrations/cmake-external-project/hawktracer.cmake
include(hawktracer.cmake)
add_executable(super_project main.cpp)
target_link_libraries(super_project hawktracer)
```

  - CMake module (if HawkTracer is installed)

```
find_package(HawkTracer 0.6.0 REQUIRED)
add_executable(super_project main.cpp)
target_link_libraries(super_project HawkTracer::hawktracer)
```

Integration with your project

- pkg-config

```
gcc your_file.c $(pkg-config --cflags --libs hawktracer)
```

- CMake
  - External Project

```
# copy hawktracer.cmake from HawkTracer repository to your project
# examples/integrations/cmake-external-project/hawktracer.cmake
include(hawktracer.cmake)
add_executable(super_project main.cpp)
target_link_libraries(super_project hawktracer)
```

  - CMake module (if HawkTracer is installed)

```
find_package(HawkTracer 0.6.0 REQUIRED)
add_executable(super_project main.cpp)
target_link_libraries(super_project HawkTracer::hawktracer)
```

- **Compile HawkTracer with your sources** - *recommended*
  Include following files to your project
  - `hawktracer.cpp` - can be compiled using C compiler
  - `hawktracer.h`
  - `ht_config.h`

Demo 1

Time measurements

Demo 2

Custom (Python) client

Future development

- Missing core features:
  - floating point numbers
  - optional fields
- More converters:
  - CTF
  - perfetto
- Official LUA and JS bindings
- Documentation improvements

Future development

- Missing core features:
  - floating point numbers
  - optional fields
- More converters:
  - CTF
  - perfetto
- Official LUA and JS bindings
- Documentation improvements

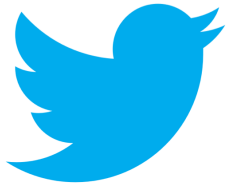
Help wanted!

Visit **www.hawktracer.org/community** to see how to get involved.

Links

- HawkTracer webpage:
  www.hawktracer.org
- Repository:
  github.com/amzn/hawktracer

- Twitter
  - @hawktracer - HawkTracer account
  - @l0ganek - my personal account



- HawkTracer Rust bindings:
  github.com/alexene/rust_hawktracer
  FOSDEM 2019 - Profiling Rust - Alexandru Ene