libside: Giving the Preprocessor a Break with a Tracer-Agnostic Instrumentation API

Mathieu Desnoyers, EfficiOS

Tracing Summit 2023
September 17—18
Bilbao, Spain

EfficiOS

DIAMON WORKGROUP

Outline

- Static instrumentation,
- User events,
- Pre-existing static instrumentation APIs,
- Userspace Instrumentation Desiderata,
- Libside
- Future Work

Static Instrumentation

- Statically defined in the instrumented code,
- Allows instrumented applications and libraries to express semantic:
  - event names,
  - fields names,
  - field types,
  - and more.
- Enabled dynamically at runtime.

Static Instrumentation Mechanisms

- Statically Defined Tracing (SDT)
- Linux kernel Tracepoints
- LTTng-UST Tracepoints
- Microsoft TraceLogging for ETW and LTTng
- (non-exhaustive list)

And There Appears User Events

- Merged into the Linux 5.18 merge window with little community review,
- Marked broken within 5.18 release candidates cycle to provide time for community feedback on the ABI,
- Feedback provided, marked unbroken for 6.4.

Concerns About User Events

- Exposes a stable ABI allowing applications to register their event names/field types to the kernel,
- Can be expected to have a large effect on application instrumentation,
- My concerns:
  - Should be co-designed with a userspace instrumentation API/ABI rather than only focusing on the kernel ABI,
  - Should allow purely userspace tracers to use the same instrumentation as userspace tracers implemented within the Linux kernel,
  - Tracers can target their specific use-cases, but infrastructure should be shared,
  - Limit fragmentation of the instrumentation ecosystem.

Limitations of SDT

- Implementation closely tied to ELF, targets mainly C/C++,
- Complex calling convention,
- Supported argument types limited to C basic types, and pointers (including pointers to strings),
- Handling of non-basic-types requires compiling specialized probe providers,
- Not natively suitable for other runtimes
  - Golang, Java, Javascript, Python, and others.
- Requires breakpoint for instrumentation, thus expensive round-trip to the kernel,
- No mechanism to coordinate concurrent use by kernel and purely user-space tracers.

Limitations of LTTng-UST Tracepoints

- Based on compilation of tracepoint probes,
- Multi-pass header inclusion:
  - hard to understand compiler errors.
- Generates a lot of code when the number of tracepoint signatures increases:
  - instruction cache pollution when tracing is active,
- Relying on C/C++ code generation prevents native integration with other runtimes (Golang, Java, Python, Erlang, Javascript, ...).
- Dependency on C function prototypes requires additional code to populate a tracepoint-agnostic ABI to hand over arguments to filter and field capture bytecode interpreter,
- Support of nested compound types not straightforward:
  - structures, arrays, variable-length arrays.
- Supports a single tracer (LTTng-UST).

Limitations of TraceLogging

- Overhead:
  - metadata sent with each event payload,
  - no mechanism to pre-register event types.

Userspace Instrumentation Desiderata

- Common instrumentation for kernel and purely userspace tracers,
- Instrumentation is self-described,
- Support compound and nested types,
- Support pre-registration of events,
- Do not rely on compiled event-specific code,
- Independent from ELF,
- Simple ABI for instrumented code, kernel, and user-space tracers,
- Support concurrent tracers,
- Natively cover statically-typed and dynamically-typed languages:
  - C/C++, Golang, Java, .NET, Python, Javascript, Rust, Erlang, and other runtimes,
- Expose API to allow dynamic instrumentation libraries to register their events/payloads.

libside

- **S**oftware **I**nstrumentation **D**ynamically **E**nabled,
- https://github.com/efficios/libside
- Instrumentation API/ABI:
  - Type system,
  - Helper macros for C/C++,
  - Express instrumentation description as data,
  - Instrumentation arguments are passed on the stack as a data array (similar to iovec) along with a reference to instrumentation description,
  - Instrumentation is conditionally enabled when at least one tracer is registered to it.
- Tracer-agnostic API/ABI:
  - Available events notifications,
  - Conditionally enabling instrumentation,
  - Synchronize registered user-space tracer callbacks with RCU,
  - Co-designed to interact with User Events.

Field Description ABI (Example)

```
enum side_type_label { SIDE_TYPE_S32, [...] };
enum side_type_label_byte_order { SIDE_TYPE_BYTE_ORDER_LE = 0, SIDE_TYPE_BYTE_ORDER_BE = 1, };

struct side_type_integer {
        const struct side_attr *attr;
        uint32_t nr_attr;
        uint16_t integer_size;          /* bytes */
        uint16_t len_bits;              /* bits. 0 for (integer_size * CHAR_BITS) */
        uint8_t signedness;             /* true/false */
        uint8_t byte_order;             /* enum side_type_label_byte_order */
} SIDE_PACKED;

struct side_type {
        uint32_t type;  /* enum side_type_label */
        union {
                struct side_type_integer side_integer;
                [...]
        } SIDE_PACKED u;
} SIDE_PACKED;
```

Event Description ABI

```
enum side_loglevel { SIDE_LOGLEVEL_EMERG = 0, [...] };

struct side_event_field {
        const char *field_name;
        struct side_type side_type;
} SIDE_PACKED;

struct side_event_description {
        uintptr_t *enabled;
        const char *provider_name;
        const char *event_name;
        const struct side_event_field *fields;
        const struct side_attr *attr;
        const struct side_callback *callbacks;
        uint64_t flags;
        uint32_t version;
        uint32_t loglevel;      /* enum side_loglevel */
        uint32_t nr_fields;
        uint32_t nr_attr;
        uint32_t nr_callbacks;
} SIDE_PACKED;
```

Event Arguments ABI (Example)

```
enum side_type_label { SIDE_TYPE_S32, [...] };

union side_integer_value {
        int32_t side_s32; [...]
} SIDE_PACKED;

union side_arg_static {
        union side_integer_value integer_value; [...]
} SIDE_PACKED;

struct side_arg {
        uint32_t type;  /* enum side_type_label */
        union {
                union side_arg_static side_static;
                union side_arg_dynamic side_dynamic;
        } SIDE_PACKED u;
} SIDE_PACKED;

struct side_arg_vec {
        const struct side_arg *sav;
        uint32_t len;
} SIDE_PACKED;
```

Instrumentation Helper Macros

```
#include <side/trace.h>

side_static_event(my_provider_event, "myprovider", "myevent", SIDE_LOGLEVEL_DEBUG,
        side_field_list(side_field_s32("myfield", side_attr_list())),
        side_attr_list()
);

int main()
{
        side_event(my_provider_event, side_arg_list(side_arg_s32(42)));
        return 0;
}
```

Demo Tracer

Tracer notified of events inserted

Tracer notified of events inserted

provider: myprovider, event: myevent

---

provider: myprovider, event: myevent, fields: [myfield: { value: 42 }]

Tracer notified of events removed

provider: myprovider, event: myevent

Tracer notified of events removed

Stack-Copy Basic Types

SIDE_TYPE_NULL,
SIDE_TYPE_BOOL,
SIDE_TYPE_U8,
SIDE_TYPE_U16,
SIDE_TYPE_U32,
SIDE_TYPE_U64,
SIDE_TYPE_S8,
SIDE_TYPE_S16,
SIDE_TYPE_S32,
SIDE_TYPE_S64,
SIDE_TYPE_BYTE,
SIDE_TYPE_POINTER,
SIDE_TYPE_FLOAT_BINARY16,
SIDE_TYPE_FLOAT_BINARY32,
SIDE_TYPE_FLOAT_BINARY64,
SIDE_TYPE_FLOAT_BINARY128,
SIDE_TYPE_STRING_UTF8,
SIDE_TYPE_STRING_UTF16,
SIDE_TYPE_STRING_UTF32,

Other Stack-Copy Types

```
/* Stack-copy compound types */
SIDE_TYPE_STRUCT,
SIDE_TYPE_VARIANT,
SIDE_TYPE_ARRAY,
SIDE_TYPE_VLA,
SIDE_TYPE_VLA_VISITOR,

/* Stack-copy enumeration types */
SIDE_TYPE_ENUM,
SIDE_TYPE_ENUM_BITMAP,

/* Stack-copy place holder for dynamic types */
SIDE_TYPE_DYNAMIC,
```

Gather Types

```
/* Gather basic types */
SIDE_TYPE_GATHER_BOOL,
SIDE_TYPE_GATHER_INTEGER,
SIDE_TYPE_GATHER_BYTE,
SIDE_TYPE_GATHER_POINTER,
SIDE_TYPE_GATHER_FLOAT,
SIDE_TYPE_GATHER_STRING,

/* Gather compound types */
SIDE_TYPE_GATHER_STRUCT,
SIDE_TYPE_GATHER_ARRAY,
SIDE_TYPE_GATHER_VLA,

/* Gather enumeration types */
SIDE_TYPE_GATHER_ENUM,
```

Dynamic Types

```
/* Dynamic basic types */
SIDE_TYPE_DYNAMIC_NULL,
SIDE_TYPE_DYNAMIC_BOOL,
SIDE_TYPE_DYNAMIC_INTEGER,
SIDE_TYPE_DYNAMIC_BYTE,
SIDE_TYPE_DYNAMIC_POINTER,
SIDE_TYPE_DYNAMIC_FLOAT,
SIDE_TYPE_DYNAMIC_STRING,

/* Dynamic compound types */
SIDE_TYPE_DYNAMIC_STRUCT,
SIDE_TYPE_DYNAMIC_STRUCT_VISITOR,
SIDE_TYPE_DYNAMIC_VLA,
SIDE_TYPE_DYNAMIC_VLA_VISITOR,
```

Event and Type Attributes

- Instrumentation can specify { key, value } pair attributes,
- Allows for tracer-specific custom extensions and pretty-printing hints.

Integer Field Base Attribute Example

- Integer field:

```
side_field_u16("u16base2", side_attr_list(side_attr("std.integer.base", side_attr_u8(2))))
```

- Prints as:

```
u16base2: { attr: [ { key: "std.integer.base", value: 2 } ], value: 0b0000000000110111 }
```

Formatted String Example

```
side_static_event_variadic(my_provider_event_format_string,
	"myprovider", "myeventformatstring", SIDE_LOGLEVEL_DEBUG,
	side_field_list(side_field_string("fmt", side_attr_list())),
	side_attr_list(side_attr("lang.c.format_string", side_attr_bool(true)))
);
void test_fmt_string(void)
{
	side_event_cond(my_provider_event_format_string) {
		side_arg_dynamic_define_vec(args,
			side_arg_list(side_arg_dynamic_string("blah", side_attr_list()), side_arg_dynamic_s32(123, side_attr_list())),
			side_attr_list()
		);
		side_event_call_variadic(my_provider_event_format_string,
			side_arg_list(side_arg_string("This is a formatted string with str: %s int: %d")),
			side_arg_list(side_arg_dynamic_field("arguments", side_arg_dynamic_vla(&args))),
			side_attr_list()
		);
	}
}
```

Formatted String Example

- Results in (without pretty-printing):

provider: myprovider, event: myeventformatstring,

attr: [{ key: "lang.c.format_string", value: true }],

fields: [fmt: { value: "This is a formatted string with str: %s int: %d" }],

fields:: [arguments:: { elements:: [{ value:: "blah" }, { value:: 123 }] }]

How Tracers Interact with libside

- User-space tracer
  - Register callback to be notified when a new event description is available,
  - Register callback to be called when an event is emitted.
- Kernel tracer
  - Libside invokes User Events ioctls(),
  - User Events modifies the enabled state when it wishes to be called when a libside event is emitted.

Future Work

- Extensibility of libside ABI,
- Portability of libside ABI (pointers vs uint64_t),
- Reevaluate the usefulness of each supported type,
- Integration with LTTng-UST:
  - Register LTTng-UST as a libside tracer,
  - Implement event registration notification callbacks,
  - Translate libside event descriptions to bytecode,
  - Bytecode interpreter for ring buffer serialization,
  - Bytecode interpreter for runtime filter and field capture,
  - Integrate libpatch (dynamic instrumentation) with LTTng-UST through libside,
- Integration with User Events.

Questions / Comments ?