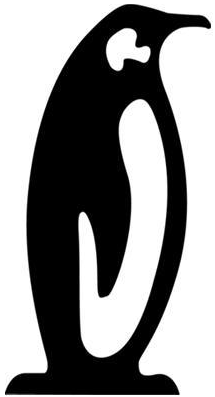


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

SFrame for Arm64 Reliable Stacktrace

Dylan Hatch <dylanbhatch@google.com>

Weinan Liu <wnliu@google.com>

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Agenda

- What is a Stacktrace?
- Why Reliable Stacktrace?
- Providing Reliable Stacktrace
- Current Reliable Stacktrace/Livepatch Support for Arm64
- SFrame as a Solution

What is a Stacktrace?

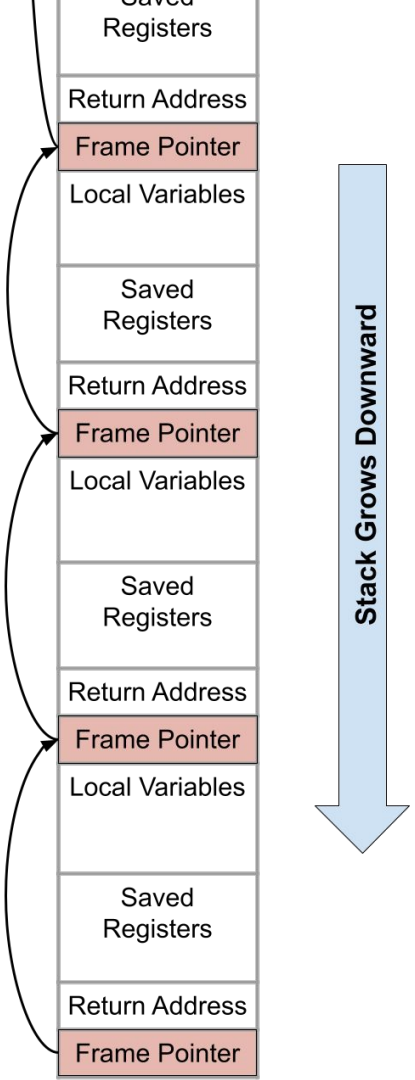
What is a Stacktrace?

- A stacktrace is a record of functions called by a thread of execution, as recovered from the call stack.
- Typically they appear in a human-readable format along with other diagnostic information

```
[<c12ba080>] ? dump_stack+0x44/0x64
[<c103ed6a>] ? __warn+0xfa/0x120
[<c109e8a7>] ? module_put+0x57/0x70
[<c109e8a7>] ? module_put+0x57/0x70
[<c103ee33>] ? warn_slowpath_null+0x23/0x30
[<c109e8a7>] ? module_put+0x57/0x70
[<f80ca4d0>] ? gp8psk_fe_set_frontend+0x460/0x460 [dvb_us
[<c109f617>] ? symbol_put_addr+0x27/0x50
[<f80bc9ca>] ? dvb_usb_adapter_frontend_exit+0x3a/0x70 [d
[<f80bb3bf>] ? dvb_usb_exit+0x2f/0xd0 [dvb_usb]
[<c13d03bc>] ? usb_disable_endpoint+0x7c/0xb0
[<f80bb48a>] ? dvb_usb_device_exit+0x2a/0x50 [dvb_usb]
[<c13d2882>] ? usb_unbind_interface+0x62/0x250
[<c136b514>] ? __pm_runtime_idle+0x44/0x70
[<c13620d8>] ? __device_release_driver+0x78/0x120
[<c1362907>] ? driver_detach+0x87/0x90
[<c1361c48>] ? bus_remove_driver+0x38/0x90
[<c13d1c18>] ? usb_deregister+0x58/0xb0
[<c109fbb0>] ? SyS_delete_module+0x130/0x1f0
[<c1055654>] ? task_work_run+0x64/0x80
[<c1000fa5>] ? exit_to_usermode_loop+0x85/0x90
[<c10013f0>] ? do_fast_syscall_32+0x80/0x130
[<c1549f43>] ? sysenter_past_esp+0x40/0x6a
```

What is a Stacktrace?

- Conventionally, these traces are made by following the saved frame pointer as a linked-list up the stack
- The saved return address is used to identify the function associated with each frame

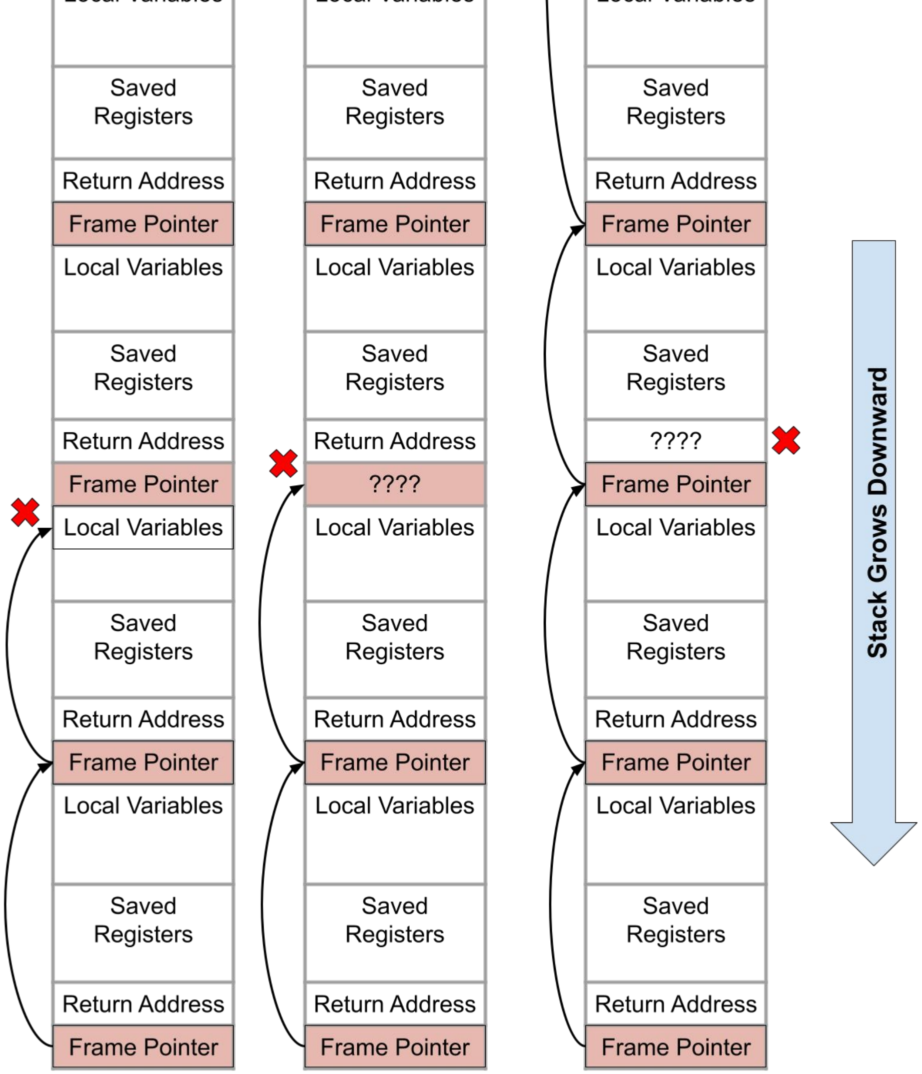


What is an **Unreliable** Stacktrace?

A stacktrace can be unreliable if it:

- Does not contain the full stack
- Contains incorrect frame/function data
- Contains duplicate entries

How does this happen? More on this later...



東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Why Reliable Stacktrace?

Why Reliable Stacktrace?

Stacktraces are generally useful for profiling and debugging:

- KDump analysis
- Warnings
- Ftrace stack tracing
- Perf event profiling

The benefit of these tools is diminished (but still potentially useful) when the stack itself becomes unreliable.

Refs

https://www.man7.org/linux/man-pages/man2/perf_event_open.2.html

https://docs.kernel.org/trace/ftrace.html#stack-trace

https://docs.kernel.org/admin-guide/kdump/kdump.html

Why Reliable Stacktrace? – Livepatch

Livepatch is a system for applying kernel fixes without rebooting.

- Implemented as a module containing replacement functions
- Control flow is redirected to replacement functions using the dynamic ftrace framework
- Consistency is enforced on a per-task level

Livepatch relies on **reliable stacktraces** to implement this consistency model

Refs

https://docs.kernel.org/livepatch/livepatch.html#

Why Reliable Stacktrace? – The Livepatch Consistency Model

Upon enabling a patch, the Livepatch subsystem enters a transition state

- Individual tasks switch over to the patched state confirmed not to be executing any affected functions
- Transition state only ends when all tasks have been switched to the target patch state

Livepatch applies three distinct strategies to enforce its consistency model

Refs

https://docs.kernel.org/livepatch/livepatch.html#consistency-model

Why Reliable Stacktrace? – The Livepatch Consistency Model

The first two are partial solutions that **do not** rely on stacktracing:

1. Kernel exit switching. A task is switched when it returns to user space from a system call/user space IRQ/signal.
   - Patching I/O-bound user tasks which are sleeping on an affected function. SIGSTOP and SIGCONT will exit the kernel.
   - Patching CPU-bound user tasks the next time they get interrupted by an IRQ.

2. Call klp_update_patch_state() on current task directly
   - Useful for idle “swapper” tasks, which never exit the kernel. Call in the klp_update_patch_state() in idle loop.
   - No complete solution exists for kthreads.

Refs

https://docs.kernel.org/livepatch/livepatch.html#consistency-model

Why Reliable Stacktrace? – The Livepatch Consistency Model

The third is essential for livepatch success:

3. Check the stacks of sleeping tasks. If no patched functions are on a task's stack mark it as patched. Needs `HAVE_RELIABLE_STACKTRACE`

   - This is the most effective strategy to apply patches, and usually succeeds after one try
   - **Architectures without `HAVE_RELIABLE_STACKTRACE` are not considered fully supported for livepatch.**

Refs

https://docs.kernel.org/livepatch/livepatch.html#consistency-model

Providing Reliable Stacktrace

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Providing Reliable Stacktrace – Considerations

Identifying successful termination of stack trace

- Unwinding ends at an expected kernel entry point
- Location of final stack frame matches expectation
- Unwinder did not exit early with error

Refs

https://docs.kernel.org/livepatch/reliable-stacktrace.html

Providing Reliable Stacktrace – Considerations

Interrupts and exceptions can occur at any point in a function's lifetime.

This can cause inconsistencies on the stack:

- Frame pointer can be invalid during function prologue/epilogue
- Return address may be held in a general purpose register (not on the stack)

Refs

https://docs.kernel.org/livepatch/reliable-stacktrace.html

Providing Reliable Stacktrace – Considerations

Ftrace and kprobe trampoline can result in the rewriting/obscuring of the return address. Thankfully, this problem is solved with generic helpers in both cases:

- `ftrace_graph_ret_addr()`
- `kretprobe_find_ret_addr()`

Refs

https://docs.kernel.org/livepatch/reliable-stacktrace.html

Providing Reliable Stacktrace – Considerations

Link register unreliability – the value of the link register may not be updated in lock-step with the setup/teardown of the stack frame.

- Can result in “double counted” functions in the stack trace

Refs

https://docs.kernel.org/livepatch/reliable-stacktrace.html

Providing Reliable Stacktrace – The ORC Unwinder

On x86, the ORC unwinder provides reliable stacktraces

- At build time, objtool generates a mapping from instruction address (`.org_unwind_ip`) to stack metadata (`.orc_unwind`).
- Creates an off-band method of unwinding the stack

Refs

https://docs.kernel.org/arch/x86/orc-unwinder.html

Providing Reliable Stacktrace – The ORC Unwinder

- Stack unwind is then done reliably using this metadata
- With Frame Pointers disabled completely, this has performance benefits over using frame pointer
- Enables reliable unwind, even across exception boundaries

However, Arm64 is missing the required support from objtool so ORC unwind tables cannot be generated

Refs

https://docs.kernel.org/arch/x86/orc-unwinder.html

Providing Reliable Stacktrace – The ORC Unwinder

A prior RFC proposed to add Arm64 support to objtool to generate ORC tables, and enable reliable stacktrace

- Proposed a “dynamic” form of frame pointer validation
    - The frame pointer on the stack is validated against the frame pointer offset pulled from the ORC table
    - If the calculated FP does not match the expected one, the unwind is considered unreliable
- Involved substantial reorganization of the objtool/ORC code to separate the architecture-generic vs architecture-specific components

Refs

https://lore.kernel.org/live-patching/20230202074036.507249-1-madvenka@linux.microsoft.com/

"Madhavan T. Venkataraman" <madvenka@linux.microsoft.com>

Providing Reliable Stacktrace – The ORC Unwinder

Feedback from the upstream community indicated a problem with this approach:

- Objtool needs to reverse-engineer the control flow generated by the compiler, creating the possibility that objtool will fail to do so for certain edge cases (e.g. jump tables)
- As code generation strategies evolve, objtool will need to be maintained to keep with the changes, making this approach inherently brittle.

Refs

https://lore.kernel.org/live-patching/ZByJmnc%2FXDcqQwoZ@FVFF77S0005N.cambridge.arm.com/

Mark Rutland <mark.rutland@arm.com>

Providing Reliable Stacktrace – The ORC Unwinder

A solution using data produced directly from the compiler would be preferred.

- This feedback aligns with past feedback to avoid reverse-engineering control flow: https://lpc.events/event/11/contributions/971/
- SFrame was proposed as an alternative solution.

Refs

https://lore.kernel.org/live-patching/ZByJmnc%2FXDcqQwoZ@FVFF77S0005N.cambridge.arm.com/

Mark Rutland <mark.rutland@arm.com>

Current Reliable Stacktrace Support for Arm64

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Current Reliable Stacktrace Support for Arm64

In v6.12 a patch series landed adds the concept of a “Meta” frame record to explicitly identify key points of interest on the stack:

- Exception boundaries
- Final frames

Marks the two missing pieces to meeting the `HAVE_RELIABLE_STACKTRACE` requirements.

Refs

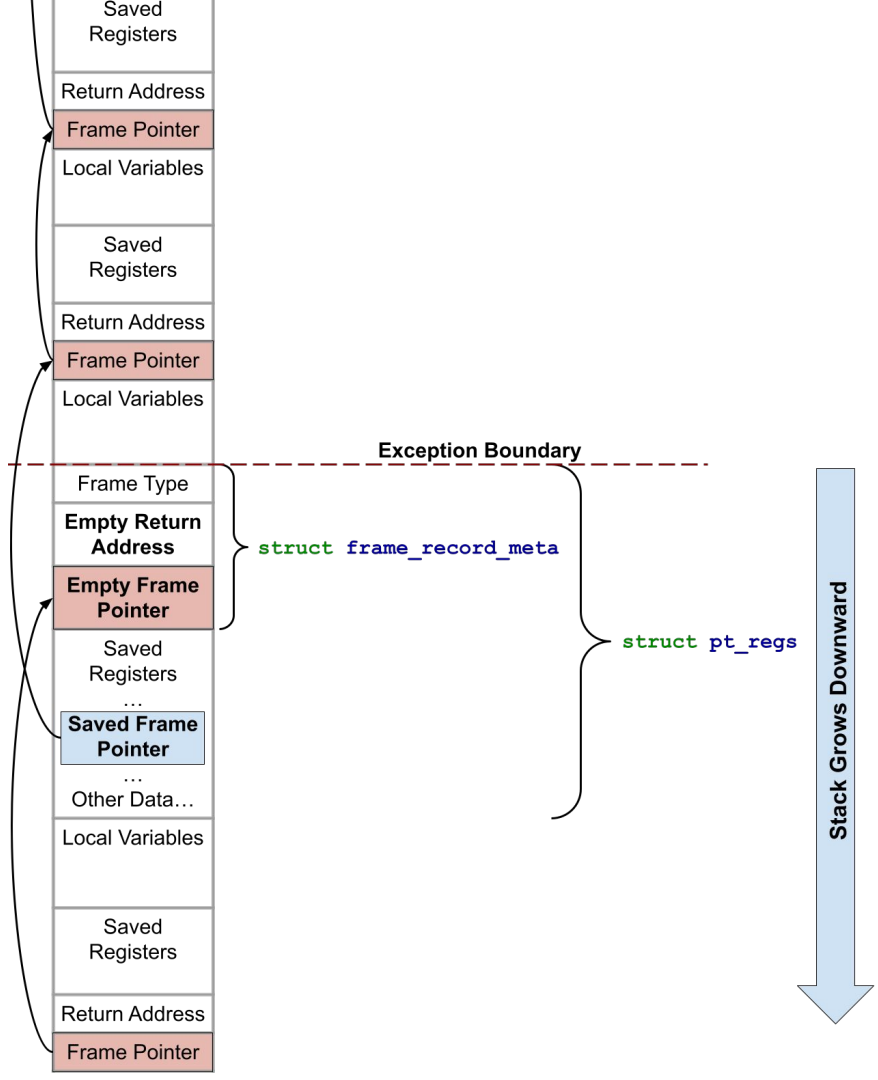
https://lore.kernel.org/linux-arm-kernel/20241017092538.1859841-1-mark.rutland@arm.com/

Mark Rutland <mark.rutland@arm.com>

Current Reliable Stacktrace Support for Arm64

`FRAME_META_TYPE_PT_REGS` indicates the presence of a saved `struct pt_regs`

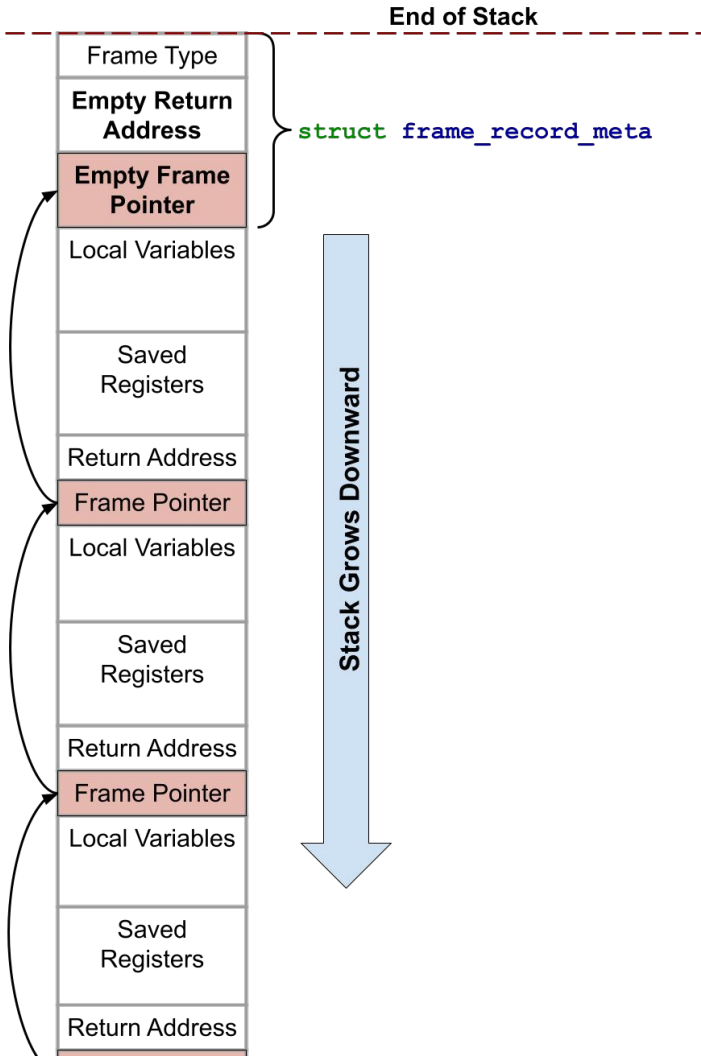
- Occurs at the exception boundary
- Special handling needed to recover the saved Frame Pointer and Instruction Pointer



Current Reliable Stacktrace Support for Arm64

`FRAME_META_TYPE_FINAL` Indicates the final stack frame

- Once validated to match the task's `struct pt_regs`, this confirms successful termination of the stacktrace



Current Reliable Stacktrace Support for Arm64

Source tracking in the unwind state indicates how each frame was recovered

- "C" – from caller of unwind function
- "T" – from blocked task's saved PC
- "P" – from a pt_regs::pc

Possible 2nd source

- "F" – from fgraph
- "K" – from kretprobes

```
| Call trace:
|  show_stack+0x20/0x38 (CF)
|  dump_stack_lvl+0x60/0x80 (F)
|  dump_stack+0x18/0x28
|  nmi_cpu_backtrace+0xfc/0x140
|  nmi_trigger_cpumask_backtrace+0x1c8/0x200
|  arch_trigger_cpumask_backtrace+0x20/0x40
|  sysrq_handle_showallcpus+0x24/0x38 (F)
|  __handle_sysrq+0xa8/0x1b0 (F)
|  handle_sysrq+0x38/0x50 (F)
|  pl011_int+0x420/0x570 (F)
|  __handle_irq_event_percpu+0x60/0x220 (F)
|  handle_irq_event+0x54/0xc0 (F)
|  handle_fasteoi_irq+0xa8/0x1d0 (F)
|  generic_handle_domain_irq+0x34/0x58 (F)
|  gic_handle_irq+0x54/0x140 (FK)
|  call_on_irq_stack+0x24/0x58 (F)
|  do_interrupt_handler+0x88/0xa0
|  el1_interrupt+0x34/0x68 (F)
|  el1h_64_irq_handler+0x18/0x28
|  el1h_64_irq+0x6c/0x70
|  default_idle_call+0x34/0x180 (P)
|  default_idle_call+0x28/0x180 (L)
|  do_idle+0x204/0x268
|  cpu_startup_entry+0x3c/0x50 (F)
|  rest_init+0xe4/0xf0
|  start_kernel+0x738/0x740
|  __primary_switched+0x88/0x98
```

Refs

https://lore.kernel.org/linux-arm-kernel/20241017092538.1859841-1-mark.rutland@arm.com/

Mark Rutland <mark.rutland@arm.com>

Current Reliable Stacktrace Support for Arm64

"L" would also indicate when the current frame was recovered from the Link Register, but this feature was removed

- The LR is not always live when an exception occurs
- LR unwinding has to be skipped, meaning unwinding is still unreliable at the exception boundary

Refs

https://lore.kernel.org/all/20241211140704.2498712-2-mark.rutland@arm.com/

Mark Rutland <mark.rutland@arm.com>

Current Reliable Stacktrace Support for Arm64

A follow-up patch series completes support for reliable stacktrace and livepatch **without sframe**

- When an exception boundary is detected, stacktrace is marked as unreliable
- Instead of providing/guaranteeing reliable stacktrace in all cases, it is sufficient for `arch_stack_walk_reliable` to report when an individual unwind is unreliable
- In practice, a livepatch can be reliably landed with a second unwind pass

Refs

https://lore.kernel.org/live-patching/20250320171559.3423224-1-song@kernel.org/

Song Liu <song@kernel.org>

Current Reliable Stacktrace Support for Arm64

This can be considered a partial solution

- Exception boundaries are detected but not yet reliably unwound
- It remains to be seen how reliable this solution is on its own

It should also be noted that this type solution can only help the livepatch use case

- Other applications (and humans) gain no benefit if exceptions are still not reliable!

Refs

https://lore.kernel.org/live-patching/20250320171559.3423224-1-song@kernel.org/

Song Liu <song@kernel.org>

SFrame as a Solution

SFrame as a Solution for Reliable Stacktrace

SFrame provides a generalized, toolchain-based solution, inspired by the ORC unwinder

- Provides an assembler-generated `.sframe` section, which contains a two-level lookup structure
- Similar to ORC, but avoids reverse-engineering the binary

Refs

https://sourceware.org/binutils/docs/sframe-spec.html

https://static.sched.com/hosted_files/ossna2023/d1/OSS_LinuxCon_SFrame_final.pdf?_gl=1

Indu Bhagat <indu.bhagat@oracle.com>

SFrame as a Solution

Provides the minimal necessary information to provide a stacktrace:

- Frame Pointer (FP)
- Return Address (RA)
- Canonical Frame Address (CFA)

Refs

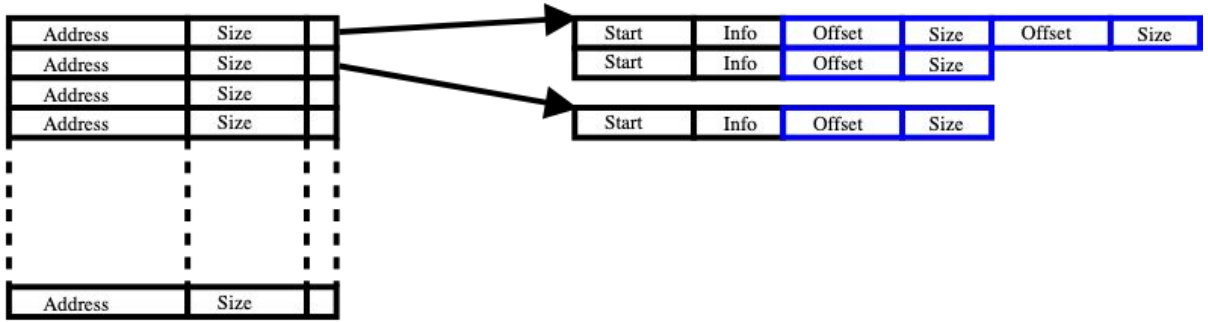
https://sourceware.org/binutils/docs/sframe-spec.html

https://static.sched.com/hosted_files/ossna2023/d1/OSS_LinuxCon_SFrame_final.pdf?_gl=1

Indu Bhagat <indu.bhagat@oracle.com>

SFrame as a Solution

In full, the SFrame table contains the information necessary to calculate the next frame's register values



Refs

https://lwn.net/Articles/1029189/

SFrame as a Solution

Sframe Function Descriptor Entry (FDE) table provides function metadata:

- Start address
- Size
- FRE count
- Other metadata...

FDEs can be sorted on PC for faster lookup

Refs

https://sourceware.org/binutils/docs/sframe-spec.html

SFrame as a Solution

Sframe Frame Row Entry (FRE) table is per-FDE and provides a complete record of stack trace information for all instructions in each function.

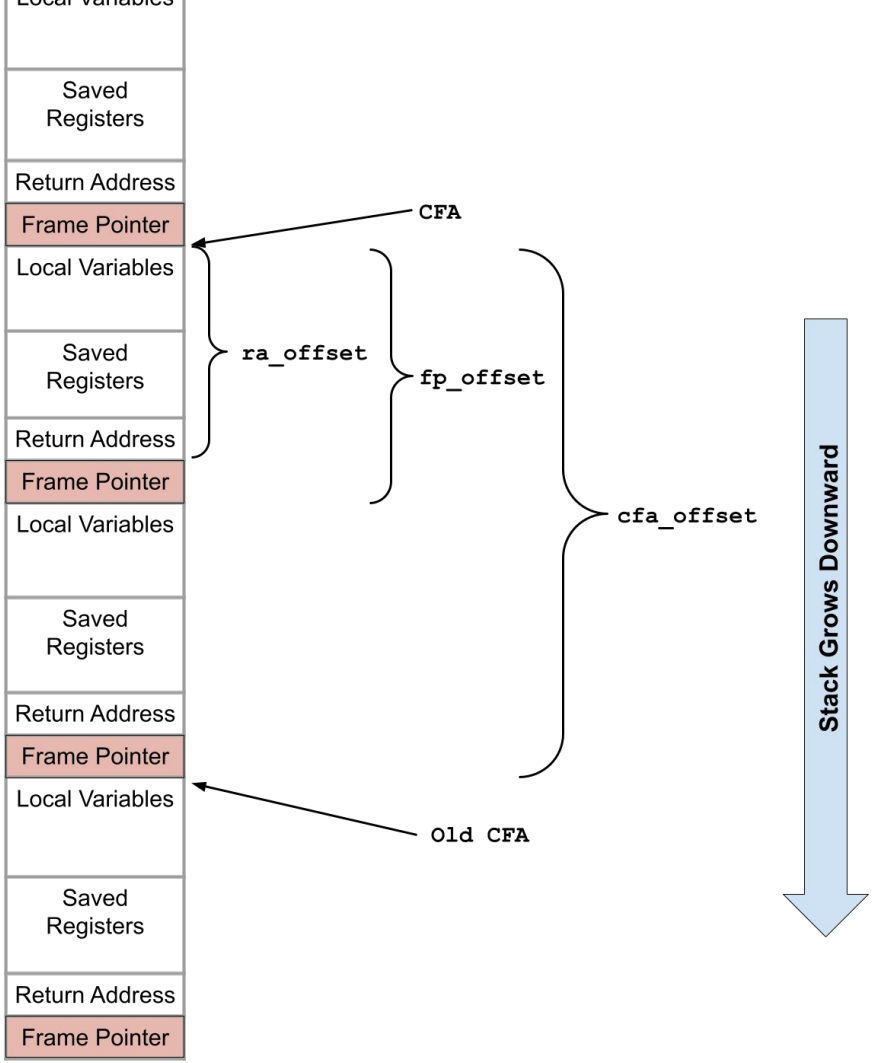
- FRE word info provides generic metadata:
  - CFA BASE_REG ID (SP or FP)
  - Other data...
- FRE definitions for Aarch64:
  - CFA = BASE_REG + offset1
  - RA = CFA + offset2
  - FP = CFA + offset3

Refs

https://sourceware.org/binutils/docs/sframe-spec.html

SFrame as a Solution

With the SFrame information, the locations of the Frame Pointer and Return Address can be calculated reliably at each unwind step.



SFrame as a Solution

Pseudocode unwind algorithm:

```
fre = sframe_find_fre (pc);
if (fre)
    base_reg_val = sframe_fre_base_reg_fp_p (fre) ? fp : sp;
    cfa_offset = sframe_fre_get_cfa_offset (fre);
    ra_offset = sframe_fre_get_ra_offset (fre);
    fp_offset = sframe_fre_get_fp_offset (fre);

    cfa = base_reg_val + cfa_offset;
    next_frame->sp = cfa;

    ra_stack_loc = cfa + ra_offset;
    next_frame->pc = read_value (ra_stack_loc);

    if (fp_offset is VALID)
        fp_stack_loc = cfa + fp_offset;
        next_frame->fp = read_value (fp_stack_loc);
    else
        next_frame->fp = fp;
else
    ret = ERR_NO_SFRAME_FRE;
```

Refs

https://sourceware.org/binutils/docs/sframe-spec.html

SFrame as a Solution – Our Proposal

How to unwind across exception boundaries

- Similar to ORC unwinder:
  - Annotate where the base register is and its relative CFA offset
  - Compiler will generate `.sframe` FRE provides a CFA offset relative to the base register

Refs

https://lore.kernel.org/lkml/20250904223850.884188-1-dylanbhatch@google.com/

Dylan Hatch <dylanbhatch@google.com>
Weinan Liu <wnliu@google.com>

SFrame as a Solution

- Sframe can support x86_64, AArch64, s390x
  - We can reuse same sframe unwind functions for different architectures
  - Generated by compiler directly so we don't have to run objtool after binaries are built

Refs

https://sourceware.org/binutils/wiki/sframe

SFrame as a Solution

- Complement with FP based approach. Fallback to FP approach if...
  - OOT kernel module is built without sframe information.
  - SFrame unwind encounters an error and is deemed unreliable

Refs

https://lore.kernel.org/lkml/20250904223850.884188-1-dylanbhatch@google.com/

Dylan Hatch <dylanbhatch@google.com>
Weinan Liu <wnliu@google.com>

SFrame as a Solution

SFrame for userspace call-stack unwinding support

- Add userspace unwinding support by accessing the SFrame data in the ELF section of the executable from the kernel.
- https://lore.kernel.org/lkml/cover.1730150953.git.jpoimboe@kernel.org/

Refs

https://lore.kernel.org/all/20251119132323.1281768-1-jremus@linux.ibm.com/

https://lore.kernel.org/all/cover.1737511963.git.jpoimboe@kernel.org/

https://lore.kernel.org/all/20250827201548.448472904@kernel.org/

Jens Remus <jremus@linux.ibm.com>
Josh Poimboeuf <jpoimboe@kernel.org>
Steven Rostedt <rostedt@kernel.org>

SFrame as a Solution – Ongoing Improvements

However, SFrame is currently only supported for GCC

- Support for LLVM is an ongoing effort
- Ongoing discussions point to an uncertainty about whether the tradeoffs SFrame presents align with the maintenance burden in LLVM

Toolchain support is essential for broader adoption of SFrame

Refs

https://discourse.llvm.org/t/rfc-adding-sframe-support-to-llvm/86900

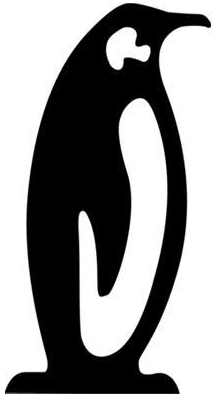
SFrame as a Solution – Ongoing Improvements

A number of improvements are anticipated for SFrame V3

- Flexible support for topmost frames
- Mark outermost frames

Refs

https://sourceware.org/binutils/wiki/sframe/sframev3todo



東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025