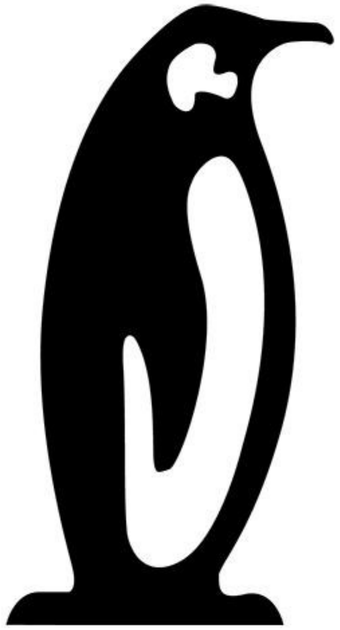

Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

CTF Frame in Linux kernel

Indu Bhagat

Toolchains Track

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

CTF Frame - overview

- A simple unwind format for virtual stack unwinding
  - Fast unwinding using a small unwinder
  - No stack machine, no complex expression encoding
  - Supported on x86_64 and aarch64
- Oversimplification: Akin to interpreted DWARF EH frame info, but for reduced state
  - CTF Frame only recovers: CFA, FP, RA
- Oversimplification: Similar to ORC in principle

CTF Frame - overview

- Not tied to CTF debug for unwinding
  - .ctf_frame can be used without .ctf section

CTF Frame Row Entry

- One self-sufficient FRE per PC (similar to ORC) to recover CFA, FP, RA. Helps unwind fast (less CPU intensive unwind)

```
type CTF_FRE_addr1 / addr2 / addr3 =
  struct {
    uint<8> / uint<16> / uint<32> fre_start_address;
    CTF_Frame_FRE_Info fre_info;
    union {
      int<8>[fre_info.offset_num] offsets_1B : fre_info.offset_size == 1B;
      int<16>[fre_info.offset_num] offsets_2B : fre_info.offset_size == 2B;
      int<32>[fre_info.offset_num] offsets_4B : fre_info.offset_size == 4B;
    } offsets;
  };
```

CTF Frame – more details

- .ctf_frame section – loadable, allocated section in a segment of its own (PT_GNU_CTF_FRAME)
- Supported in GNU Toolchain: Generated by GAS by parsing the .cfi_* directives embedded by the compiler. ld support to merge .ctf_frame.
- Size*: 0.8x (x86_64) , 0.7x (aarch64) relative to EH Frame

CTF Frame – what can it do ?

	DWARF-based .eh_frame	CTF Frame	Other application specific formats
Asynchronous	Yes	**Almost***	Yes?
Fast	Somewhat	Yes	Yes
Small unwind info	Yes	Yes	Somewhat
Small (simple) unwinder	**No**	**Yes**	**Yes**
Application specific	No	No	Yes
ABI/Arch support	Extensive	Aarch64, AMD64	x86_64
Toolchain support	Yes	**Yes**	**No**

How can CTF Frame help in the Linux kernel

- Can CTF Frame help objtool for ORC generation?
  - Simple to decode as compared to DWARF opcodes in .eh_frame
  - Simplify (can avoid?) control-flow reconstruction needs (the case of AARCH64 ?)

How can CTF Frame help in the Linux kernel ?

- Are there other unwinding needs, e.g., unwinding userspace stacks, where CTF Frame format can be used ?

In spirit of improving CTF Frame format

- What compiler optimizations are known to work against the needs of reliable backtraces ?
- What patterns/code stubs generated by the toolchain need improved unwind info ?

Extra Slides

CTF Frame – fully asynchronous?

- NOTE 1 - Some .cfi_* directives are skipped:
  - .cfi_negate_ra_state => No unwind around PAC insn in aarch64
  - .cfi_signal_frame => cannot tag signal frame FDEs
  - .cfi_escape => treated as black box atm, skipped altogether
- NOTE 2 - CTF Format representation is designed for the most common case:
  - aarch64 return register: LR
  - Only CFA, FP, RA can be recovered
    - X86_64: RA = cfa - 8
    - aarch64: RA = LR
- Short Answer: No, but its close.
- **Q: How much impact does this have for the usecase of reliable stacktraces in the Linux kernel?**

CTF Frame format – key ideas

- One self-sufficient FRE per PC (similar to ORC) to recover CFA, FP, RA. Helps unwind fast (less CPU intensive unwind)
- readelf --ctf-frame=.ctf_frame <OBJ>

func idx [20]: pc = 0x401636, size = 322 bytes

STARTPC	CFA	FP	RA
0000000000401636	sp+8	u	u
0000000000401637	sp+16	c-16	u
000000000040163a	fp+16	c-16	u
0000000000401777	sp+8	c-16	u