Unwinding on arm64

Linux Plumbers Conference 2021

Mark Rutland <mark.rutland@arm.com>

Arm64 calling convention

- Linux uses the **AAPCS64** calling convention
  - AKA “Procedure Call Standard for the Arm® 64-bit Architecture (AArch64)”
  - https://github.com/ARM-software/abi-aa

- Function calls made with `BL <label>` or `BLR <reg>`
  - Places return address in **Link Register** (`LR`, AKA `X30`)
  - `RET` returns to address in `LR` (or `RET <reg>` uses arbitrary GPR)

- Callee responsible for saving/restoring return address ***as necessary***
  - Adds to a linked list of **frame records** pointed to by **Frame Pointer** (`FP`, AKA `X29`)
  - Compiler decides if/when/where/how to do so

- Contents of `LR` and `FP` are defined at **function-call boundaries**
  - Change at arbitrary points within a function body
    - Unwinding across **exception boundaries** requires metadata and/or restrictions on codegen
    - We over-estimate today (reporting both LR and FP)

Arm64 code example

ARMv8.0-A baseline

```
unsigned long bar(void);

unsigned long foo(void)
{
        return bar() + 1;
}
```

```
<foo>:
        stp     x29, x30, [sp, #-16]!
        mov     x29, sp
        bl      <bar>
        add     x0, x0, #0x1
        ldp     x29, x30, [sp], #16
        ret
```

Arm64 code example

ARMv8.0-A baseline + shrink-wrapping

```
unsigned long bar(void);

unsigned long foo(long a)
{
        if (a)
                return a;

        return bar() + 1;
}
```

```
<foo>:
        cbz     x0, .L1
        ret
.L1:
        stp     x29, x30, [sp, #-16]!
        mov     x29, sp
        bl      <bar>
        add     x0, x0, #0x1
        ldp     x29, x30, [sp], #16
        ret
```

Arm64 code example

With BTI + PAUTH + patchable-function-entry + shrink-wrapping

```
unsigned long bar(void);

unsigned long foo(long a)
{
        if (a)
                return a;

        return bar() + 1;
}
```

```
<foo>:
        bti     c
        nop
        nop
        cbz     x0, .L1
        ret
.L1:
        paciasp
        stp     x29, x30, [sp, #-16]!
        mov     x29, sp
        bl      <bar>
        add     x0, x0, #0x1
        ldp     x29, x30, [sp], #16
        autiasp
        ret
```

arm

Thank You
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה

Arm64 basics

Registers & branch instructions

Registers

- `X0` … `X30` – General Purpose Registers
  - `X29` (`FP`) – Frame Pointer
  - `X30` (`LR`) – Link Register
- `SP` – Stack Pointer
- `PC` – Program Counter

Unconditional branch instructions

- `B <label>`
  - Branch to label (+/-128MiB)
- `BR <reg>`
  - Branch to address in reg (64-bits)
- `BL <label>`
  - Place return address in Link Register (`LR`)
  - Branch to label (+/-128MiB)
- `BLR <reg>`
  - Place return address in Link Register (`LR`)
  - Branch to address in reg (64-bits)
- `RET` / `RET <reg>`
  - Branch to address in reg (`LR` by default)