eBPF support in the GNU Toolchain

Jose E. Marchesi

Oracle Inc.

LPC 2019
The Project

- **Phase 1:** add eBPF target to the toolchain
  - `bpf-unknown-none`
  - `binutils (upstream since May 2019)`
  - `GCC (upstream since September 2019)`

- **Phase 2:** make the generated programs palatable for the kernel loaders and verifier, and **keep it that way**.

- **Phase 3:** provide development goodies for eBPF developers
  - GNU simulator
  - GDB
  - ...

Some Characteristics of the Port

- **Very** peculiar compilation target (fun! :D).
- Kernel helpers are implemented as compiler builtins.
- GCC does not support inlined asm transliteration => CO-RE is a must!
- `bpf-helpers.h`
- `-mkernel={4.0,4,1,...,5.2,latest}`
- `-mbig-endian -mlittle-endian`
- `-mframe-size=BYTES`
Try to support as much C as possible

e.g. GCC uses %r9 as the stack pointer to implement `alloca` and VLAs.

```c
int foo (int a, int b)
{
    char[a] array;
    return array[3];
}
```

```
.text
.align 4
.global foo

foo:
    stxdw [%fp+-8],%r9
    mov %r9,%fp
    add %r9,-48
    ...
    add %r1,7
    and %r1,-8
    sub %r9,%r1
    ldxb %r0,[%r9+3]
    ldxdw %r9,[%fp+-8]
    exit
```
GCC - Testing

=== bpf Summary ===

# of expected passes  230

=== c-torture Summary ===

# of expected passes  11743
# of unresolved testcases  28
# of unsupported tests  1482

=== gcc-dg Summary ===

# of expected passes  23062
# of unexpected failures  819
# of unexpected successes  3
# of expected failures  267
# of unresolved testcases  559
# of unsupported tests  649
Question: the name of The Thing

• cBPF ⇒ eBPF ⇒ BPF.
• The port uses:
  • ebpf in documentation and displays.
  • bpf in symbols, options, etc.
• Should I stop using the “ebpf” term? If so, the sooner the better.
**RFC - xbpf**

- Experimental BPF (or some other name, I don’t care)
- `-mxbpf`

**Lifted restrictions:**
- Stack frame size (with an 64Kb upper limit.)
- Indirect call instruction (`callx %reg` in llvm, use same encoding)
- Passing arguments on the stack (`%fp relative addressing in callee.`)
- Stack traces.
- Signed division instruction

**Purposes:**
- Compiler testing
- Debugging of eBPF programs: backtraces.
- Explore the impact of lifting restrictions, beforehand.
- Leverage ELF linking capabilities more?

**Wanna do it in llvm?**
What constitutes a safe eBPF program?
Invalid eBPF programs should be detected as soon as possible in the development process.
The kernel verifier is getting more and more complex and sophisticated.
We want to avoid having to replicate and maintain that logic in the simulator and, partially, in the compiler.
Program, library, something else?
• What constitutes a valid eBPF ELF program?
• Currently:
  • What the llvm backend produces.
  • What the kernel loaders (bpflib, bpf_load.c) implement.
• We need a documented ABI for compiled eBPF:
  • Relocations.
  • Standard section names.
  • ...
Coordination

- More players producing/consuming compiled eBPF:
  - llvm
  - gcc
  - kernel
  - dtrace
  - bpftrace
- We need to be in the loop.
- We want to contribute to the design.
Discussion