`thorin` and other debuggers

Ajay Tatachar

GLUG

2021

thorin?

1. A debugger for C programs
2. Written in Rust and C
3. For x86_64 Linux and macOS
4. Named after this guy:

Figure 1.0: Thorin Oakenshield

Demo!

thorin

1. Like GDB but much simpler
2. Reads DWARF-formatted debug info

How do debuggers work?

Static analysis:

1. DWARF debug info
2. Source code
3. Other stuff in object files

Dynamic program interaction:

1. Read the register state and memory of a running program
2. Inject code to set breakpoints
3. Step through instructions, add hooks to syscalls, etc.

DWARF

```
fig7.c:
        1:  int a;
        2:  void foo()
        3:  {
        4:      register int b;
        5:      int c;
        6:  }

<1>:    DW_TAG_subprogram
          DW_AT_name = foo
<2>:    DW_TAG_variable
          DW_AT_name = b
          DW_AT_type = <4>
          DW_AT_location = (DW_OP_reg0)
<3>:    DW_TAG_variable
          DW_AT_name = c
          DW_AT_type = <4>
          DW_AT_location =
                (DW_OP_fbreg: -12)
<4>:    DW_TAG_base_type
          DW_AT_name = int
          DW_AT_byte_size = 4
          DW_AT_encoding = signed
<5>:    DW_TAG_variable
          DW_AT_name = a
          DW_AT_type = <4>
          DW_AT_external = 1
          DW_AT_location = (DW_OP_addr: 0)
```

Figure 7. DWARF description of variables *a*, *b*, and *c*.

1. A big tree of compilation units, functions and blocks
2. Each “scope” has an associated instruction pointer range
3. Variable nodes are associated with scopes and encode location as absolute addresses or base-pointer offsets.
4. User-defined types are associated with base types or a list of members

Interacting with running programs

Using the `ptrace` syscall:

NAME top

ptrace - process trace

SYNOPSIS top

```
#include <sys/ptrace.h>

long ptrace(enum __ptrace_request request, pid_t pid,
            void *addr, void *data);
```

DESCRIPTION top

The **ptrace**() system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers. It is primarily used to implement breakpoint debugging and system call tracing.

How does thorin work?

1. Read DWARF info from object file
2. Construct a scope tree with IP-ranges and associated variables and types
3. Run the program and `ptrace` it
4. Wait for a segfault/abort/etc.
5. Figure out where we are in the scope tree and what variables we can access based on registers

Misc. things

1. I had to disable ASLR when starting child processes
2. Rust is difficult and not always worth it
   2.1 But pattern matching and macros are great
3. gdb and lldb are very powerful; this is just a small toy
   3.1 but it was a great learning experience in systems programming