LLDB in FreeBSD

Ed Maste
FreeBSD Vendor Summit
November 2013

FreeBSD needs a new debugger

- FreeBSD base has GDB 6.1.1 (June 2004)
- Major shortcomings
  - poor C++ support
  - usability issues for threaded inferiors
  - limited scripting
  - poor performance
  - ...
- FreeBSD project policy precludes GPLv3
- Last GPLv2 GDB is 6.6, December 2006
  - only marginally better than 6.1.1

LLDB History

- Debugger in the LLVM family of projects
- Originated within Apple
- Released as open source in June 2010
- ~ 650 KLOC (GDB is ~3M)
- 36 contributors last 12 months
  - up from 17, previous 12 months
- 22 contributors last month
  - 4 new
- Apple (16), Intel (7), FreeBSD (1), Debian (1), Valve Software (1), NetBSD (1), Mentor (1), Individuals and unknown (22)

LLDB Benefits

- Speed
  - Multi-threaded, leverages performant LLVM classes
- Efficiency
  - Minimize memory footprint - lazy and partial evaluation
- Accuracy
  - Improved ability to set breakpoints, expression parsing
  - Breakpoints are always symbolic - reparsed after .so loading

LLDB Extensibility and Reusability

- Classes for process, thread, dynamic loader, object files, object containers, symbols, disassembly, instruction emulation
- lldb commandline, XCode, Python front ends

```
>>> import lldb
```

- built-in python interpreter for scripting
    - easily extended for other languages

LLDB Syntax

GDB

```
% gdb a.out
(gdb) break main
Breakpoint 1 at 0x100000f33: file main.c, line 4
(gdb) run
```

LLDB

```
% lldb a.out
(lldb) breakpoint set --name main
Breakpoint created: 1: name = 'main', locations = 1
(lldb) process launch
```

LLDB Syntax

GDB	LLDB
	`(lldb) process launch`
`(gdb) run`	`(lldb) run`
`(gdb) r`	`(lldb) r`
	`(lldb) thread step-in`
`(gdb) step`	`(lldb) step`
`(gdb) s`	`(lldb) s`
`(gdb) info break`	`(lldb) breakpoint list`
	`(lldb) br l`
`(gdb) info args`	`(lldb) frame variable`
and	`(lldb) fr v`
`(gdb) info locals`	

Demo

LLDB Status in FreeBSD

- Testsuite
  - 260 tests run, ~ no failures without associated PR
  - 17 open PRs
- Targets
  - amd64
  - i386 code in tree, does not work
  - MIPS in development, partially committed
  - ARM supported in LLDB core, not Linux / FreeBSD
  - Others unaware of any plan

LLDB Status in FreeBSD

- Userland core files
  - “Just works” for 9.2+ and HEAD cores
  - for some value of “Just works”
  - further testing needed
- Userland live debugging (ptrace)
  - Process launch, process attach by pid
  - Process attach by name
  - Breakpoints
  - Watchpoints
  - Threads (in development)

LLDB Status in FreeBSD

- Kernel core files
  - Unimplemented
  - straightforward - follow userland ELF core example
  - or modify kernel / savecore to produce ELF dumps
- Kernel live debugging
  - Unimplemented
  - gdb remote protocol (serial stub)
  - /dev/mem

LLDB Status in FreeBSD

- Remote debugging - GDB protocol
  - Need to enable / test on Linux & FreeBSD
- Remote debugging - debugserver
  - Unimplemented, Intel doing infrastructure work
- Cross debugging
  - Cross-arch and cross-OS
  - Should “just work”
  - Fails due to some assumptions in source, but not difficult

Short term

- Source in contrib/llvm/tools/lldb
- FreeBSD build infrastructure committed
- Source in 10.0, currently not built by default
- WITH_LLDB= in src.conf
- Testing

Medium term

- amd64 thread support for ptrace
- watchpoints
- MIPS host and target
- test suite failures

Longer term

- ARM support
- Kernel debug
- Remote debug