Introducing LLDB for Linux on Arm and AArch64

Omair Javaid
Agenda

- Brief introduction and history behind LLDB
- Status of LLDB on Linux and Android
- Linaro’s contributions in LLDB
- An overview of LLDB Architecture
- More detail on working and architecture
- Overview of basic functionality and commands
- Overview of expressions and data formatters.
LLDB - Introduction

● Debugger component of LLVM project.
● A modular, high-performance source-level debugger written in C++
● Re-uses LLVM/Clang code JIT/IR for expression parsing, disassembly etc.
● Provides a C++ Debugger API which can be used by various clients
● Very permissive BSD-like license.
LLDB - Introduction (cont…)

- **Supported Host Platforms**
  - OS X, Linux/Android, FreeBSD, NetBSD, and Windows

- **Supported Target Architectures**
  - i386/x86_64, Arm/AArch64, MIPS/MIPS64, IBM s390

- **Supported Languages**
  - Fully support C, C++ and Objective-C while SWIFT and GoLang (under development)
LLDB - History

- Apple stopped shipping GDB after version 6.3.5
- Last GPLv2 based GDB release.
- Announced LLDB at WWDC2010.
- Extensively under development for various use-cases.
- Active code contributors are Apple and Google.
- Default debugger for OSX, Xcode IDE, Android Studio.
- Also adopted by FreeBSD as their default debugger.
LLDB - Linaro’s contributions

- LLDB port for ARMv7 architecture.
  - Register Context and Basic Debugging support.
  - Support for SysV ARM/Linux ABI.
  - Support for Arm Hardware Watchpoints
  - Support for Arm Single Byte-Selection Watchpoints.
  - Various bug fixes and improvements.
  - Support for Arm Hardware Breakpoints (NEW)
LLDB - Linaro’s contributions (cont…)

● LLDB port for ARMv8 architecture.
  ○ Support for SysV AArch64/Linux ABI.
  ○ Support for AArch64 Hardware Watchpoints
  ○ Various bug fixes and improvements.
  ○ Support for AArch64 Hardware Breakpoints (NEW)

● Maintenance and Testing
  ○ Buildbot development and maintenance
  ○ Tester bring up and validation
  ○ Buildbot Failure Triage
**LLDB - Features Status ARM/AArch64**

- LLDB features stable on ARM and AArch64
- For more details visit: [http://lldb.llvm.org/status.html](http://lldb.llvm.org/status.html)

<table>
<thead>
<tr>
<th></th>
<th>Process control</th>
<th>Thread Control</th>
<th>Breakpoints</th>
<th>Register Inspection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARM</strong></td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td><strong>AArch64</strong></td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
</tbody>
</table>
LLDB - Arm/AArch64 Test Devices

- Buildbots contributed by google test various Android devices
  - Link: lab.llvm.org:8011/builders/lldb-x86_64-ubuntu-14.04-android

- Linaro keeps track of LLDB on Linux platform for ARM and AArch64 targets.
  - Buildbot is still not public but we plan to do so.

<table>
<thead>
<tr>
<th>Platform/Architecture</th>
<th>ARMv7</th>
<th>ARMv8 - AArch64</th>
<th>ARMv8 - AArch32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android M</td>
<td>Nexus 5</td>
<td>Nexus 5x</td>
<td>Nexus 5x</td>
</tr>
<tr>
<td>Ubuntu Linux - Xenial</td>
<td>Raspberry Pi2</td>
<td>Pine64</td>
<td>Pine64</td>
</tr>
</tbody>
</table>
LLDB Architecture

- LLDB Command line
- LLDB-MI Driver
- IDEs (e.g., Xcode...)
- LLDB Python Module

**Platform**
- Linux
- Android
- MacOSX
- NetBSD
- Windows

**Target**
- Unwinder
- Instruction Emulation
- Dynamic Loader
- Symbols - DWARF, PDB
- Object File - ELF, JIT, MACH-O, PECOFF

**Host Abstraction Layer**
- Commands
- Interpreter
- Core
- Utility Classes
- Data Formatters
- Breakpoints

**Expressions Parser** - Clang, Go

**Target**
- Process
- Thread
- Run-Control
- Stack Frame
- Variables

**Host Abstraction Layer**
- Expressions
- ABI - SysV and MacOSX (Arm, AArch64, i386, etc...)

**Disassembler** - LLVM

**Data Formatters**

**Breakpoints**

**GDB**
- Registers
- Memory
- Breakpoints

**Utility Classes**

**Remote Platform**
- System APIs

**Platform**
- Linux, Android, MacOSX, NetBSD, Windows, etc...

**Debug APIs**

**Core**

**IDEs** (e.g., Xcode...)

**Interpreter**
LLDB Debugger API

- A C++ shared library with object-oriented interface.
- **LLDB.framework** on MacOS X and **lldb.so** on linux.
- Used by
  - lldb - The LLDB Debugger command line
  - lldb-mi and lldbmi2 - Machine Interface (MI) drivers
  - XCode and Android Studio - IDEs with graphical front-ends.
  - lldb Python module - LLDB API exposed through python script bindings using SWIG.
**lldb-server - Platform Mode**

- File transfer to remote target.
- Spawns gdbserver on target.
- Runs shell commands on target.
- List remote processes.
- Allows single or multiple parallel debug sessions.
lldb-server - Platform Mode (Single Session)

- Spawn lldb-server on remote-android target in platform mode for single session.

- Select remote-android platform and connect.

- Try connecting another lldb remote-android platform session.
lldb-server - Platform Mode (Multiple Sessions)

Select remote-android platform and connect.

Connect another remote-android platform session.

lldb-server platform -- server mode

lldb-server multiple sessions connected
lldb-server - GDB Server Mode

- Run as a gdb-remote-stub.
- Communicates with LLDB over RSP protocol.
- On Mac OSX and iOS, the remote-gdb functionality is in debugserver binary.
lldb-server - GDB Server Mode

Start lldb-server in gdbserver mode

```
omair@omAlien:/work/lldb-test/build/host/bin$ ./lldb-server gdbserver *
lldb-server-local_buildConnection established.
Launched '/home/omair/work/lldb-dev/hwbreak/x86_64.out' as process 26892...
```

Connect to gdb remote stub and start a debugging session

```
omair@omAlien:/work/lldb-test/build/host/bin$ ./lldb /home/omair/work/lldb-dev/hwbreak/x86_64.out
(lldb) target create "/home/omair/work/lldb-dev/hwbreak/x86_64.out"
Current executable set to "/home/omair/work/lldb-dev/hwbreak/x86_64.out" (x86_64).
(lldb) process connect connect://localhost:5432
(lldb) b main
Breakpoint 1: where = x86_64.out:main + 8 at test.c:14, address = 0x000000000040054b
(lldb) run
Process 26892 launched: '/home/omair/work/lldb-dev/hwbreak/x86_64.out' (x86_64)
Process 26892 stopped
* thread #1, name = 'x86_64.out', stop reason = breakpoint 1.1
  frame #0: 0x000000000040054b x86_64.out:main at test.c:14
    11    int main()
    12    {
    13      int temp = 0;
    14      printf("Read Op: Value of temp is %i",temp);
```

lldb - Debuggee (or inferior) context on host system

- **Platform**
  - Local or remote platform functions.

- **Process GDB Remote Client**
  - Let lldb host talks to lldb-server running in gdb remote mode.
  - Performs debug operations via GDB Remote Serial Protocol (RSP)

- **Target**
  - Implements debugee (inferior) management functions like
    - Process Control
    - Thread Control
    - Registers and Memory
    - Stack Frames and Variables
    - Breakpoints and Watchpoints
    - Module loading
lldb - target create and process launch

# Select target platform
platform select <name>

# Connect to platform
platform connect <url>

# Load file and symbols
target create <file-name>

# Set breakpoint at main
breakpoint set -b main

# Start remote inferior
process launch
# lldb - Target and Process commands

<table>
<thead>
<tr>
<th>Target</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create / Delete</td>
<td><strong>Connect</strong> - to remote debug.</td>
</tr>
<tr>
<td>List and Select</td>
<td><strong>Launch</strong> - current target executable</td>
</tr>
<tr>
<td>Symbols</td>
<td><strong>Attach/Detach</strong> - to a process</td>
</tr>
<tr>
<td>Variables</td>
<td><strong>Continue/Interrupt/Kill</strong> - current target process.</td>
</tr>
<tr>
<td></td>
<td><strong>Load/Unload</strong> - shared library</td>
</tr>
</tbody>
</table>
# Ildb - Thread, Frame, Memory & Registers commands

<table>
<thead>
<tr>
<th>Thread</th>
<th>Frame, Memory and Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stepping</strong></td>
<td>• Frame</td>
</tr>
<tr>
<td>• Step-in, step-out and step-over</td>
<td>• Variable</td>
</tr>
<tr>
<td>• step-inst and step-inst-over</td>
<td>• Select</td>
</tr>
<tr>
<td><strong>Select</strong> - Select current thread.</td>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td><strong>Backtrace</strong> - Thread call stacks.</td>
<td>• Read</td>
</tr>
<tr>
<td><strong>Continue</strong> - Continue one or all threads.</td>
<td>• Write</td>
</tr>
<tr>
<td><strong>info/list</strong> - Information about threads in current process.</td>
<td>• Registers</td>
</tr>
<tr>
<td></td>
<td>• register read &lt;reg-name&gt;</td>
</tr>
<tr>
<td></td>
<td>• register write &lt;reg-name&gt; &lt;data&gt;</td>
</tr>
</tbody>
</table>
lldb - Break/Watchpoints & Disassembly commands

**breakpoint/watchpoint**

- **set <options>**
  - Set a breakpoint based on options.
- **clear <line-no or filename>**
- **command <add, delete or list>**
  - Run command or script when breakpoint is hit.
- **delete**
  - All or specified breakpoint is deleted.
- **enable/disable**
  - All or specified breakpoint.
- **read/write**
  - Read from or write breakpoints to file.

**disassemble**

- **(-f) --frame**
  - from the start of the current frame
- **(-m) --mixed**
  - mixed source and assembly display.
- **(-n) --name <function-name>**
  - entire contents of the given function.
- **(-p) --pc**
  - around the current pc.
- **(-a) --address <address>**
  - function containing address.
- **--start-address / --end-address**
lldb - command alias <alias> <command> <args>

Some Built-in Aliases

File <file>
- target create <file>

b main
- breakpoint set -b main

run
- process launch
lldb - Using .lldbinit file

.lldbinit file on top right of this slide is running at startup:

- Platform select and connect
- Load object file and symbols
- Set breakpoint at main
- Start inferior process
- Run to main.

LLDB command-line output on bottom right shows lldb state after we start it with .lldbinit file in user’s home directory.
LLDB - Brief description of components

● **ABI**
  ○ Executes JITTed function calls in process being debugged.
  ○ Call frame calculation based on target calling convention.

● **Unwinder**
  ○ Implements stack unwinding features.

● **Instruction Emulation**
  ○ Helps with stack frame constructions.
  ○ Helps single step by emulating execution paths.

● **Expression Parser**
  ○ A modified DWARF expression parser
  ○ Clang front-end latest c++ support
LLDB - Brief description of components

- **Disassembler**
  - LLVM disassembler

- **Dynamic Loader**
  - Helps detect dynamically loaded modules

- **Object File**
  - Handles object files management.

- **Symbols**
  - Handles debug symbols management

- **Host Abstraction Layer**
- **Core and Utility Classes**
- **Interpreter and Commands**
- **Data Formatters**
- **Breakpoint Utilities**
LLDB - unique_ptr variables with data formatters

Without deleter
- frame variable nup
- frame variable iup
- frame variable sup

With deleter
- frame variable ndp
- frame variable idp
- frame variable sdp
LLDB - `shared_ptr` & `weak_ptr` with data formatters

--ptr-depth 1

- Dereference pointer once

**std::shared_ptr**

- frame variable `nsp`
- frame variable `isp`
- frame variable `ssp`

**std::weak_ptr**

- frame variable `nwp`
- frame variable `iwp`
- frame variable `swp`
LLDB - `std::map` variables with data formatters

```
std::map<std::string, int>  
  • frame variable `si`

std::map<int, std::string>  
  • frame variable `is`

std::map<std::string, std::string>  
  • frame variable `ss`
```
LLDB - Evaluate **expressions** using clang parser

- Compile “hello world” code
- Launch it using LLDB.
- Stop in main before return.
- Now our expressions will have a context to work with.
- Let's try some example expressions.
LLDB - Evaluate expressions using clang parser

- Let's try evaluating a few simple expressions.
- Create integer type local variables i and j
- Let see if we can call getpid
- Evaluate getpid() + i + j
- Evaluate getpid() + i & j
- Evaluate getpid() + i ^ j
- Evaluate getpid() + i << j
LLDB - Evaluate **expressions** using clang parser

- Create context as we did in previous slide.
- Try evaluating a `for` loop expression.
LLDB - Evaluate expressions using clang parser

- Create a c++ lambda expression named add.
- Specify return value auto
- Evaluate lambda expression on our dummy context.
- Cool we have them working.
- Lets try a more complex lambda with auto arguments
Conclusion

- Start using LLDB.
- It's more stable than ever before.
- Wanna go an extra mile? Feel free to contribute.
- Topics for future
  - Using LLDB Python API and built-in Python interpreter
  - How-To on more advanced expressions
  - How to use more advanced and Python based data formatters
Questions

Email: omair.javaid@linaro.org
IRC: omjavaid @ freenode and oftc
References

- LLDB (Architecture, GDB to LLDB commands, Doxygen)
- LLVM LAB - LLDB Android Buildbot
- Linaro LAB - LLDB Linux Buildbot (Under Development)
- LLDB on FreeBSD
Thank You

#BUD17

For further information: www.linaro.org
BUD17 keynotes and videos on: connect.linaro.org