Porting LLDB

Hafiz Abid Qadeer

mentor.com/embedded
Outline

- Introduction
- A Typical Debug Session
- LLDB Architecture
- Porting LLDB
- Roles of Plugins
- Questions
Introduction

- LLVM Debugger
- Open Sourced in 2010
- Written in C++
- Small but helpful community

Status

- Default debugger for xcode
- Linux and FreeBSD ports are working
- Package available in many distros.
- Windows support is coming online
Example Session

> lldb hello
(lldb) target create "hello"
Current executable set to 'hello' (x86_64).

(lldb) b main
Breakpoint 1: where = hello`main + 4 at hello.c:5, address = 0x000000000400531

(lldb) r
Process 7093 launching
Process 7093 launched: '/home/abidh/demos/hello' (x86_64)
Process 7093 stopped
* thread #1: tid = 7093, 0x0000000000400531 hello`main + 4 at hello.c:5, name = 'hello', stop reason = breakpoint 1.1
   frame #0: 0x0000000000400531 hello`main + 4 at hello.c:5
       2       int counter = 10;
       3       int main(void)
       4       {
       -> 5           printf("Hello World!\n");
       6               counter++;
       7               return 0;
       8           }

(lldb) p counter
(int) $0 = 10

(lldb) bt
* thread #1: tid = 7093, 0x0000000000400531 hello`main + 4 at hello.c:5, name = 'hello', stop reason = breakpoint 1.1
  * frame #0: 0x0000000000400531 hello`main + 4 at hello.c:5
     frame #1: 0x00007fffffff7a35ec5 libc.so.6`__libc_start_main + 245
     frame #2: 0x000000000400469 hello
So a debugger needs to …

- Read the object file
- Find symbols information if available
- Provide execution control
  - run, stop, break
- Provide visibility into the program
  - Variables and expressions
  - Register, memory, disassembly
  - Target function call
  - Dynamic Objects
  - OS Awareness
Architecture

- Provides a C++ API which can be used by various clients

```plaintext
Architecture

Process
  - Gdb Remote

Disassembly
  - LLVM

Symbols
  - Dwarf

Object Files
  - Elf
  - Mach-o

API

lldb

lldb-mi

Core
```
Life cycle of a Plugin

- PluginManager
- Initialize
  - Register itself with the PluginManager
- Find a Plugin to do something
  - Call PluginManager::FindPlugin with enough information
- CreateInstance
- Overload required functions
- Terminate
  - Unregister itself from PluginManager
So a debugger needs to ...

- **Read the object file**
- Find symbols information if available
- Provide execution control
  - run, stop, break
- Provide visibility into the program
  - Variables and expressions
  - Register, memory, disassembly
  - Target function call
  - Dynamic Objects
  - OS Awareness
ObjectFile

- Chances are that your file is already supported
  - Elf, mach-o, PE-coff

- Creation
  - If able to handle a given object file

- CreateSections
  - Figure out how many sections are in this object file

- ReadSectionData
  - Read data from a given section

- Other useful information
  - ByteOrder, AddressSize, Architecture
  - EntryAddress
So a debugger needs to ...

- **Read the object file**
- **Find symbols information if available**
- **Provide execution control**
  - run, stop, break
- **Provide visibility into the program**
  - Variables and expressions
  - Register, memory, disassembly
  - Target function call
  - Dynamic Objects
  - OS Awareness
SymbolFile & SymbolVendor

- SymbolFile
  - Dwarf

- SymbolVendor
  - Controls the process of finding symbols
  - Separate or multiple symbols file
Example Session

(lldb) target module dump line-table hello.c
Line table for /home/abidh/demos/hello.c in `hello
0x0000000000040052d: /home/abidh/demos/hello.c:4
0x00000000000400531: /home/abidh/demos/hello.c:5
0x00000000000400540: /home/abidh/demos/hello.c:6
0x0000000000040054f: /home/abidh/demos/hello.c:7
0x00000000000400554: /home/abidh/demos/hello.c:8
0x00000000000400556: /home/abidh/demos/hello.c:8

(lldb) target module dump sections
Dumping sections for 3 modules.
Sections for `/home/abidh/demos/hello' (x86_64):

<table>
<thead>
<tr>
<th>SectID</th>
<th>Type</th>
<th>Load Address</th>
<th>File Off.</th>
<th>File Size</th>
<th>Flags</th>
<th>Section Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>regular</td>
<td>0x00000000 0x00000000 0x00000000</td>
<td>hello.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000002</td>
<td>regular</td>
<td>[0x00000000000400238-0x00000000000400254]</td>
<td>0x00000238 0x0000001c 0x00000002</td>
<td>hello..interp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000003</td>
<td>regular</td>
<td>[0x00000000000400254-0x00000000000400274]</td>
<td>0x00000254 0x00000020 0x00000002</td>
<td>hello..note.ABI-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td>regular</td>
<td>[0x00000000000400274-0x00000000000400298]</td>
<td>0x00000274 0x00000024 0x00000002</td>
<td>hello..note.gnu.build-id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
So a debugger needs to ...

- Read the object file
- Find symbols information if available
- **Provide execution control**
  - run, stop, break
- Provide visibility into the program
  - Variables and expressions
  - Register, memory, disassembly
  - Target function call
  - Dynamic Objects
  - OS Awareness
Process Plugin

- What already there
  - Linux
  - FreeBSD
  - Window
  - Gdb-remote
  - Elf-core
  - ...

- If your target is not one of these
  - Is remote debugging an option for you
Process Plugin

- Selection

- Major Components
  - Process
    - Attach, connect
    - Read/write memory
    - Breakpoint
    - Run/stop
  - Thread
    - Stop Reason
    - Target specific step operation
    - GetUnwinder
  - Register Access
    - Recognize PC, SP, FP
Process Plugin

LLDB

Process Plugin

Asynch Events Thread

Target
Process Plugin

- WillLaunch, DoLaunch
- DoConnect
- WillResume, DoResume
- WillHalt, DoHalt
- EnableBreakpointSite, DisableBreakpointSite
- DoReadMemory, DoWriteMemory
- DoAllocateMemory
- UpdateThreadList
- GetStatus
- DoDestroy
So a debugger needs to ...

- Read the object file
- Find symbols information if available
- Provide execution control
  - run, stop, break
- Provide visibility into the program
  - Variables and expressions
  - Register, memory, disassembly
  - Target function call
  - Dynamic Objects
  - OS Awareness
Program Visibility

- Register
- Memory
  - Done by Process Plugin
- Disassembler
  - LLVM disassembler
- Stack
  - Good debug information
  - UnwindAssembly Plugin
- Expressions
  - Clang integration
Program Visibility

- **Target Function Call**
  - ABI Plugin

- **DynamicLoader**
  - Shared Objects
  - Step over function trampoline
  - TLS

- **OperatingSystem Plugin**
  - OS awareness
Misc Stuff

- ArchSpec
  - Provides target description
- Elf header
  - Handles your architecture
- Thread::Unwinder
  - Select default unwinder
- Add your Plugins in the build
- Call your Initialize/Terminate functions
- Test cases
  - Add test cases specific to your architectures
Hurray ...

- Read the object file
- Find symbols information if available
- Provide execution control
  - run, stop, break
- Provide visibility into the program
  - Variables and expressions
  - Register, memory, disassembly
  - Target function call
  - Dynamic Objects
  - OS Awareness
QUESTIONS