An LLVM developer setup

Modern C++ development tools

Arnaud de Grandmaison, FOSDEM 2016

Foreword

- Goals :
  - Provide an overview of available tools for C++ development
  - Make you aware these exists.
    - That's the first step to start using them :)
- Targeted audience: non LLVM developers
- I did not write those tools, all credits goes to their authors

Agenda

- Overview of the LLVM project
- LLVM development setup
- Available tools for developers

Agenda

- Overview of the LLVM project
- LLVM development setup
- Available tools for developers

The LLVM project

- http://www.llvm.org
- No longer an acronym !
- Can refer to both the umbrella project and the core libraries.
- A modular collection of reusable components around compilation :
  - LLVM Core : intermediate representation
  - Clang : a compiler
  - lldb : a debugger
  - lld : linker
  - libc++ : a standard library
- BSD style license

LLVM community

- A vibrant community !
- Lots of very different usages of the project
- 2 developers meeting per year:
  - in Europe around March
  - in the US around November
- Regular social events:
  - Cambridge/UK
  - Paris/France
  - Zürich/Switzerland
  - Bay area/US

LLVM

- Core libraries:
  - Intermediate representation (IR)
  - Mid-end optimizers
  - Code generation
  - Machine optimizations
  - Object file support
  - JIT
- Some stats (from openhub) :
  - Mostly written in C++11
  - ~ 1.5MLoC
  - ~ 130 contributors
  - ~ 1200 commits / month
- Provides backends for x86, ARM, AArch64, MIPS, PowerPC, …

Clang

- A C/C++/ObjC compiler
  - Built on top of the LLVM core libraries
  - Provides a collection of reusable (and reused!) components :
    - Libclang, a stable high level C interface to clang
    - Or the C++ clang libraries if full control over the AST is needed
- Some stats (from openhub) :
  - Mostly C++11
  - ~ 1+ M LoC
  - ~ 90 contributors
  - ~ 500 commits / month
- Platforms : Linux, Windows, MacOS, FreeBSD

Other projects

- Lldb :
  - A debugger, built as a set of reusable components
  - Reuse other components, like Clang's parser
  - Platforms : MacOS, iOS, Linux, FreeBSD, Windows
- Libc++ :
  - a new implementation of the C++ standard library, targeting C++11 and beyond
- Lld :
  - A set of modules for creating linker tools
  - Supports ELF, Mach-O and PE/COFF

Agenda

- Overview of the LLVM project
- LLVM development setup
- Available tools for developers

LLVM development

- Builds with itself:) and recent enough versions of gcc
  - decent C++11 support required
- Configuration stage : Cmake (configure being deprecated)
- Build : ninja / make
- Test:
  - Unit tests
  - Testsuite
  - Buildbot setup, running all kinds of test on all kind of platforms

Tips & tricks

- Cmake ≥ 3.4 have good CCache support
  - Use `-DCMAKE_${LANG}_COMPILER_LAUNCHER:...`
- For DEBUG builds, you may want to use shared libs :
  - `-DBUILD_SHARED_LIBS:BOOL=ON`
  - Unless you have a lot of memory
- If you wish to build yourself the tools advertized in this presentation, you'll need llvm, clang, compiler-rt and clang-tools-extra.

LLVM development

- Compilation database :
  - Optionally generated by cmake
  - Contains compile flags for each source file in the project
  - JSON format
  - Used by a number of llvm tools

Agenda

- Overview of the LLVM project
- LLVM development setup
- Available tools for developers

Sanitizers

- Also available with gcc
- Valgrind is a great tool
  - but it is slow
- Sanitizers provide fast and focused runtime checks, inserted by the compiler.
  - Address sanitizer : addressability issues
  - Thread sanitizer : data races & deadlocks
  - Memory sanitizer : uninitialized memory
  - Leak sanitizer : memory leaks

Sanitizers

- When to use them ?

- Always !
- Well, almost…

- As part of the continuous integration testing
  - For example LLVM has builders with the sanitizers on
- When you face a strange bug, and your developer's experience/intuition suggests some class of bugs

Using ASan

- Add `-fsanitize=address` to your compilation flags
- Recompile
- Et voilà !
- Hint:
  - To get a workable output, you probably want to use `-g -fno-omit-frame-pointer`
- Demo

- Asan can also perform some more detailed / expensive checks
  - Those need to be explicitly enabled, either at compile time or with an env variable
  - Read the doc to learn about available checks
- Demo

Fuzzing

- As developers, we of course pay great attention to make sure we covered all cases, exceptional situations, and ill-formed inputs
  - But we fail at it –- let's be honest ;)
  - Consequences can be really bad
    - remember openssl / heartbleed ?
  - Some bad guys are actively trying ill-formed inputs

- Careful programming and code reviews can help
  - But if the domain is not trivial, bugs will slip through
  - And even when it's trivial...

Fuzzing

- Fuzzing is a testing technique to provide random inputs to a program, possibly starting from a corpus of known inputs (i.e. seeds)

- LLVM provides libFuzzer:
  - Intended for in-process coverage-guided testing of other libraries

- Typical workflow:
  - Mix and match different build modes (asan, msan, …) and optimization levels (-O{0,1,2,…})
  - Collect an initial corpus of inputs
  - Run the fuzzer
  - And watch it catch bugs...

Fuzzing

- My piece of advice:
  - Fuzzing is an incredibly efficient technique
  - Do a favour to your project and your users
    - And yourself ultimately
  - Use some fuzz testing, libFuzzer or any other available technology, including your own if you are in specific domain.

Code completion

- Stop using weird heuristics, use a real compiler !

- clang_complete:
  - vim plugin
  - https://github.com/Rip-Rip/clang_complete

- YouCompleteMe
  - https://github.com/Valloric/YouCompleteMe
  - Vim, emacs, sublime text, ... plugin

- Both are libclang based
- Demo

Code formatting

- Formatting :
  - is more than just indentation
  - is similar to what text processing applications like TeX are doing.

- Formatting is important
  - Just like comments ;)
  - We all know about this
  - And it can end up in a *religious* wars

- Formatting is just boring…

clang-format

- Supports formatting C, C++, Java, JavaScript, Objective-C, Protobuf code
- Not based on Clang :(
  - But darn useful !
- VIM & Emacs integration
- Configuration:
  - Can use a predefined style, in a `.clang-format project` file
  - Or just guess from the surrounding code
- Demo

clang-tidy

- Clang-based C++ linter tool (and much more)

- >50 checks
  - Readability, efficiency, correctness, modernize, …
  - Can automatically fix the code in many cases
  - “Easy” to add your own domain specific checks
    - Once you have a fairly good grasp of clang's AST

- Watch the presentation from Manuel Klimek & Daniel Jasper at the US LLVM dev conference :
  https://www.youtube.com/watch?v=dCdOaL3asx8&index=18&list=PL_R5A0lG
  i1AA4Lv2bBFSwhgDaHvvpVU21

- Demo

Thank you !