Google Summer of Code 2024 Proposal

The LLVM Compiler Infrastructure

Support clang plugins on Windows

Thomas Fransham(tfransham@gmail.com**)**
Mentors: Vassil Vassilev, Saleem Abdulrasool

Overview

The Clang compiler is part of the LLVM compiler infrastructure and supports various languages such as C, C++, ObjC and ObjC++. The design of LLVM and Clang allows the compiler to be extended with plugins. A plugin makes it possible to run extra user defined actions during a compilation or add new pragmas and attributes that it can process. Dynamically loaded plugins are supported on unix and darwin, but not on windows because symbols have to be explicitly exported from shared libraries with either __declspec(dllexport) attribute manually added to c++ declarations or by providing a symbol list to the linker.

Project Motivation

Allow the use of clang plugins on windows without having resort to cumbersome workarounds like directly compiling plugin code into a clang binary like what chrome currently does for its Plugins or having to use makeshift solutions to build a list mangled c++ symbol names to pass to the linker for it to export. The latter involves using LLVM's extract_symbols.py script, which extracts and filters the symbol names from all the LLVM static libraries using regex name patterns that can end up missing symbols. Without filtering the clang dll export list symbols can exceed the 64k architectural limit of DLL exports. It also doesn't solve the problem for data symbols that need to be explicitly annotated

with dllimport when referenced by external code outside the DLL for the linker to correctly import them.

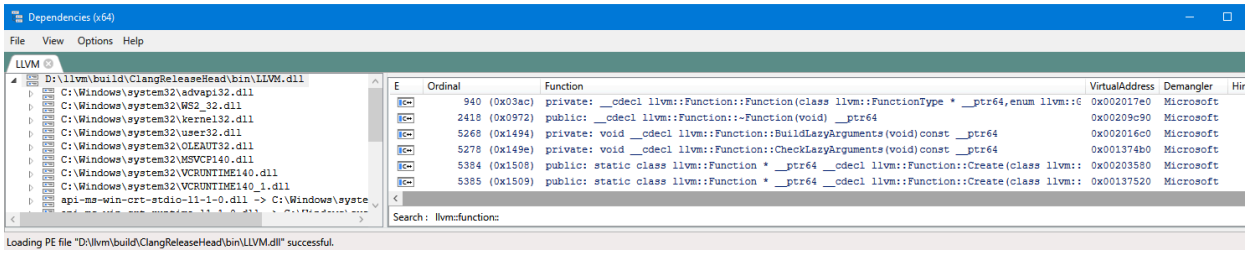
Project Goals

- Allow using the LLVM and Clang C++ API without static linking on windows, many third party tools hit limitations of the C bindings not exposing something and are forced to distribute a custom build of libclang with some extra code linked in to export the APIs they need.
- Increase the ecosystem for windows clang plugins to normal end users who could use prebuilt plugins linked against the official clang binaries created by the LLVM project, instead of it currently being limited to people who build their own custom version clang.
- Shrink the install size of LLVM on windows from gigabytes to 100s of megabytes by not having statically link copies of the same llvm code into clang and all the other llvm tools distributed in official builds.
- Reduce the number of symbols exported by the LLVM shared library on Linux by setting stricter internal symbol visibility and explicitly exporting symbols that are the public headers by reusing the same export macros that will be added to the code for windows.

Personal Motivation

This project excites me because it will allow me to contribute to open source and collaborate with a vibrant community such as the LLVM developers community. The project is intellectually challenging and will develop my compiler engineering skills to the next level but at the same time be of enormous benefit to the windows users.

I have already started some initial exploration of some of the more complex manual code changes required for exporting some LLVM's classes and familiarized myself with the Clang AST and tooling API used by the idt clang tool.



A work in progress demonstration of selective exporting of symbols for Windows

Implementation Details

The code from Tom Stellard's existing [llvm pull request] that starts to annotate things with export macros can be used as base to start from since it has many of the required manual export fixes already.

To automate adding dll export macros classes and functions I will use and update the Clang tool created by Saleem Abdulrasool and improved by Tom Stellard to automate adding dllexport macros to classes and non class functions in public include headers. Templates declared extern also need to be annotated export macros as well, but it needs tobe done slightly differently on Linux vs Windows where the dllexport macro needs tobe on the template instantiation.

The main issue to solve is when a dll export specifier is put on a class is it will instantiate all class members this can trigger compile errors from members not currently called by llvm code, common examples are auto generated copy constructor, assignment operators for class with non movable members like `std::unique_ptr`, the most common fix is manually declaring the special members as deleted, an alternative idea proposed in previous discussion is adding a non movable utility class was for classes to derive from similar to ones proposed for the c++ std library.

Another problem to handle is members of a templated base can trigger compile errors if instantiated for a particular parent class that does not implement members to support a particular feature, this is not normally an issue for normal llvm code that doesn't use API when it's not supported, but putting dllexport on a class will.

Timeline

Community Bonding Period	
May 1 - May 26	Engage with the community. Establish regular meetings with the mentors. Set up the development environment. Set up external tools as discussed here and here. Start an RFC with the proposed approach. Send the first trivial patch.
Coding period begins	
Week 1 27.05.2024-02.06.2024	Start exporting information from the lowest level of the llvm build graph. **Deliverable**: CMake build system can create a llvm DLL with exported symbols from libLLVMSupport without the need for an exported symbol list .def file being passed to the linker.
Week 2 03.06.2024-09.06.2024	Experiment with the build configurations to make better use of the symbol exports. **Deliverable:** Demonstrate reduced artifact binary sizes due to the reduction of exported symbols.
Week 3-4 10.06.2024-16.06.2024 17.06.2024-23.06.2024	Use the developed idt tool combined with manual fixes to all llvm public API headers api should be exported. Potentially develop patches for the idt tool to automate the manual work. **Deliverable:** llc is usable on windows when linked with llvm as shared library, RFC on the more complex fixes required for using dll export on some classes
Week 5	Apply the work at scale. Try to build the llvm project with the new export infrastructure. Resolve bugs in the export lists and the idt tool.

24.06.2024-30.06.2024	**Deliverable:** llvm tests build and run without any linker errors
Week 6 01.07.2024-07.07.2024	Buffer week if there are delays in patch reviews or other deliverables.
Week 7 08.07.2024-14.07.2024	Try to build the clang project with the new export infrastructure. Resolve bugs in missing symbol exports and the idt tool. **Deliverable:** Clang tools build without any linker errors.
Midterm Evaluations	
Week 8 15.07.2024-21.07.2024	Run build and runtime benchmarks on clang using the new symbol exports. **Deliverable:** A short technical report comparing performance, binary size reduction, etc.
Week 9 22.07.2024-28.07.2024	Demonstrate the flexibility of the new approach by enabling the test plugins in the clang infrastructure. That might require several changes in the cmake configurations and re-enabling clang's plugin registry on Windows. **Deliverable:** Clang test plugins build without link errors
Week 10 29.07.2024-04.08.2024	Support running of clang test plugins on Windows. **Deliverable:** Fully support test clang plugins on windows.
Week 11 05.08.2024-11.08.2024	Scale the clang plugin infrastructure on Windows to real-world plugins such as the Clad plugin enabling autodiff for C++ in clang. **Deliverable:** Demonstrate a complex clang plugin such as Clad running on windows.
Week 12 12.08.2024-18.08.2024	Buffer week if there are delays in patch reviews or other deliverables.
Week 13 19.08.2024-25.08.2024	Extended testing, developing documentation, presenting the work. **Deliverable**: test cases, demonstrated reduction of the binary sizes, blog post about the achieved results; presentation at the compiler-research.org team meeting.

Stretch goals

- Update ClangRepl to use Orc runtime on windows to fix missing symbols errors from brittle hardcoding of a custom list of explicitly exported c++ runtime symbols from the compiled binary.
- Add more ClangRepl windows tests to make sure it continues to work reliably on windows.
- Prototype support for remote Orc JIT on windows.

Personal details

Name: Thomas fransham
Email: tfransham@gmail.com
Time Zone: GMT
Github: https://github.com/fsfod
Discord: fsfod
Country: UK