llvm-mingw

https://github.com/mstorsjo/llvm-mingw

+

Martin Storsjö, VideoLAN Developer Days 2024

What is llvm-mingw?

- A mingw-w64 (Windows) targeting toolchain based on LLVM components
  - Freely redistributable, properly licensed opensource
- Most LLVM components can be used as drop-in replacements in an existing toolchain
  - Clang ↔ GCC
  - LLD ↔ GNU ld
  - compiler-rt + libunwind ↔ libgcc
  - libc++ ↔ libstdc++
  - LLDB ↔ GDB
- llvm-mingw is set up standalone from scratch with all components replaced with LLVM counterparts

Why?

- Originally: Wanting a mingw toolchain to target Windows on ARM / AArch64
- Ended up as a universal modern toolchain alternative
  - Equal support for all 4 architectures: i686, x86_64, armv7, aarch64
  - Modern features
    - PDB debug format support
    - Address sanitizer, undefined behaviour sanitizer
    - Control Flow Guard
- LLVM code base very easy to work with

What does it look like?

- Small, self-contained toolchain package
  - Every toolchain package can cross compile for Windows, for any of the target architectures i686, x86_64, armv7, aarch64
  - Prebuilt toolchains running on
    - Linux: x86_64, aarch64
    - macOS: Universal
    - Windows: i686, x86_64, armv7, aarch64
- Releases built on Github Actions
  - A new release every 2 weeks during most of the year

What’s new (since last year)?

GCC

GCC targeting aarch64-w64-mingw32

- The original purpose of llvm-mingw was for targeting ARM and ARM64
- GCC is getting support for targeting ARM64/Windows now as well
  - Work ongoing for a couple of years, initial steps are finally upstreamed
  - Still in early stages: Only supports C, not C++ yet
    - No unwind info, no exception handling
  - Not ABI compatible with the established mingw/aarch64 ABI yet
    - wrong size for long double
    - Variadic arguments uses the wrong calling convention
  - Only uses msvcrt.dll, not UCRT yet
- End target is Cygwin/MSYS2 support (which is needed for fully native Git on Windows)

What’s new in mingw-w64

D3D12

- Much more complete D3D12 headers (via Wine)
  - mingw-w64 takes many headers from Wine
  - A frequent complaint used to be that the D3D12 headers were outdated
  - The main D3D12 headers should now be up to date with the latest MS SDKs (as of end of 2023)

What’s new in mingw-w64

Math functions from UCRT

- Traditionally, mingw has used msvcrt.dll
- msvcrt.dll was originally provided by MSVC 6.0 in 1998
- Now ships as part of Windows, parts updated along the Windows versions, parts stale, parts missing (only promises what MSVC 6.0 did)
- mingw has provided their own, statically linked replacements for many things - in particular for math - for completeness and C99 compliance
- UCRT is C99 compliant
  - Can skip the statically linked math functions
  - Some UCRT math functions are much faster than the old ones provided by mingw (mingw powf was 7x slower than UCRT)

What's new in LLVM
`-fno-auto-import`

- Normally, external variables can be linked from a DLL without `dllimport` attributes - comes with a small code generation overhead (all potentially dllimported variables referenced via `.refptr` stubs)
  - With GCC, this can be omitted with `-mcmodel=small` (GCC only does `.refptr` on x86_64)
  - Felt that code model isn't the right match here - range is only one out of many reasons for `.refptr` (Clang uses it on all architectures)
- Added a new option `-fno-auto-import` in Clang, for both compilation and linking
  - Affects code generation when compiling
  - Tells the linker to not auto import variables
    - Gives a linker error rather than potential runtime error, if code was compiled with the options above but variables would have been auto imported

What's new in LLVM

Misc

- Improved LTO support
  - The whole base mingw-w64 libraries, including CRT startup files, can now be built with LTO
    - Not really relevant for real toolchain use (not provided in prebuilt toolchains)
    - If base files are LTO compiled, every linked executable requires LTO compilation
- COFF linker now respects `SOURCE_DATE_EPOCH` for timestamps in binaries
  - Useful for reproducible builds

What’s new in LLVM
ARM64EC

- Lots of work on ARM64EC - ”Emulation Compatible”
  - Windows 11 on ARM64 can emulate x86_64 binaries
  - ARM64EC is an ABI for generating ARM64 code that fits into the x86_64 emulator
    - Struct layouts, calling conventions match that of x86_64
    - Some ARM64 registers disallowed (everything must be mappable back to a x86_64 register)
    - Allows you to get near-native speeds for critical code by compiling it for ARM64EC
    - Can mix and match ARM64EC and x86_64 DLLs within the same process
      - Allows x86_64 plugin DLLs in an otherwise fully ported app
    - Allows mixing ARM64EC and x86_64 within each EXE/DLL, on a function level

What's new in llvm-mingw

C++ Modules

- Support for C++20 modules
  - Works with recent CMake versions
  - The toolchain has to provide `clang-scan-deps` to let the build system figure out dependencies between source files and modules
- Support for using libc++ as a C++23 `std` module
  - Required cleanups of mingw-w64 headers

What's new in llvm-mingw

Switching to Clang config files

- The defaults in Clang for a mingw target is to use libgcc, libstdc++ and link with `<triple>-ld`
- llvm-mingw traditionally uses wrapper scripts
  - `<triple>-clang` (and `<triple>-gcc` to ease use with some build systems) is a wrapper (script or executable), internally invoking `clang -target <triple> -rtlib=compiler-rt -unwindlib=libunwind -stdlib=libc++ -fuse-ld=lld`
  - Wrappers hide this configuration from other tools
    - Other Clang based tools like clangd (for IDE code completion) or clang-scan-deps operate on stored commands
    - The tools see a command like `<triple>-clang -c mysource.cpp` - they can deduce that this means `clang -target <triple> -c mysource.cpp`
    - They don't know that `<triple>-clang` actually is a wrapper that implicitly sets other flags
- Could set hardcoded defaults in Clang binary when compiling
  - Hardcoded defaults apply to all targets, making the same Clang binary unusable for any other target

What's new in llvm-mingw

Switching to Clang config files

- Instead set target specific defaults via config files
- Next to the Clang binary, store a config file `<triple>.cfg`
- When Clang (or a Clang based tool) is invoked, it implicitly looks for any config files for the specific target it is invoked for (either explicit `-target` option, or implicit default)
- Avoids needing to set defaults in wrappers
- Allows getting rid of wrappers (almost)
  - `<triple>-clang` can be a symlink to plain `clang`, clang picks up the target from the executed binary name
  - Implicit options for UWP targets still require a wrapper though

Taking the same concept further?

Extending the concept?

- Single toolchain package for cross compiling for a multitude of targets is a neat thing
  - Zig does this very nicely, even for multiple OSes
- Could we do the same kind of setup for e.g. Linux cross compilation?
  - Targeting Linux, ”any” arch, from any OS
  - Easily set up a similar kind of toolchain for targeting Musl
    - Single package, prebuilt Musl and libc++ for a number of architectures
    - Not useful for building programs for ”regular” Linux distributions though, as they usually have Glibc, but works well with static executables
    - 69 MB package for targeting 6 architectures (i386, x86_64, arm, aarch64, powerpc64le, riscv64)
- Haven’t productized `llvm-musl` (yet)
  - Unsure about committing to maintaining another project
  - Not sure how much extra value it adds vs regular GCC cross compiler packages

Extending the concept?

- What about Glibc?
  - Compiling Glibc with Clang doesn’t (even close) work out of the box upstream
  - There is a somewhat maintained branch with 137 patches on top of upstream, that should be buildable with Clang
  - Even then, bootstrapping it is much more complex than mingw-w64 or Musl
- Even if we’d have a Clang+Glibc cross compilation toolchain, we’d need libstdc++ to produce binaries that work on a regular distribution (or statically link libc++)
  - Maybe the same issue with libunwind as well, but that can possibly masquerade as libgcc
  - Pretty much would need to set up the cross sysroots with GCC anyway

Thank you!

https://github.com/mstorsjo/llvm-mingw