LLVM in OpenMandriva

An update from the first Linux distribution to use clang as its main compiler

Bernhard “bero” Rosenkränzer <bero@lindev.ch>

LLVM distributors conference 2021, 2021-09-16

What is OpenMandriva?

- Linux distribution focused on the desktop, more recently also targeting ARM and RISC-V devices (e.g. PinePhone) and devboards (e.g. Beagle V, Rock Pi, Raspberry Pi) and servers/VMs
- Independently developed distribution, not based on anything else (but borrowing some stuff from everywhere ;))
- Purely community driven Open Source project, descendant of Mandrake/Mandriva
- Switched to clang as main compiler as early as 2015
- For more details, check our website or communications channels mentioned at the end

Why did we switch to clang?

- We like readable, maintainable code. gcc is great for users, but not so good for working with its code
- We expect targets like amdgpu to become increasingly important
- Less duplication - we need LLVM anyway (Mesa)
- Useful tools (sanitizers, creduce)
- Usually not a lot of trouble with prerelease versions (we’ve been using 13.0 since -rc1)

What parts do we (not) use, and why?

- LLVM libraries, clang, lld, openmp, polly, libunwind:
  used as default tools
- LLVM binutils:
  used a lot, but through wrapper scripts. We want clang and gcc to coexist — nm, ar and friends from LLVM binutils fail on gcc LTO files, GNU nm, ar and friends fail on LLVM LTO files (bitcode). So we use a wrapper script that checks for either compiler's LTO output and picks the tool version that will work

What parts do we (not) use, and why?

- *PGO:*
  Used for some key libraries (where generating useful profile data is possible/easy). Temporarily mostly disabled because of bug 51624.
  We expect to extend the use of PGO to additional packages.
- lldb:
  Included, but most people use gdb because it's what they're familiar with
- compiler-rt:
  Used to be enabled by default, but we had to switch back to libgcc because of corner cases (library X built with gcc [therefore linking to libgcc] and library Y built with clang [including compiler-rt] being used in the same application can cause clashes), plan is to switch back to compiler-rt after sorting out problems

What parts do we (not) use, and why?

- libc++:
  available, but not used by default: binary compatibility concerns with other distributions. If our system libraries (e.g. Qt) use libc++ and a user installs a non-free application/game built on another distribution built against Qt using libstdc++, that will cause problems. And we don't want to ship multiple copies of system libraries...

What parts do we (not) use, and why?

- libc:
  We're watching, but it's not really usable yet. Also, same binary compatibility issues as with libc++. At some point, if libc turns out well, we might do a libc/libc++ build to see if it's worth doing from a performance perspective...

Modifications

- Minor tweaks like adding target triplets, default settings
- Backports of some patches from master and phabricator (RISC-V linker relaxation for lld)
- Probably most controversial: We change the default of `-fgnuc-version` to **`11.2`**

Why -fgnuc-version=11.2?

```
#if __GNUC__ >= 8
      fast_version_that_uses_advanced_compiler_features();
#else
      slow_version_that_works_with_ANSI_C_from_the_1970s();
#endif
```

- Yes, this shouldn’t be done — but it’s everywhere, and distributions by definition deal with 3rd party code. We’re not in a position to fix this.

-fgnuc-version=11.2 problems

```
#if __GNUC__ >= 11
       something_really_gcc_only();
#else
       alternative_version_that_works_with_clang_too();
#endif
```

- We get this instead of the slow versions mentioned before — but this is better: It results in a compile failure so it doesn't go unnoticed (unlike the optimizations we miss out on by pretending we're gcc 4.2.1). And of course `CFLAGS="$CFLAGS -fgnuc-version=4.2.1"` is always a "quick fix"

export CC=”gcc”; export CXX=”g++”

- Not considered a taboo — still used in a few packages (we’d like to fix this, but it’s not a high priority — help welcome!):
  - glibc - big pain, lots of gcc specific code
  - grub2 - theme engine broken when built with clang
  - ppl - compile time error
  - wine (x86_64 only) - 64-bit winecfg crashes on startup when built with clang
  - mesa (x86_64 only) - reports of crashes on old hardware that are fixed by CC=gcc, no details provided - possible use of SSE3 or higher despite no -march= allowing it?
  - systemd (x86_64 only) - hang on upgrades when built with clang on x86_64 but not znver1

Thanks for your attention

We can’t take questions because of the lightning talks format — but feel free to ask on the CFP, or find the OpenMandriva team on

Matrix: https://matrix.to/#/%23openmandriva:matrix.org
IRC: libera.chat #openmandriva
Forum: https://forum.openmandriva.org/

or email bero@lindev.ch