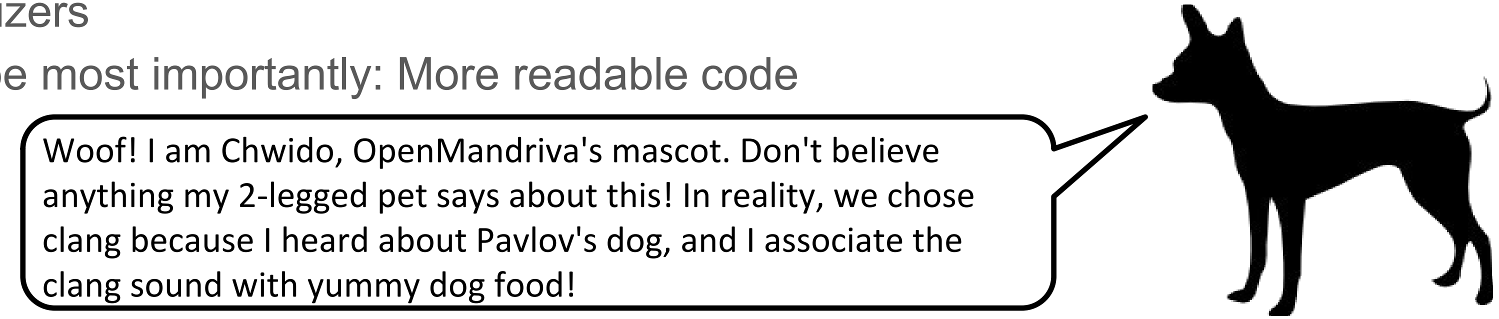
Switching a Linux distribution's main toolchain to LLVM/Clang

Bernhard "Bero" Rosenkränzer, OpenMandriva/LinDev

EuroLLVM 2019

When and why did we do it?

- Made the decision and started the transition around the time of the clang 3.5 release - when clang started being competitive in terms of completeness and performance. First build with cc -> clang symlink was a pre-3.5 snapshot (svn rev 209634, May 29, 2014). The first release with clang as default compiler happened 5 months later, Oct 26, 2014.
- Most important reasons for making the switch:
  - Better compile times (for resulting binaries with very similar performance)
  - Less pain with crosscompilers
  - Sanitizers
  - Maybe most importantly: More readable code


Woof! I am Chwido, OpenMandriva's mascot. Don't believe anything my 2-legged pet says about this! In reality, we chose clang because I heard about Pavlov's dog, and I associate the clang sound with yummy dog food!

Benefits discovered later

- Usually less breakage even when using snapshots (important to us: There’s not much of a point in developing against an older version of a toolchain when we already know our final release will be using a newer one. We know our 4.1 release will use clang 9 or 10, so we’ll switch our development tree to a 9 snapshot as soon as our 4.0 release (which uses clang 8) is done).
  This helps find potential bugs early, but lots of Internal Compiler Errors can slow down other work.

And pain points...

- Biggest problem right now: **Missing RISC-V support.**
  Our newest target is RISC-V Linux (rv64imafdc) -- presently not supported at all in LLVM/Clang master, only partial patches available.
- Temporary workaround: Use gcc on RISC-V, clang on all other architectures (x86_64, aarch64, i686, armv7hnl, znver1)

And pain points...

- Many projects (still) never test building with anything but gcc - don’t notice when code relies on gcc extensions (VLAIS, nested functions, asm goto, …)
- Projects are quick to blame the compiler for their undefined behavior code that gcc happens to compile as expected
- Some upstream projects (elfutils, to name the biggest one) are downright hostile to "make it build with clang" style patches
- Missed optimizations because of #if statements
-
```
#ifdef __GNUC__
do_something_sane();
#else
assume_the_compiler_sucks();
#endif
```

And pain points...

- Or worse:

```
#if defined(__GNUC__) && !defined(__clang__)
do_something_sane();
#else
// based on trying clang 3.0 and never
// looking at current versions...
assume_the_compiler_sucks();
#endif
```

And pain points...

- Or worse:

```
#if __GNUC__ >= 5
do_something_sane();
#else
assume_the_compiler_sucks();
#endif
```

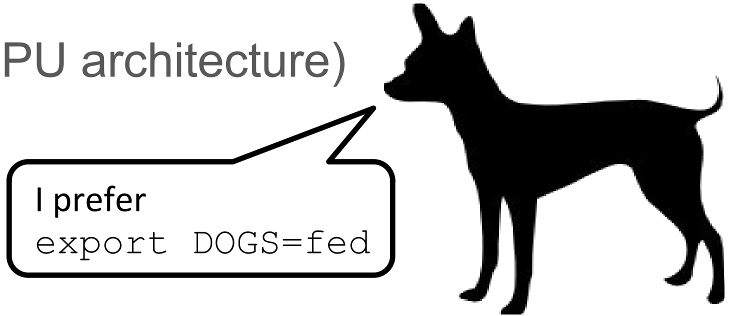
Even clang master pretends to be gcc 4.x in terms of __GNUC__ defines

Increasing `__GNUC__`

- In terms of standard compliance etc., current clang is much closer to gcc 8.x than 4.x, so increasing the values for `__GNUC__` and friends makes sense…
  … except clang doesn't have some gcc 8.x extension, triggering new compile failures in e.g. gnulib's `__builtin_mul_overflow_p` use if "gcc" is new enough
- Projects should really use `__has_builtin`/`__has_attribute` instead of `#if __GNUC__` to determine what features are available (but not all projects care -- and it breaks support for old compilers).
- We're using clang with increased `__GNUC__` now -- better to trigger a compile failure (that will be fixed) than a missed optimization nobody will notice. Should that patch go upstream?

export CC=gcc; export CXX=g++

- We don't hate gcc -- so "`export CC=gcc`"/"`export CXX=g++`" is an acceptable workaround that doesn't make developers jump at the package when there's more important problems -- without more contributors, we're likely to stick with this workaround in some packages for a while…
  (Yes, consider this a call for volunteers ;))
- Currently:
  - 20330 packages total
  - 139 have clang related patches
  - 327 use gcc/g++ (some only on a specific CPU architecture)
  - over 2000 related patches upstreamed



export CC=gcc; export CXX=g++

- *various packages* -- clang generates calls to `__muloti4` and similar functions that are provided by compiler-rt, but not libgcc. Workarounds are either `CC=gcc` or `--rtlib=compiler-rt`.
  Either workaround "solves" the problem, but linking libraries built with `--rtlib=compiler-rt` into applications built with libgcc may cause hard to trace issues
- *gnu-efi* and other parts of the UEFI stack -- Makefiles hardcode compiler flags not currently available in clang, e.g. `-maccumulate-outgoing-args`

export CC=gcc; export CXX=g++

- *glibc* -- relies on numerous gcc and GNU binutils specific features
- *Linux kernel* -- `asm goto` is the primary remaining pain point here
- *mesa* on x86_64 -- compiles fine with clang, works well on newer hardware, but crashes at runtime on older x86_64 hardware when built with clang (AVX instructions somehow getting run on processors that don’t have them). Clang-built mesa on aarch64, armv7hnl, i686 seems to be ok.
  Needs debugging, probably mesa uses hand-written asm code and CPU feature detection goes wrong.
- *blender* -- implicit `float` to `bool` conversions, and SSE and OpenMP support available only with gcc, should be fixable

export CC=gcc; export CXX=g++

- *gcc* -- doesn’t like being compiled with anything but itself (sadly, would be nice to not have circular dependencies. Build gcc with clang and clang with gcc).
- *grub2* -- inline asm constructs (.addrsig etc.)
- *autoconf* -- hardcodes filenames for known compilers [but not clang/clang++] in its AC_PROG_CC and AC_PROG_CXX macros. This has been fixed for a while, but there hasn't been a release for years.
- *openjdk* on 32-bit x86 -- makes ABI assumptions that work only with versions of glibc that don't use SSE, unless compiled with (gcc specific) `-mincoming-stack-boundary=2` flag https://bugs.openjdk.java.net/browse/JDK-8199936
- *gpm* - heavy use of nested functions

Qt issue with -reduce-relocations

- *Qt* -- signal/slot connections to lambdas fail at runtime if Qt is built with clang and -reduce-relocations -- *[update: dropped -reduce-relocations instead of using gcc]*
  A similar problem seems to exist with ICC: https://bugreports.qt.io/browse/QTBUG-52439.
- Needs further debugging.

LLVM components we aren’t using yet

- *LLD* -- not yet feature complete to a point where it could fully replace gold and BFD ld, will keep checking it out as we move to newer major releases
  Currently mostly missing support for linker script constructs used in many packages
- *as* -- partially used through clang inline assembly, but /usr/bin/as is still GNU (binutils) as, mostly because of projects using pre-unified ARM32 asm code (and some extensions), and command line compatibility (gcc makes assumptions about *as*)
- *strip, objdump* -- rpm’s separate debug package feature is currently tied to the elfutils implementations of strip and binutils objdump

nm and friends...

- llvm-nm and friends can't handle object files generated with gcc -flto -- binutils nm and friends can't handle object files generated with clang -flto...
  So we need a different handler. This should really be fixed upstream -- but...
  Which upstream? Current "solution":

```
REAL_NM=binutils-nm
PARENT="$(readlink /proc/$PPID/exe)"
WRAPPED=false
if [ -z "$PARENT" ]; then
    WRAPPED=true
elif echo "$PARENT" |grep -qE -- '-nm$'; then
    WRAPPED=true
elif echo "$PARENT" |grep -qE -- 'qemu'; then
    if grep -qE -- '-nm' /proc/$PPID/cmdline; then
        WRAPPED=true
    fi
fi
```

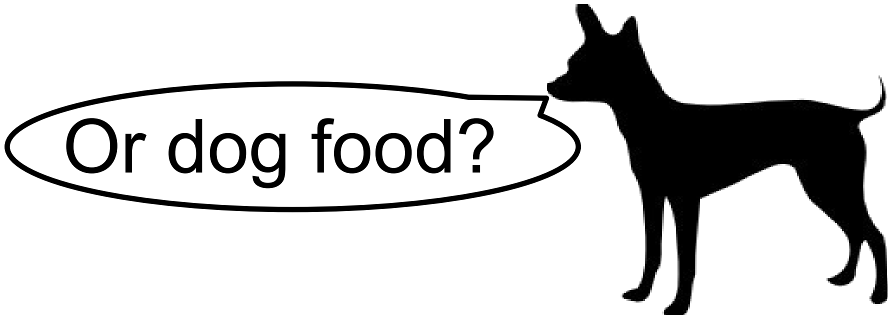
nm and friends...

```
if ! "$WRAPPED"; then
    for i in "$@"; do
        [ "$(echo $i |cut -b1)" = '-' ] && continue
        if echo "$i" |grep -qE '\.(o|a)$' && [ -e "$i" ]; then
            if LANG=C gcc-nm "$i" 2>&1 |grep -qi "file format not recognized"; then
                which llvm-nm &>/dev/null && REAL_NM=llvm-nm
                break
            fi
        fi
    done
    if [ "$REAL_NM" = 'binutils-nm' ] && which gcc-nm &>/dev/null; then
        REAL_NM=gcc-nm
    fi
fi
exec "$REAL_NM" "$@"
```

LLVM components we aren’t using yet (and might never use)

- *libc++* -- Binary compatibility issues: If we build e.g. Qt with libc++ and a user tries to install a binary that uses Qt and was built on any other Linux distribution, it’s going to fail. Can’t do this while people insist on using binary-only software.
  We might start using it on architectures where binary-only components don’t matter [yet], such as RISC-V
- *compiler-rt* -- similar issue - a library linked to compiler-rt doesn’t always work with binaries built with libgcc. Also, gcc (which we still rely on for some packages) can’t use compiler-rt

Questions? Comments?



Ask here, or email

bero@lindev.ch

http://openmandriva.org/
http://lindev.ch/