OpenMandriva’s switch to clang

Linux Plumbers Conference 2014,
LLVM Microconference

Bernhard “Bero” Rosenkränzer, OpenMandriva
bero@lindev.ch

For those unfamiliar with OMLx

OpenMandriva Lx is a Linux distribution primarily for desktop users.

Released versions run on x86_64 and i586, experimental builds exist for armv7hl and aarch64.

For those unfamiliar with OMLx

After the 2014.1 release, we’ve decided to make clang our default compiler.

rpm is now configured to default to clang, /usr/bin/cc is a symlink to clang, /usr/bin/c++ is a symlink to clang++

We also switched the default linker to gold.

Why make that change?

gcc is good -- but comparing gcc today (4.9) to gcc 3 years ago (4.6) and clang today (3.5) to clang 3 years ago (2.9) shows much bigger improvements on the clang side (probably because its code is more readable).

Keep up the pace!

Why make that change?

switching the default doesn’t mean throwing gcc away, we keep packaging it and we can and do fall back to

```
CC=gcc CXX=g++ ./configure …
```

where it makes sense.

Overall experience

The transition was quite smooth - a mass build resulted in around 800 build failures due to compiler changes. Not more than we usually see with a major gcc update. (And only 1 compiler crash!)

However...

Blame the compiler...

“That thing is pure crap. It can’t even compile hello world. configure tells me ‘C compiler cannot create executables’”.

-- Complaint by someone trying to rebuild packages… after doing

`export CFLAGS=”-O3 -frecord-gcc-switches”` in `~/.profile`

Real issues: gcc extensions

Most failures are caused by code relying on gcc extensions:

- Nested functions (elfutils, rpm)
- Variable-length arrays in structs (kernel)
- __builtin_va_arg_pack (various libcs)

Real issues: Bugs ignored by gcc

```
class A {
    friend b(int, const char *s=0);
};
```

(default parameter given in friend declaration)

Seen in FLTK.

Real issues: Bugs ignored by gcc

```
class A {
    enum { a, b, c, d };
};
int main(int argc, char **argv) {
    return A::a;
}
```

Seen in FLTK.
Very current gcc complains about this as well.

Real issues: Bugs ignored by gcc

```
void something(char n[30]) {
    if(!memcmp(buffer, n, sizeof(n))) {
        …
    }
}
```

Real issues: inline semantics

```
/usr/bin/ld: error: ../mpi/.libs/libmpi.a(mpi-bit.o): multiple definition of '_gcry_mpih_add'
/usr/bin/ld: ../mpi/.libs/libmpi.a(mpi-add.o): previous definition here
```

Caused by code assuming C89 inline semantics -- fix: -std=gnu89

Real issues: to be debugged

X.Org drivers built with clang crash on X startup. For now, we’re using CC=gcc there.

Two patches to clang itself...

Clang's __GNUC__ and __GNUC_MINOR__ macros identify it as gcc 4.2 -- but in fact it is much closer to 4.9.

Only drawback: glibc assumes gcc 4.9 has __builtin_va_arg_pack -- but glibc headers can be patched

Two patches to clang itself...

Not quite standardized triplets: `armv7hl-linux-gnueabi` vs. `armv7l-linux-gnueabihf`

OpenMandriva used to use `armv7hl-linux-gnueabi` (Red Hat and a few others still do) - clang didn’t recognize this as a hardfloat target

Upstreaming

Most patches to make things work with clang are accepted upstream, with a few notable exceptions (elfutils etc.)

We may need to create a central repository to collect those patches for other distributions making the switch.

What else?

What else can we do to make switching to clang even more painless?