Compiling Android with Clang

Linux Plumbers Conference 2013
Bernhard “Bero” Rosenkränzer
bero@linaro.org

Linaro

Why would you want to do this? gcc is good...

gcc is good enough – but some competition can never hurt. (If nothing else, a clang build allows gcc people to see where clang outperforms gcc, and what can be done to get gcc to generate the same (or better) code).

Parts of Android (libpng, compiler-rt, renderscript) are always built with clang (unfortunately, a rather old version of clang that has been forked into the Android tree) so they can integrate better with llvm GPU code – and this is expected to increase...

And of course a different compiler may have different warnings pointing out bad code...

Problems...

gcc has been the only relevant compiler for so long that people have come to believe anything that happens to work in gcc is fine...

even if it is weird.

```
int vecs[3][4];
vecs[0][5] = something; // assume this is the same as
vecs[1][1]
```

Found in external/quake/src/WinQuake/gl_model.cpp

This also broke gcc 4.8, but didn't even produce a warning in earlier gcc versions.

Linaro

Command line arguments to compilers

Android's build system doesn't make it easy to use a compiler that doesn't take exactly the arguments they expect – and doesn't have an official way to detect compilers, compiler versions, etc. Often, this even causes problems when updating to a newer version of gcc...

Much more so when trying to use a different compiler altogether, clang's invocation is similar to gcc, but not identical (especially when crosscompiling).

But it is easy enough to write a wrapper that takes gcc arguments (and gcc command names) and translates them to what clang expects to see.

http://git.linaro.org/git-ro/people/bernhardrosenkranzer/clang-wrapper.git

And it's possible to build Android-style toolchains using the wrapper. Prebuilt binaries for Linux X86 are built daily, and released to
http://snapshots.linaro.org/components/toolchain/llvm-clang-trunk/latest
http://snapshots.linaro.org/components/toolchain/llvm-clang-3.3/latest

A possible pitfall: arm-linux-androideabi vs. arm-linux-gnueabi

Clang accepts the arm-linux-androideabi target – but internally, takes it as an alias to arm-linux-gnueabi – almost right, but not quite 100%.
Looking at gcc sources shows a few subtle differences that can break things in unexpected ways:

- androideabi forces enum sizes to 32 bits.
  Dalvik has some code that relies on this.
- Android defaults to generating position independent code (PIC).

The solution:

- Clang has support for fixed size enums (`-fno-short-enums`) and PIC code (-fPIC) – so if an android target is requested, we just add those parameters to the command line in the wrapper

Linaro

CPU feature checks

gcc adds some automatic defines to indicate some CPU characteristics – such as:

- `__ARM_FEATURE_DSP` on ARM CPUs other than Cortex-M.
- `__ARMEB__` on Big-Endian ARM targets
- `__THUMBEB__` on Big-Endian ARM Thumb targets
- `__IWMMXT__` on CPUs with IWMMXT support (XScale)

Bionic relies on those to select the best optimized version of strcmp for the CPU being targeted.

Those defines don't exist in clang – so let's `-D` them in the wrapper (currently done only for `__ARM_FEATURE_DSP`)

Constructs accepted by gcc, but not by the standard

Arithmetic on void* pointers

```
void *buf;
[…]
buf = buf + offset;
```

(found in *frameworks/base/core/jni/com_google_android_gles_jni_GLImpl.cpp*)

What they meant to do:

```
void *buf;
[…]
buf = ((char*)buf) + offset;
```

Similar issue: `char[]` and `char*` are not the same (*frameworks/av/media/libmedia/Visualizer.cpp*):

```
Visualizer.cpp:147:20: error: arithmetic on a pointer to an incomplete type 'char
[]'

    memcpy(&p->data+sizeof(int32_t), &size, sizeof(size));

           ~~~~~~~~~^
```

Use of C++11 features without putting the compiler into C++11 mode

```
SkRect r{0,0,0,0};
```

found in frameworks/base/core/jni/android/graphics/Paint.cpp – possible fixes are

```
SkRect r = { 0, 0, 0, 0 };
```

or telling the compiler we want ISO C++ 11 (`-std=c++11`, `-std=gnu++11`)

gcc extensions not supported by clang

Named registers

```
register unsigned long current_sp asm(“sp”);
```

Used in several places in the kernel

Support for this might be added in an upcoming version of clang.

Variable length array in structs

Used in several places in the kernel and in skia

Clang developers are pointing out that this “extension will never be supported”, so we need to adapt the code.

Nested functions

Used extensively in elfutils

Extension will likely never be supported in clang because (outside of elfutils) virtually nothing makes use of this extension.

Fix: Just un-nest them (and win a chance to receive upstream's “moron of the year” award for uglifying the code)

Using functions before they are declared or prototyped

- *hardware/libhardware/legacy/wifi/wifi.c*:
  747 `wifi_close_sockets(index);`
  /* interpreted as implicit declaration of "`int wifi_close_sockets(int)`"
  822 `void wifi_close_sockets(int index)`
  /* conflicting declaration (different return type) */

  Fix: Add a prototype declaration, or move the functions around

Linaro

Variable-length arrays of non-POD elements

frameworks/base/libs/hwui/OpenGLRenderer.cpp:

AAVertex wLines[verticesCount];

Fix: Turn it into AAVertex *wLines, allocate with new/delete[]

Linaro

struct vs. class

Swallowed by gcc but not clang:

```
class TreeIteratorBase;
[…]
struct TreeIteratorBase {
        […]
};
```

(*frameworks/compile/libbcc/include/bcc/Linker.h*)

class and struct are the same thing with different default visibility – but let's be consistent in what we use to refer to a speicifc class/struct

Slightly different symbol visibility/coexistence and symbol resolution rules

`__kernel_sindf(double)` and friends in Bionic lead to clashes because they're defined multiple times (header included by several files), causing a fatal error with clang – fix: declare `static inline` (similar problems occur throughout e2fsprogs, and with `rotate270` and friends in Galley2's JNI code)

`__weak_reference` and `__strong_reference` need to be defined in asm code for clang

```
static const int digits10 = digits10<int, digits, is_signed>::value;
```
(*external/astl/include/limits*)
causes a compile error with clang because it assumes the right-hand reference to “digits10” to refer to the static const int definition on the left-hand side – passing template parameters to an int doesn't make much sense.
Fix:
```
static const int digits10 = ::digits10<int, digits, is_signed>::value;
```

Libunwind API difference

API change in clang's unwinder:

```
uintptr_t pc = _Unwind_GetIP(context)
```

```
uintptr_t pc;
_Unwind_VRS_Get(context, _UVRSC_CORE, 15, _UVRSD_UINT32, &pc);
```

Additional warnings (in code built with -Werror by default)

```
dalvik/vm/analysis/RegisterMap.cpp:507:9: error: variable 'addr' is used
uninitialized whenever switch default is taken [-Werror,-Wsometimes-
uninitialized]
        default:
        ^~~~~~~
dalvik/vm/analysis/RegisterMap.cpp:513:52: note: uninitialized use occurs
here
        const RegType* regs = vdata->registerLines[addr].regTypes;
                                                   ^~~~
dalvik/vm/analysis/RegisterMap.cpp:497:17: note: initialize the variable
'addr' to silence this warning
        int addr;
                ^
                 = 0
1 error generated.
```

(Nicely spotted and nice message, but not relevant in this particular case because the default: switch ends in “dvmAbort();”)

Linaro

And of course every compiler has its own share of bugs...

http://llvm.org/bugs/show_bug.cgi?id=16590
http://llvm.org/bugs/show_bug.cgi?id=16736

Workaround: `-O0` can be rather useful for some pieces of code...

“Undefined reference” errors in Gallery2 JNI code (similar issue in srec) – complaining about functions defined `__inline__` in the same file. No obvious bugs in the code – seems to be a clang bug (but needs more investigation).
Workaround: Make it “`static`” instead of “`__inline__`”

“Undefined reference” errors in e2fsprogs, seemingly caused by “`extern`” declarations in a header, on functions not actually being used. (They are used inside of unused “`inline`” functions, so this may be related to the problem above)

Current status: It compiles, but...

With modifications along these lines, a Nexus 10 image passes Windows style QA (“It compiles, therefore it works”®)
The image size is slightly larger than an equivalent build with gcc 4.7 (gcc 4.8 can't compile a bootable Exynos5 kernel at the moment)
Build time is significantly faster (60min vs. 90min)

Just one little detail that's easy to overlook:
It doesn't boot (userland issue in a component needed very early [possibly Bionic, init] – same problem occurs when using a gcc-built kernel with clang-built userland)

Reasons for this still need to be investigated.

® indicates what would be a registered trademark of Microsoft Corp. if trademarks Were assigned based on actual practice instead of on demand ;)

Questions?

Ideas?

Linaro