Building Android with clang

Linux Plumbers Conference 2014, LLVM Microconference

Bernhard “Bero” Rosenkränzer, Linaro
bero@linaro.org
Quick status update for the impatient

“It compiles, therefore it works”

(A dangerous method of detecting bugs - might be patented by Microsoft QA dept.)
Quick status update for the impatient

boots to text mode

“It compiles, therefore it works”
Quick status update for the impatient

and runs many apps

boots to text mode

“It compiles, therefore it works”

Compiles for Nexus 4, 5, 7 and 10.

4 is currently untested. 10 works great, 7 works
Patch submission status

Overall 112 patches submitted
81 accepted
27 waiting
4 abandoned in favor of better solutions

Script to apply all waiting patches:
git://android.git.linaro.org/aosp-patchsets.git
Quick performance check

clang-built AOSP is currently about 2.6% bigger than gcc-built AOSP. Performance depends on what is being checked, but overall gcc is still ahead.

clang is around 20% faster at “make droidcore”.
Cheating twice...

We’re currently setting
\texttt{LOCAL\_CLANG := false}
for /init and the GLESv1 and GLESv2 wrappers in frameworks/native/opengl/libs -- causing those bits to be compiled with gcc.
Cheating twice...

clang-built /init reboots the device before adb or other tools useful for debugging come up (even running clang-built init on an already built system causes a reboot - we’re likely triggering an error handler)

clang-built GLESv2 crashes the UI on startup.
Workaround for GLESv2 wrapper

The GLESv2 wrapper has a generic C version that works perfectly with clang (`#if USE_SLOW_BINDING`), and asm versions that don’t.

Unfortunately, adding overhead to any OpenGL call is not a good idea...
#define GET_TLS(reg) "mrc p15, 0, " #reg ", c13, c0, 3 \n"
#define CALL_GL_API(_api, ...) 
asm volatile( 
    GET_TLS(r12) 
    "ldr   r12, [r12, %[tls]] \n"
    "cmp   r12, #0 \n"
    "ldrne pc, [r12, %[api]] \n"
: [tls] "J"(TLS SLOT_OPENGL_API*4), 
    [api] "J"(__builtin_offsetof(gl_hooks_t, gl._api)) 
: "r12" 
);
Sometimes clang is picky...

● “register” keyword usage in Chromium
● array subscripts of type “char” (hexdigit[‘0’] =…)
● undefined internal functions, undefined variables
● Use of GNU initializers instead of C99
● Conditions that can’t be true
Sometimes clang is picky...

- empty structs
- `asm("add w0, w0, #-1");`
  - (converted to `sub w0, w0, #1` by gas, but not by clang)
- unused parameters
Sometimes clang is picky...

- Complains even about code that is about to be thrown away:

```c
static void a();
void b() {
    if (false)
        a();
}
```
... and sometimes it finds real bugs

UCHAR a[X];
for(int i=0; i<X; i++)
    b = a ? tagCpe++ : tagSce++;

from MPEG TP decoder
... and sometimes it finds real bugs

```c
char str[30];
snprintf(str, "%s", x);
if(str == NULL)
    return ERROR;
```

from qcom camera HAL
... and sometimes it finds real bugs

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

from qcom bluetooth kernel module
... and sometimes it finds real bugs

class A {
    void *something() {
        if(this == NULL) return NULL;
        return something;
    }
}

from Binder
gcc extensions

AOSP used to use some gcc extensions not supported by clang:

- Nested functions
- __builtin_va_arg_pack
- variable-length arrays of non-POD types
- variable-length arrays in structs
... and 1 clang bug

There’s only 1 place in which we have to work around a clang bug:

```c
char s[x] __attribute__((__aligned__(PAGESIZE)));
```

http://llvm.org/bugs/show_bug.cgi?id=13007

/init and GLESv2 miscompiling may or may not be clang bugs.
Things that still need to be done

● Fix /init and GLES wrappers
● Investigate crashing apps
● Test other devices (esp. Aarch64, x86, MIPS)
● Set up daily builds so we detect new breakages and patches no longer applying quickly
Things that still need to be done

- Update clang (AOSP currently uses a pre-3.5 snapshot)
- Test different compiler options
- Build the kernel with clang too (currently using the prebuilt kernel)
- Investigate where clang based builds are much slower, optimize
Things that still need to be done

- Fix build failures with integrated as (right now, we’re forcing -no-integrated-as into the compiler flags)
What else?

What else can we do to help AOSP work with llvm/clang toolchains?