C11 and C++11 in FreeBSD

David Chisnall

February 3, 2013

Why?

- The language is the standard (kind-of)
- The new standards are better
- People want to use them
- We have to support them

C11 Overview

- Alignment specifiers
- Multithreading Support
- A memory model (finally!)
- Cosmetic improvements (anonymous structs and unions)
- Static assertions
- Unicode support

C++11 Overview

- Better locale support
- Atomics, threads, thread-local storage
- Smart pointers
- Tuples
- Lots of language changes: constexpr, lambdas, r-value references, initializer lists, variadic templates, range-based for loops, (very limited) type inference...

The Stack

- csu, libgcc_s, libc, libcxxrt, libc++, clang[++]
- Many 'new' features were already GNU extensions, now standard
- Names usually changed

sys/cdefs.h

- File included by all FreeBSD system headers
- Provides macros abstracting GNU features
- Now names macros with their C11 names in all dialects
- Compilers with native support for these features don't need the macros
- e.g. `_Alignof()`, `_Thread_local`, `_Static_assert`.
- You can start using them now, whatever C/C++ dialect you use.

The Compiler vs the OS

- Clang supports the language features (well, a growing subset of them)
- FreeBSD needs to support the library features
- Some come from libc++
- Some need some libc functionality as well.

Atomic Operations

- `_Atomic()` types in C11.
- `std::atomic<>` in C++, implemented on top of C11 atomics
- Complex memory model: acquire, release, consume, acquire-release, sequentially-consistent, relaxed
- Trivial for small types (CPU atomic operations, sometimes with fences)
- C++11 and C11 allow it for any types
- Implemented as library functions
- Problem: What happens with big atomics in shared memory? (nothing sensible)

Thread-Local Storage

- Already done for C, GNU `__thread` keyword
- Harder for C++
- Thread-local objects can have nontrivial constructors / destructors.
- Extra hooks needed to run them.

Non-POD TLS

How does this work?

```
struct A{
    A() { puts("Created"); }
    ~A() { puts("Destroyed"); }
};

thread_local A a;

void *thr(void *ignored) { return 0; }

int main(void)
{
    pthread_t thread;
    pthread_create(&thread, 0, thr, 0);
    sleep(1);
    return 0;
}
```

Making non-POD TLS Work

- Needs hooks in libthr for construction / destruction
- More complicated for scoped `static` variables.
- Currently: Not supported.
- Soon: ?

Quick Exit (C and C++)

- Fast exit path.
- Runs some cleanups, not all
- Added support in libc
- Working in 9.1

Exception Madness (C++)

- C++11 allows the current exception (regardless of type) to be encapsulated
- Can then be passed to another thread, added to a queue, and rethrown
- Useful for fault isolation
- Extra code in libcxxrt, provides access to current exception
- Code in libc++ uses this to box the exception and to rethrow it in another thread.

Unicode Support

- `uchar.h` (C) `<uchar>` (C++) add UTF-8/16/32 character types
- Supported by `wchar` functions already
- Needs (thin) wrappers written
- Any volunteers?
- Compiler support for unicode literals still needed (not hard)
- ... gcc has a very strange notion of sanity with respect to locales

Threading APIs

- C++11 adds some thread classes encapsulating the platform's threading APIs
- C11 adds some C threading APIs that are poorly designed
- libc++ supports the C++11 threading APIs using pthreads
- No one cares about the C11 ones (ask phk!)

Locale-aware POSIX2008 Functionality

- `newlocale()` and `duplocale()` create locales.
- `uselocale()` sets per-thread locale.
- `_l`-suffixed functions take explicit locales. e.g:
- `int asprintf_l(char **, locale_t, const char *, ...)`
- `struct lconv *localeconv_l(locale_t)`

Locales

- C++11 has rich locale support
- POSIX 2008 also has extended locale support
- libc++ uses the POSIX2008 locales (so does GNOME)
- FreeBSD 9.1 supports them.
- Useful for any multithreaded code doing locale-aware things.

Current Status

- Most of of C++11 works.
- Most of the interesting bits of C11 work
- Still lots of work to do!
- (And some C99 `math.h` functions are still missing)