Freestanding design policy

Ben Craig

Questions for audience?

- Are there any existing “minimal” C++ freestanding libraries?
- Are there any existing “minimal” C freestanding libraries?
- GPU experts?

Presentation questions

- Does the committee agree with the vision, goals, and non-goals?
- Does the general direction make enough sense to justify a proof-of-concept implementation?
- Is the proposed direction beneficial for the expected users of the freestanding implementation?

My background

- Driver developer
  - Windows, Linux, Mac OSX, Pharlap, VxWorks
- Firmware developer
  - ARMv5, Hexagon
- Experience using stripped down STLPort in the kernel.
- Experience using libstdc++, libc++, and Dinkumware in firmware.

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

* Mark as much of the library freestanding as we can without violating the OS interaction or space overhead restrictions

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

*Don’t drag along partial features that are useless in isolation

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

*The starting point is the C and C++ hosted library. We may require C functions that wg14 doesn’t include in the C freestanding implementation.

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

*We may be inside the OS, on bare metal, or in an “exotic” environment like a GPU. Filesystems, timers, and other facilities may be absent or unavailable.

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

Vision

Provide the (nearly) maximal subset of the C and C++ hosted library that does not require OS interaction or space overhead

*Zero **space** overhead

*You don't pay for what you don't use

*Leave no room for another language

Freestanding Goals

- Give library writers the tools to write portable libraries for freestanding environments
- Avoid standardizing portability traps as part of the freestanding library
- Standardize existing practices
- Avoid warnings from hosted portions of the library

Freestanding Goals (pt 2)

- Have a list of freestanding functionality that developers can reference, rather than needing to experiment or learn through hearsay.
- Provide a library for systems with 10 KB+ of RAM.
- Provide better error messages when a developer uses something unsupported that is outside of the freestanding library

Non-goals

- Mark entire headers as freestanding or not
  - i.e. Half of a header may be freestanding
  - Maybe the standard library modules can fix this
- Avoid dependencies on hosted C functions
  - i.e. memcpy isn't C11 freestanding, but we will use it and require its presence
  - Maybe wg14 will take the C portions of our list and adjust their definition of freestanding.

Non-goals (pt 2)

- Provide dynamically sized containers
  - I'm avoiding the heap
- Mark an entire class freestanding or not
  - In some cases, I want to be able to omit a small subset of functions from the hosted implementation.
- Expose all the library support for language features
  - I'm willing to leave this to freestanding extensions

Features to avoid

- Exceptions
- RTTI
- Global state
- Heap management
- Threads and thread-local storage
- Floating point operations
- Filesystem and console usage

New paper errata

- Remove error_category::message
  - Though that is virtual, so possibly a non-starter
- Remove boyer_moore and boyer_moore_horspool searchers
  - Internal heap allocations

Things that could be dropped

- System_error and to / from_chars
  - message() functions drag in std::string
- <cwchar> functions
  - These are easy to implement, but uncommon to use in freestanding environments
- <random>
  - High implementer burden
- <execution>
  - No algorithms that depend on it

Things that could be added

- <chrono> minus *_clock::now()
- Floating point support
  - Users can avoid it where it doesn't make sense
  - Portability trap though
- Errno and string functions that rely on it
- <exception> and <typeinfo>
  - Would generally be unused, or a portability trap
- complex<integer>