Transitioning from uclibc to musl for embedded development

Embedded Linux Conference 2015

Rich Felker, maintainer, musl libc

March 24, 2015

What is musl?

musl is a libc, an implementation of the user-space side of the standard C/POSIX functions that are the foundation of most systems.

musl is a *general-purpose* libc. Unlike uClibc, it's not specific to the embedded domain.

History and Motivations

- Based on work that begin in 2005, seeking an alternative to glibc bloat with strong UTF-8 support.
- uClibc was an emerging option, but lack of stable ABI made it unattractive.
- Project really launched 2010-2011.
- Milestone 1.0 release in 2014.

Core Principles

Not all chosen from the outset, but evolved:

- Simplicity as the core approach to size, performance, security, and maintainability
- Factoring for minimal code duplication
- Ease of navigating and understanding code
- Robustness/fail-safety
- Not depending on fancy compiler/toolchain features
- First-class status for UTF-8, non-ASCII characters

Motivations for switching from uClibc to musl

Three major areas:

- Technical advantages
- Project health & development process
- License

License

uClibc is LGPL. musl is under a permissive (MIT) license:

```
Copyright © 2005-2015 Rich Felker, et al.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

Permissive license means you can make static-linked binaries without license-conformance concerns.

Project Health

- Time-based releases every 1-2 months
- Tracking current standards
- Rapid turnaround for bug fixes
- Responsive mailing list and IRC channel
- Stable ABI

uClibc's declining health

- Almost 3 years with no official releases
- 3 options for threads, all outdated & buggy
- Numerous broken configurations
- Major C99 and POSIX 2008 features missing
- Buildroot considering switching away

Technical benefits of musl

Quantitatively

	musl	**uClibc**
Source code size	~48kloc	~230kloc
Library binaries	~500k	~500k
Minimal static binary	1.8k	7k
Minimal `printf` static binary	13k	70k
Minimal threaded static binary	6k	114k
Dynamic linking overhead	20k	40k
UTF-8 performance	4x glibc	2x glibc

(Sizes vary by arch; measured on x86.)

Fail-safety

musl does not introduce unnecessary failure cases.

Operations that can be performed in-place or in small bounded space without resource allocation never fail.

After `main()` is entered, all failures are reportable. musl will never `abort()` behind the program's back.

No lazy binding or lazy TLS allocation.

Advanced `posix_spawn()`

The `posix_spawn` function is like `fork+execve` in one.

Avoids all the dangers of `vfork` (ewontfix.com/7/).

musl's `posix_spawn` implementation uses `CLONE_VM` and close-on-exec to synchronize with child's `execve`.

Compatible with NOMMU and optimal for low-memory environments.

Advanced threads implementation

- Lightweight - around 10-15k total.
- Supports C11 and POSIX threads APIs.
- Safe-to-use thread cancellation (ewontfix.com/2).
- Strong adherence to POSIX and C11 semantics.
- Available on all supported archs.

`iconv` charset conversions

musl's `iconv()` supports most major legacy character encodings, including legacy CJK & GB18030.

All in 128k of code & tables.

Important Differences

musl is *not* configurable

And that's a good thing.

For static linking, efficient factorization of object files gives most of the same benefits as configurable features would, but without the configuration burden on the user (you).

As a result, testing is practical and we don't have continually breaking feature combinations.

musl supports fewer archs

But it's easy to port.

Supported by both uClibc and musl

i386, x86_64, ARM, PowerPC, MIPS, Microblaze, SuperH

Supported only by uClibc

Alpha, AVR32, Blackfin, c6x, Cris, HPPA, Itanium, m68k, Nios, Sparc, Vax, Xtensa

Supported only by musl

AArch64, OpenRISC 1000

And hopefully (GSoC) RISC-V!

What's involved in a port?

- 12 mandatory asm files (~200 lines)
- 5 mandatory arch-def headers (~150 lines)
- 27 `bits/` headers defining types/kernel interfaces
- Small build-system glue
- Optional optimized versions of bottleneck functions

musl doesn't use glibc headers

uClibc uses (outdated, modified) copies of the glibc headers and defines `__GLIBC__` to “trick” applications.

musl has its own clean-room headers.

musl's headers do not depend on any kernel headers, but may conflict with some uses of kernel headers.

musl is only one lib file

Threads, math, `clock_gettime()`, etc. are always available without needing `-lm`, `-lpthread`, etc.

Even the dynamic linker is integrated.

Empty `libm.a`, `libpthread.a`, etc. are provided for build-time compatibility (and conformance).

There is no `libm.so`, `libpthread.so`, etc.

musl behaves differently

In some ways.

But usually they're good, once you understand them.

Dynamic linking

- Always `RTLD_NOW` behavior (no lazy binding).
- Dynamic TLS is reserved at `dlopen` (no lazy allocation).
- Loaded libraries are never unloaded (by `dlclose`).

As a result, most archs have **zero** lines of arch-specific dynamic-linker code.

Threads

- Default thread stack size is small (80k vs 2-8 MB).
- Thread cancellation doesn't interact with exceptions.
- Dynamic TLS is reserved at thread creation.

Locale and charset

- Character encoding is always UTF-8 (even C locale).
- Character properties are hard-coded to Unicode, not locale-specific and not generated from glibc locales.
- `iconv` supports different (mostly, more) charsets and may behave differently.

Further misc. differences

- Math functions don't set `errno`, only `fenv` flags.
- DNS lookups are done in parallel.
- Regex implementation has different/fewer extensions.

Toolchains & Distributions

Canonical toolchain is musl-cross

https://bitbucket.org/GregorR/musl-cross

These are the patches intended for gcc upstream and will eventually make it there.

Simple musl-targeted cross compiler build scripts are included. Precompiled x86 binaries available.

Buildroot supports musl

- Well-known to uClibc users.
- musl is an option on the toolchain menu.
- Still labelled “experimental”.

musl-based distributions

- Sabotage Linux - the original musl-based dist and patch-source for packages that don't build against musl out-of-the-box.
- OpenWRT - supports musl-based builds; plans to switch default to musl.
- Alpine Linux - server- and security-oriented distribution with binary packages for x86[_64] and ARM.
- Many more - see the musl community wiki.

Thank you

http://www.musl-libc.org

@musllibc, @RichFelker

https://www.patreon.com/musl