C Library

From OSDev Wiki

The C standard library provides string manipulation (string.h), basic I/O (stdio.h), memory allocation (stdlib.h), and other basic functionality to C programs. The interface is described in the C standard, with further additions described in POSIX as well as vendor extensions. On Unix platforms, the library is named `libc` and is linked automatically into every executable.

You need a C standard library implementation with the necessary features to run C programs on your operating system. C++ programs can usually use the C standard library as well and the C++ implementation is normally built on top of libc. It is possible to use the C standard interface in a kernel if the library implementation supports this.

Contents

Freestanding and Hosted

There are two flavors of the C compilation environment: Hosted, where the standard library is available; and freestanding, where only a few headers are usable that contains only defines and types. The hosted environment is meant for user-space programming while freestanding is meant for kernel programming. The hosted environment is default, but you can switch to the freestanding by passing `-ffreestanding` to your compiler.

The `__STDC_HOSTED__` macro expands to `1` on hosted implementations, or `0` on freestanding ones. The freestanding headers are: `<float.h>`, `<iso646.h>`, `<limits.h>`, `<stdalign.h>`, `<stdarg.h>`, `<stdbool.h>`, `<stddef.h>`, `<stdint.h>`, and `<stdnoreturn.h>`. You should be familiar with these headers as they contain useful declarations you shouldn't do yourself. GCC also comes with additional freestanding headers for CPUID, SSE and such.

Implementations

There are several open-source C library packages available, and using one may be a viable solution for you. All of them will require some degree of modification to suit your needs.

There is a comparison table of some of these at: [1] (http://www.etalabs.net/compare_libcs.html)

MyOS libc

> *Main article:* Creating a C Library

The best option, in terms of code quality and integration with your operating system, is to write your own C library. You can aim for making a clean and high-quality implementation that follows standards well. It will integrate cleanly with your kernel as no portability layer is needed. You can be secure and robust. You can surpass the limits of what you can do with existing implementations. You can add vendor-extensions that replace bad interfaces and do things better. You can break a lot of code because you followed the standards and applications didn't, then fix the applications as well. It can be better than the competition if you make that your goal.

This is the idealist path and most in the osdev spirit. It has a lot of advantages, but has the steep price of requiring a lot of effort. This may very well be worth it if your goal is to make a *good* operating system, not just *a* operating system. You don't have to be a libc expert when starting out, but you'll be one when you're done. You might just realize much of the competition out there isn't very good.

Creating your own minimal C library is relatively easy. Providing all the primitives needed by real program is a much larger task, but straightforward as you can attempt to cross-compile the software and implement all the missing features causing compilation errors.

Newlib

- The license is unrestricted (not GPL or LGPL), but each file likely has a different copyright notice.

- Requires threading, so is more appropriate for a runtime library
- About 400 functions supported
- newlib website (http://sourceware.org/newlib/)

GlibC

- GPL license
- Should be absolutely complete (even has all the bloat)
- glibc website (http://www.gnu.org/software/libc/)

Musl

- MIT or BSD licenses
- No kernel portability layer, uses the Linux system calls directly.
- A full set of math and printf functions
- Support for about 1200 functions
- Many system calls needs to be implemented as it assumes you are a full Linux
- musl website (http://www.etalabs.net/musl/)

PDCLib

- Creative Commons Zero license (public domain)
- Under active development, and not at full working release 1.0 yet
- Good for linking into kernels
- Support for about 120 functions, currently
- 10 (plus one optional) required syscalls need to be implemented
- No ASM -- should be fully portable
- PDCLib website (http://pdclib.e43.eu/)

uClibc

- LGPL license
- uclibc website (http://www.uclibc.org/)

diet libc

- GPL license
- dietlibc website (http://www.fefe.de/dietlibc/)

Google's Bionic

- BSD license
- No support for locales

- No libthread_db or libm implementation
- Its own smallish implementation of pthreads based on Linux futexes
- Support for x86, ARM and ARM thumb CPU instruction sets and kernel interfaces
- bionic website (https://github.com/android/platform_bionic/tree/master/libc)

Standards

Especially if you want to roll your own C lib, you may want to buy the ISO/IEC 9899 specification to work from. It is not free. Expect a PDF to cost somewhere around $250 (US) or 250 Swiss Francs, depending on currency conversions.

The older standards (C89/C90, C99) are not commercially available anymore. To find the current standard, go to one of the following sites and search for document "ISO/IEC 9899".

Specification Stores:

- From ANSI (http://webstore.ansi.org)
- From SAI Global (http://infostore.saiglobal.com/store)
- From ISO Store (http://www.iso.org/iso/home/store)

Retrieved from "http://wiki.osdev.org/index.php?title=C_Library&oldid=17023"
