The C standard library is too small

Sébastien Wilmet

April 2022

Introduction

When presenting a programming language, we can separate its *core language* from its *standard library*:

- The core language comprises the syntax, the keywords and the general features. Examples in C: how to write a new function, and how to create a `struct`.
- The standard library is usually provided in the form of a set of already-implemented functions or classes. Examples in C: the `strlen()` function, and the `FILE` type.

In C, both the core language and the standard library are small.

While the small core language of C can be seen as an advantage (it permits to quickly master the major features), the small standard library can be seen as a disadvantage, because it is *too* small (it is usually not sufficient in order to write code comfortably).

1 The need to use an additional foundational library

Free software projects and companies have developed — over the years — additional libraries on top of the C standard library, that serve as a more complete foundation.

These additional libraries are not only useful to write large-scale pieces of software, but also for writing a small or medium-sized program.

In the free software world, there are at least a half-dozen of such libraries to choose from:

- GLib[1]: used by GTK, GNOME, some other free desktops, GStreamer, etc.
- ELL[2] (Embedded Linux Library): used by the Wi-Fi daemon iwd.
- APR[3] (Apache Portable Runtime): used for example by Apache httpd and Subversion.
- ZeroMQ[4] and CZMQ (High-level C Binding for ZeroMQ).

[1]`https://gitlab.gnome.org/GNOME/glib`
[2]`https://01.org/ell`
[3]`https://apr.apache.org/`
[4]`https://zeromq.org/`

- C-Util[5] (Common Utility Libraries for C11): used by dbus-broker.
- What systemd[6] provides, for example sd-bus and sd-event.

2 Ensuing problems

The additional foundational C libraries listed above are not standard, they widely differ in their APIs and style. And they can be quite large, so it takes time to get accustomed with one of them, to be comfortable writing code with it.

As a result, here are some problems that we see:

- When switching to a new job or community, if the foundational library is not the same, the programmer needs a lot of time to learn and to practice before being proficient, to know really well the API.
- Since the style differs as well (including the documentation), a programmer might not like using a different library.
- From a company's point of view, it's potentially harder to hire and train developers.
- It's even more complicated when juggling with several projects using different foundational libraries, because it's almost like using different programming languages. So doing consultancy work can be a hindrance.

Conclusion — the need to have a feature-full stdlib

For a programming language with a standard library that is large enough to be "batteries included", it doesn't cause the problems outlined above. Because since the library is standard, it is used across projects, companies and communities.

So when choosing a programming language, we believe that it's important to also look at whether its standard library contains enough features in order to write code comfortably.

Revisions:

- April 2022: first version, heavily inspired by a previous article that I wrote in 2021 and titled "*C dialects versus C++ dialects*".

[5] `https://c-util.github.io/`
[6] `https://systemd.io/`