Emilua: 1000 commits later

Recently the main repository for Emilua had accumulated 1000 commits in the branch `emilua-0.11.x`. That was enough to make me interested in writing a little retrospective.

Notable Emilua releases

0.4

That's certainly one of the releases that took the most to finish. I was convinced that the release could not happen before the new Antora-based documentation infrastructure was in place, and that took a while to finish (there was a gap of two years without any release until the new documentation infrastructure was ready).

In the time it took for the new documentation infrastructure to be ready, the main code repository increasingly accumulated the features that culminated in the new release.

The new release introduced the type `byte_span` which became the basis for all IO operations in Emilua. The inspiration for `byte_span` originally came from an old small unrelated experiment by Justin Cormack.

That's also the release that introduced support for Linux namespaces (plus subprocess-based actors). It required a ton of reviews, refactors, and tests, but that's what eventually enabled so many unique features that appeared in later Emilua releases.

0.5

It required only a few months to finish, but a ton of mostly refactoring-related work went on to reach this point.

- Almost all references to Linux-specific code in the subprocess-based actor support had been removed, and the code now worked on FreeBSD as well. Linux's pidfd and FreeBSD's procdesc are too far apart, and it ain't easy to unify their behavior. All-in-all, I still prefer FreeBSD's procdesc to this day. It's a shame that Linux decided to invent its own (inferior) solution to this problem, but I still secretly hope they come around someday.
- A lot of refactoring was also done in the module-resolution layer to unify more details between `spawn_vm()` and `require()`. There's a common saying that lazy people make good programmers, but I cannot bring myself to imagine lazy people doing boring refactors that are essential to polish a project all-around. And let me tell you: boredom will definitely come to greet you! Programming **is** serious business. Programming isn't something you should be doing for "kicks".

For the end-user, it was most definitely a boring release with little new features, but the future work enabled by this release was nothing of short.

0.11

This is the release that refactored the project build as a whole. The project is now further split into even more libraries. To mention just two of the them:

libmain

A framework to generate C's `main()`. It's used by the new generators to create binaries that include the application's Lua source code files instead of querying the filesystem at runtime.

libc-service

Overrides functions from libc related to ambient authority. This is used for advanced sandboxing needs. It works on Linux and FreeBSD.

The module system for builtin modules advanced almost as much as I had envisioned when I started Emilua, and it's not far from done now. This support required me to refactor a lot of code, and to do plenty of testing on different OSes to figure out what would work. There are still a few bugs, but hopefully not many.

The support for builtin modules has progressed so much that some modules were removed to be developed as separate plugins. The plan was to do this even for TLS, but even further refactoring would be needed, and the extra task was then postponed to a further release.

The integration between the actor system and the module system also has been further tightened, and it has been extended to cover advanced sandboxing needs as well. Looking at old code to see where they can be improved instead of spending time into new features may not be as fun, but it's many times needed work. That's what happened here.

More code reviews were done to ensure the system will work well for set-uid binaries. More API details were fleshed out to make sure new cases of sandboxing could be tackled. Many small details were reviewed all-around. A lot of work went on to make this release happen. This is one of those releases that bring 1.0 closer to reality.

I no longer see improvements that I wish to implement in the builtin facilities related to sandboxing in Emilua. It's all here already.

Roadmap to 1.0

There's not a lot that needs to be done for 1.0, but the few things that need to be done are really annoying blockers.

- TTL modules need to be expanded to allow one to import resources into the binaries as well. The intention is to improve deploy. This can't happen yet as serd isn't a friendly parser to work with (no push parser is). I should just give up and write my own TTL parser at this point, but I'm busy doing other stuff.
- Remove the module TLS to be developed as a separate plugin. There are some branches with initial work on this front, but a big problem is a split in ASIO for two not perfectly compatible worlds — Network TS executors and standard executors. Emilua is written for the old world (network TS executors), and can't make use of ASIO facilities related to polymorphic completion

handlers that could cross a plugin ABI.

In short, you may not see 1.0 anytime soon. The good news is that these tasks aren't related to API breakage so you won't need to do a lot to upgrade to 1.0 when it finally happen.

Neighbors at the open source community

Emilua brought many of the APIs that it uses to new limits that helped to give them some new small directions. To mention just a few:

- https://reviews.llvm.org/D105758
- https://github.com/skvadrik/re2c/issues/343
- https://savannah.gnu.org/bugs/index.php?64393
- https://reviews.freebsd.org/D47351
- https://reviews.freebsd.org/D43448
- https://reviews.freebsd.org/D47361
- https://reviews.freebsd.org/D44867
- https://reviews.freebsd.org/D42780
- https://reviews.freebsd.org/D42846
- https://bugzilla.kernel.org/show_bug.cgi?id=218607
- https://github.com/shadow-maint/shadow/issues/635
- https://gitlab.gnome.org/GNOME/glib/-/merge_requests/1960

Many more improvements also hit ASIO alone. To mention a few:

- https://github.com/chriskohlhoff/asio/commit/111afa1f5859b15d6c46c6ae991c1f684a0262f3
- https://github.com/chriskohlhoff/asio/commit/50ad5e47793521a213eb461da701e8ddd3d5c70d
- https://github.com/chriskohlhoff/asio/commit/e7e24b5c3231fdef1ced094f39b9eb9dfb491686
- https://github.com/chriskohlhoff/asio/commit/3da0ef20fa864980ca226fdd4b846999b4caae2a

Truth be told, ASIO is still not perfect (specially on FreeBSD/Capsicum). What I still would like to see:

- Prefer `MSG_DONTWAIT` for sockets. If `MSG_DONTWAIT` is available and you're working on socket objects, ASIO shouldn't at all deal with `FIONBIO`/`O_NONBLOCK`. `O_NONBLOCK` is a shared state, and it plays poorly with sandboxing.
- Add support for POSIX AIO on top of kqueue (`SIGEV_KEVENT`) for file IO. Strangely as it may be, that's also important for Capsicum (specially when dealing with non-socket objects). Of course this only would be available on FreeBSD, but other BSD systems all suck anyways and there's no reason to even consider them.
- File IO on ASIO lacks `async_sync_data()` and `async_sync_all()`.

Lessons learned

This isn't a postmortem, but I'll try to document a few lessons so I don't forget them in the far future.

Vendoring

I've hit a few hard blocks these last years because it takes too much time until a new version of a dependency reaches all relevant distribution channels. Playing nice with the open source community can sometimes bring a project to crawl.

If you don't control your supply chain, you don't have a say when a problem hits you (and only you). Would you believe that I'm still waiting for a 1-line bugfix patch to be accepted by GCC folks several years after it has been sent already[1]?

I learned the value of vendoring. Sometimes vendoring is — like it or not — necessary.

Manual binding generation is obsolete

Bootstrapping an entire ecosystem from the scratch is a cruel barrier to hit. If there are no alternate roads for automatic binding generation in your project, failure will most likely be certain.

What the folks at the Carbonlang are doing by being compatible with C++ from the start is smart.

Community building isn't free

You need to do active work to actually build a community. A community just doesn't pop out out of nowhere. At the same time, it's hard to measure the value of one. We like to imagine that a community is always important, but let me say something polemic for once: is the community you built really of value or just a drag on your time?

What I can say is this: if a very talented individual show up, it's wise for you to be grateful and do whatever it's in your hands to keep such talent in your community. Do not make the mistake to think that talented individuals are easily replaceable. You may never fill the void again.

The Lua community isn't ready for execution engines

The Lua community is culturally inapt to embrace execution engines. This type of project just doesn't fit in the model that members of the community reason about problems.

Going forward, I won't start new projects for the Lua community. Emilua is enough.

Don't ignore marketing

It's not enough for a new project to do stuff that your competitors can't. You need more.

The future

Emilua was immensely helpful as a playground for concurrency experimentation. It’d be a waste to not make use of the insights discovered along the way. A new project based on these findings is being developed behind locked doors, but as soon as a good demo is ready, a public announcement with lots of details will be made. Until then, the pace in public repos will slow down a lot. See you in a few months.

[1] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=98449#c2