LibreOffice WASM - the How and Why

A status report from the journey to get LibreOffice into the browser, fully*

WASM & other neat hacks to make that Happen

FOSDEM virtual conference, 2021-02-07

Jan-Marek Glogowski - glogow@fbihome.de

Thorsten Behrens – thb@libreoffice.org

The State of the Art

Currently (LOOL/COOL):

- HTML5-canvas based browser version
- lightweight, tiled rendering
- the heavy lifting happens on the server
- all documents of all users loaded there
- all rendering & editing happens in the *data center*
- Pros:
  - light on the client
  - documents stay on-premise
  - ~easy collaborative editing – just *one* document instance to keep up-to-date
- Cons:
  - no offline mode
  - expensive to host
  - no peer2peer editing, or end2end encryption possible

Pain points of LOOL's architecture

- price of hosting
- cost of operations - not trivial to host & scale
- noticeable costs per user, if you want to run it planetary-scale
- code & technology - two separate repos, but cross-cutting changes often required

So what to do?

Idea: LibreOffice WebAssembly - lets call it LWA henceforth!

- looking at the trajectories of hardware (mobile/laptop)
  - your phone: CPUs with 8 core, up to 2GHz; 12GB RAM on the high-end
  - Ultrabooks with 32GB and 12-thread i7... – port the core to a new architecture! the new platform is … the browser! i.e. WASM – compile native code to run in your browser; W3C standard since end of 2019 – use a WASM core - LibreOffice cross-compiles to WASM (like we do for Android, iOS, Windows ARM etc)
  - use platform APIs whereever feasible (crypto, IO, network) for speed & weight reasons

Architecture & battle plan

- but, we tried that – it didn’t work?!
- we gave up, as in 2015 emscripten/WASM couldn’t even do exceptions properly
- stars are aligned now - WASM is W3C standard, with wide browser support
  - nothing missing really anymore (except perhaps threading)
  - SharedArrayBuffer currently disabled due to Spectre (but hey..)
- What needs doing? low-level cross building port big blobs to use browser APIs (NSS, I look at you!)
- strip down the monolith (target only Writer for a start)

The technical challenges

And the evolving landscape

WASM

- 2GB limits - size of the binary likely not feasible to load 100MB of WASM & survive
- 32bit address space
- browser disabled SharedArrayBuffer after Spectre exploits
- WASM thread support still being cooked: https://github.com/WebAssembly/threads
- default-off in browsers, but can be turned on
- https://github.com/emscripten-core/emscripten/wiki/Pthreads-with-WebAssembly
- so - for the moment, use WebWorkers/ServiceWorkers and message passing, if you need multithreading...
- then again, Writer is single-threaded since 1990

Hacking tools & resources

- https://anonyco.github.io/WasmFiddlePlusPlus/ for playgrounds
- or this one: https://mbebenita.github.io/WasmExplorer/ & https://webassembly.studio/
- https://webassembly.studio/
- https://developer.mozilla.org/en-US/docs/WebAssembly/Using_the_JavaScript_API
- some debugging support in FireFox: https://www.youtube.com/watch?v=R1WtBkMeGds
- best experience so far in Chromium: https://developers.google.com/web/updates/2020/12/webassembly

Current WASM status

- https://wiki.documentfoundation.org/Development/WASM
  - boils down to: feature/wasm + README.wasm
- emsdk and qt5 WASM is integrated with gbuild
- static dependency resolver (solenv/gbuild/statics.mk)
- few binaries (vcldemo, ui-ppreviewer) are build, so are some cppunit tests

“Everything” builds but nothing runs yet, except for: wasm-qt5-mandelbrot (demo)

Major emscripten, gbuild and LO problems

Looking into major problems

- gbuild: linking static executables
- LO: code dependency loops
- LO: static UNO components
- emscripten: no CPU limit for wasm-opt

- some problems still to tackle

  (+ all the still unknown stuff)

gbuild: linking static executables

- All link info private to the gbuild link targets
- No transitive information available
- Android and iOS: bin/lo-all-static-libs
  - just one binary; collect and throw everything to the linker

Implementation:

- Register all dependencies per linktarget in variables
- traverse the dependency tree to fill all libraries, externals and statics
- cache them, so "make <module>" will still work

LO: code dependency loops

- Normally loops are broken up by using dlopen
  - sc <=> scui, sw <=> swui, vcl <=> vclplug_* + filters/gie, sal <=> sal_textenc

Implementation: loader + plugin concept

- plugins register with a loader library
- executables must link to plugins, if they link to loaders
  - needs a dynamic dependency, just if the loader library dependency exists in the tree
  - done while traversing the dependency tree for static execs

LO: static UNO components

- dlopen'ed and instanciated dynamically
- emscripten can’t use dlopen and pthreads together

Implementation:

- A map of component contructors with symbol names
  - same as for Android and iOS => solenv/bin/native-code.py
- WASM specific:
  - libcomponents with dependencies on all component libraries
  - Add libcomponents to all cppuhelper users, as this calls into the "native code" symbol map table

emscripten: no limit for wasm-opt

- currently "stuck" with Qt supported emscripten 1.39.8
- em++ doesn't support -Wl,-O1
  - wasm-opt runs for minutes per binary
  - wasm-opt uses threads for all cores

Implementation:

- All binary linking is serialized
- em++.py patched to forward -Wl,-Ox

Some problems still to tackle

- Link time is still much too long
- No nested main loops / blocking the browser
  - You can run the main loop in a web worker
- Building a virtual FS image to access data
- Debugging seems to be a pain

- Recommended + more problems: “AutoCAD & WebAssembly: Moving a 30 Year Code Base to the Web”
  - https://www.infoq.com/presentations/autocad-webassembly/

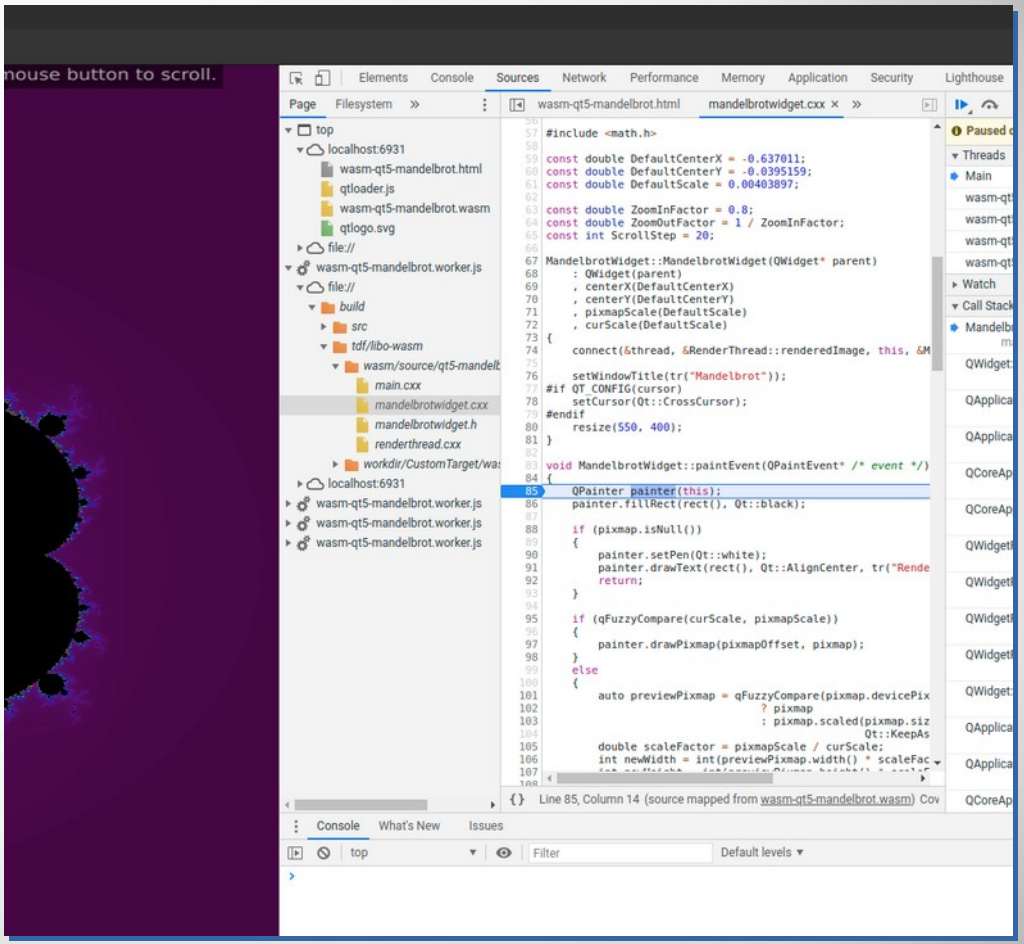
Small Demo

Build setup

- from scratch:
  - start from here - README.wasm
  - setup emscripten (around 1GB), qt5 (only for the demo, and needs around 10GB)
  - setup LibreOffice cross build (*strongly* recommended to take the configure line from the README currently)
  - this is bleeding edge, please don't expect the branch to build all the time - poke us (`thorsten`, `jmux` on irc: chat.freenode.net on #libreoffice-dev)
- run it (e.g. the demo): `$ emrun --serve_after_close workdir/LinkTarget/Executable/wasm-qt5-mandelbrot.html`
- further info - the excellent MDN article: https://developer.mozilla.org/en-US/docs/WebAssembly

Debugging

- build with `--enable-dbgutil`
- best experience currently: Chrom{e|ium}



Project plan & timeline

- hope to have vcldemo running “real soon now”
- cut LibreOffice down to minimal Writer, port the largest 3rd party libs (icu, nss) to browser APIs
- either use Qt5, or WSD to render Writer on a HTML5 canvas interactively (by summer 2021)
- get a demo End2End encrypted editing session going by autumn 2021

Q & A and credits

Thx a lot to NLnet & allotropia for sponsoring!

www.allotropia.de/