Twitter Thread by Richard Geldreich

Richard Geldreich
@richgel999

Porting vanilla (minimal external API usage, no SIMD, no threading) C++ code to work in the browser with emscripten and WebAssembly is amazingly simple now. The Emscripten docs are excellent, but here are some things that could speed up the process of getting started:

1. Beware of code which does unaligned memory reads/writes (which "may be slow on some CPU architectures")
2. Test with -fsanitize=undefined -fsanitize=address
3. Link with "-s ALLOW_MEMORY_GROWTH=1" and "-s INITIAL_MEMORY=X", X is a multiple of 64k, allows the C++ heap to grow.

(I completely avoid unaligned memory reads/writes because when compiled to asm.js they "can fail silently".)

4. Link with "-s MALLOC=emmalloc" to reduce compiled size.
5. I usually test with -O1 because -O0 can take very long to load/execute.
6. For debugging link with -s DEMANGLE_SUPPORT=1 and -s ASSERTIONS=1

7. C++ printf() outputs to the Developer Console: Chrome Settings->More tools->Developer tools
8. If something crashes, try running in Firefox which may explain the error differently.
9. I use Web Server for Chrome for development:
https://t.co/3drLehhSlL

The C FILE I/O functions work, but on a virtual file system:
https://t.co/iyJyAAVBiu

You can package up individual files from a directory that gets preloaded before your module executes.

To wrap C++ functions, classes, types etc. to expose them to Javascript code, look for "EMSCRIPTEN_BINDINGS". It's quite slick and mostly automatic.

Reading an integer property on a Javascript object from C++:

const emscripten::val& srcBuffer;

unsigned int length = srcBuffer["length"].as();

Copying from a Javascript Uint8Array to a std::vector:

And the reverse: copying from a std::vector to a Javascript Uint8Array (which was sized in Javascript to be >= the size you want to copy):

To JavaScripte/WebAssembly wizards all of this is probably the most basic beginning stuff. However, to native C++ coders not very familiar with these technologies (like me), all of this takes a while to figure out.

I use "emcmake cmake ../" in my build directory, with my CMakeLists.txt file in the parent folder. It sets environment variables that CMake needs to use emcc (the compiler).

All the heavy lifting that I do (data compression, decompression, transcoding)- stuff that doesn't require external API access - is in C/C++. Pretty much everything else, like the UI, WebGL usage, etc. is written in Javascript.

WebAssembly SIMD is the tech you want to use for SIMD in the web world. Personally, I use the intrinsic function wrappers. Functionality wise it seems vaguely somewhere in between SSE v2 and v4.

Multithreading support is still the wild west. I wouldn't depend on it being available yet in all browsers: https://t.co/hyK6E6Pvso

#emscripten #webassembly