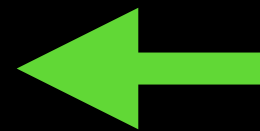


An Introduction to Emscripten

Running C/C++ in a web browser!

Slides available at: <http://bit.ly/dllemnt1>

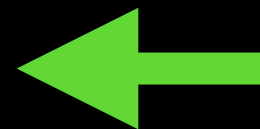


Presented by

- David Ludwig
- 15+ years C/C++ experience (arguably!), mostly in games
- <http://dll.fyi>
- Twitter: @DavidLudwig



Slides available at: <http://bit.ly/dllem1>



What is Emscripten?

Emscripten is a compiler, toolchain, and runtime library for running C and C++ code in web browsers (or Node.js), without plugins.

(This talk will focus mainly on code run within a web browser.)

What is Emscripten?

- it is a C/C++ compiler and toolchain
 - Clang-backed by default
 - also accepts LLVM byte code as input
- Outputs:
 - JavaScript
 - WebAssembly (a newer format)
- it offers a runtime, with C and C++ APIs to help work with web browsers, or NodeJS
- has many limitations over popular, 'native' C/C++ platforms, however many things are possible (and demonstratable)

Demo Time

Qt

[http://example.qt.io/qt-webassembly/Slate/
slate.html](http://example.qt.io/qt-webassembly/Slate/slate.html)

Installation

- instructions at https://emscripten.org/docs/getting_started
- Option 1: Emscripten SDK (aka, “emsdk”)
 - capable of installing and updating many, but not all, dependency technologies
 - works on Windows, Linux, and macOS (at least)
 - takes a few steps (documented at emscripten.org)
- Option 2: via a 3rd-party package manager
 - HomeBrew (aka. “brew”), for macOS: <https://brew.sh>
 - apt-get, for Ubuntu, Debian (at least)

Basic Usage

“emcc” and “em++”

- comparable to other compiler commands (mostly)
 - i.e. gcc/g++, clang/clang++, cc/c++
- Examples:
 - `em++ foo.cpp -o foo.html`
 - `em++ bar.cpp baz.cpp -o barbaz.js`
 - `em++ biff.cpp -o biff.html -s USE_SDL=2`

Libraries + APIs

(among others)

- stdio (printf, fopen, ...), iostreams (std::cout, std::fstream, ...)
- POSIX-style file system APIs (opendir, readdir, stat, ...)
 - no std::filesystem, yet
- graphics: OpenGL ES 2+, DOM, LibSDL, among others
- audio: OpenAL, LibSDL, among others.
- input (via events): keyboards, mice, touch
- partial support for sockets (via WebSockets; IO must be async)

Example #1

Hello World, printf edition

```
// Compile with: em++ hello_world.cpp -o hello_world.html
```

```
#include <stdio.h>
```

```
int main() {
```

```
    printf("Hello World\n");
```

```
}
```

Example #2

Hello World, inline JavaScript

```
// Compile with: em++ hello_world.cpp -o hello_world.html
```

```
#include <emscripten.h>

int main() {
    EM_ASM((
        var greeting = "Hello World"
        alert(greeting)
    ));
}
```

Talking to Web APIs

- *ALL* interaction with system is done through Web APIs (DOM, WebGL, etc.)
- inline JavaScript
 - very-basic data-interop is present
- export C/C++ functions as JavaScript/Web-API-callable functions
- higher level libraries
 - Embind — C++11 wrappers
 - wrap JavaScript values as C++ objects
 - wrap C/C++ structs and classes in JavaScript objects
 - call JavaScript functions with C++-style syntax

Example #3

Hello DOM (Document Object Model), via Embind

```
// Compile with: em++ --bind hello_dom.cpp -o hello_dom.js
// Uses a pre-made html file (i.e. "hello_dom.html"), containing the following tag:
// <div id="my_element"></div>
```

```
#include <emscripten/val.h>
```

```
using namespace emscripten;
```

```
int main() {
```

```
    val my_element = val::global("document").call<val>(
        "getElementById", val("my_element")
    );
```

```
    my_element.set("innerHTML", "Hello DOM!");
```

```
}
```

Demo Time

live-video processing (with CCV)

<https://fta2012.github.io/ccv-js/>

Callbacks

- browsers often run JavaScript functions run in a cooperatively-multitasked, single-threaded environment
 - i.e. `'while (true) {}'` will hang a web browser!
 - functions (may) only get a few seconds, before the browser takes action
- long running programs might need to be broken up into short-running time-slices, run via browser-registered callback functions
- browsers can invoke callback-functions on...
 - periodic time-intervals
 - various notable events (input, async IO, etc.)

Example #4

Run Indefinitely

```
#include <emscripten.h>
#include <emscripten/val.h>
#include <string>
using namespace emscripten;
```

```
int update_count = 0;
void update() {
    val my_element = val::global("document").call<val>(
        "getElementById", val("my_element")
    );
    my_element.set("innerHTML", std::to_string(++update_count));
}

int main() { emscripten_set_main_loop(update, 0, 0); }
```

```
// Compile with: em++ --bind run_indef.cpp -o run_indef.js
// Uses a pre-made html file (i.e. "run_indef.html"),
// containing the following tag:
// <div id="my_element"></div>
```

Caveats

- thread support often unavailable
 - use callbacks, instead
 - async IO options are available
- limited socket support
 - via WebSockets, which are ‘reliable’ in a manner similar to TCP
 - incomplete + experimental, non-reliable/UDP-like sockets, via WebRTC, do exist
- step-debugging not fully featured
 - partially-possible (in a web browser, via ‘source maps’)
 - reading/watching variables still in development (CyberDWARF)

Caveats

- limited library support
 - often due to other caveats (such as threading restrictions)
- keyboard shortcuts may be overridden by web-browser(s)
- setjmp/longjmp not possible, nor any low-level stack manipulations
- can (still) be slower than native code

Demo Time

in-browser gaming (via Doom 3)

<http://wasm.continuation-labs.com/d3demo/>

The End

- Your Presenter: David Ludwig
 - Twitter: @DavidLudwig
 - <http://dll.fyi>
- Are there any questions?

Slides available at: <http://bit.ly/dllem1>

