# RUNNING COMPILED CODE ANYWHERE WITH WEB ASSEMBLY AND WASI

## CDG MEETING 21ST NOVEMBER

Jonathan Hollocombe

jonathan.hollocombe@ukaea.uk

# WHAT IS WEB ASSEMBLY?

WebAssembly (aka WASM) is a binary instruction format similar to regular assembly (i.e. x86) but targetting the web.

WebAssembly is an open standard, created with the following goals in mind:

- Be executed at near-native speeds.
- Be readable and debuggable.
- Be secure.
- Interact with the rest of the web ecosystem.

# WHAT CAN IT DO?

- Compile high-level languages (C, C++, Rust, etc.) to run in the browser
- Run high-performance and high-resource applications on the web (games, image editing, visualisation, etc.)
- Render applications with WebGL without interacting with the DOM
- Run compiled server-side applications with node.js
- Enable sandboxed portable applications using WASI

# WHAT DOES IT LOOK LIKE?

WebAssembly comes in 2 formats:

- .wasm: the binary assembly which is compiled from the high-level language such as C, C++, Rust, etc. and sent to the browser
- .wat: the human readable representation of WASM that is diplayed by the browser when debugging

# EXAMPLE

## add.c

```c
int add(int first, int second)
{
    return first + second;
}
```

## add.wat

```
(module
  ;; some global exports not shown here
  (func `$add (;1;) (export "add") (param $`var0 i32) (param $var1
    local.get $var0
    local.get $var1
    i32.add
  )
)
```

# HOW TO GENERATE IT?

- WebAssembly can be used with any LLVM based compiler by specifying the `wasm32` target
- For simple code (such as the previous demo) this can be done simply:

```
clang --target=wasm32 file.c -o file.wasm
```

- For most codes you'll need access to the standard libraries (`libc`, etc.) so will need to use `emscripten`

```
emcc file.c
```

# HOW IT RUNS?

- WebAssembly requires a stack-based virtual machine to run
- This VM runs the compiled WASM module, providing an ArrayBuffer or SharedArrayBuffer that provide the memory space used, and interfaces to I/O, threads, exceptions, etc.

# HOW TO RUN IT?

- WebAssembly can be run using:
  - A browser with JavaScript glue code to instatiate the WASM module
  - node.js with JavaScript glue code to instatiate the WASM module
  - A WebAssembly runtime such as `wasmtime` or `wasmer`

# DEMO 1: SIMPLE FUNCTION

```
int add(int first, int second)
{
    return first + second;
}
```

```
clang --target=wasm32 --no-standard-libraries -Wl,--export-all -Wl
```

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <script type="module">
      (async() => {
        const response = await fetch('add.wasm');
        const bytes = await response.arrayBuffer();
        const { instance } = await WebAssembly.instantiate(bytes);
         window.alert('The answer is: ' + instance.exports.add(1,
      })();
    </script>
  </body>
</html>
```

# DEMO 2: EMSCRIPTEN

- Example of porting codes to WASM: https://github.com/emscripten-core/emscripten/wiki/Porting-Examples-and-Demos
- Pyodide: emscripten based Python
- Run demo using:

```
python3 app.py
```

# RUST TO WASM

- Rust is a popular choice for high-level WebAssembly language
  - No runtime, no exceptions, small standard library
- The Rust ecosystem has widely adopted WebAssembly - many libraries support wasm32 target
- This tends to make compiling from Rust easier than C or C++

# DEMO 3: SIGNAL VIEWER

- A very rough signal viewer application
  - Talks to a REST endpoint
  - Displays available signals
  - Plots 2D data where available
- Compiled as a desktop application:

```
cargo run
```

- Compiled as a browser application:

```
trunk serve
```

# WHAT IS WASI?

- The WebAssembly System Interface (aka WASI) is an API designed to provide OS-like features such as filesystem, sockets, clocks, etc.
- Browser independent and not dependent on Web APIs or JavaScript
- Runable in any WASI enabled WASM runtime such as `wasmtime` and `wasmer`
- Portable to any OS - WASI abstracts the OS from the WASM compiled code

# DEMO 4: IMAGE CONVOLUTIONS

This demo code is a simple image convolution tool that opens a file, runs a convolution over the data, and saves the file.

Native app:

```
cmake -Bbuild -H. -GNinja
ninja -C build
./build/convolution
```

# DEMO 4: IMAGE CONVOLUTIONS (PART 2)

WASM:

```
em++ main.cpp -fwasm-exceptions --preload-file photo.bmp -sALLOW_M
node a.out.js
```

WASM+WASI:

```
`${WASI_SDK}/bin/clang++ --sysroot=$`{WASI_SDK}/share/wasi-sysroot
```

# WASI-SDK LIMITATIONS

- No exceptions
- No threads - this is work in progress
- Had issues processing very large file
- I've haven't tried Rust WASI yet so need to try this to see if more is currently possible

# CONCLUSIONS

- WebAssembly is way for leveraging the power of compiled high-level languages to write tools that can run on the Web
- A lot of the tooling around WebAssembly in the browser is fairly mature, less so with running outside of the browser
- Rust compilation for WASM is easier than for other compiled languages

# CONCLUSIONS (CONT)

- WASI is a fairly recent addition and still in development - the tooling and features need to be expanded before it is truly viable
- WASM+WASI does have the potential for compiling sandboxed, portable code that can be run anywhere

# LINKS

Tools:

- https://webassembly.org/
- https://wasi.dev/
- https://wasmer.io/
- https://emscripten.org/

# LINKS (CONT)

Useful articles:

- https://thenewstack.io/what-is-webassembly-and-why-do-you-need-it
- https://scientificprogrammer.net/2019/08/18/what-the-heck-is-webassembly

Demonstration code:

- https://github.com/jholloc/wasm_wasi_talk

**Solomon Hykes**
@solomonstre · **Follow**

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

> **Lin Clark** @linclark
> WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...
>
> 📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)
>
> hacks.mozilla.org/2019/03/standa...

8:39 PM · Mar 27, 2019

Read the full conversation on Twitter

2.1K Reply Copy link

Read 30 replies