Myth and Reality: How to port C/C++ application

Nikolay Khodov
16/07/2020

2. About me

- Full-Stack Software Engineer (node.js/Python/web)
- Focus on the frontend development
- Really like to try out new technologies
- [Disclaimer] Not a C/C++ rockstar (and never wanna be)

3. Structure

- My projects and goals
- Crash course a.k.a WASM/Emscripten 101
- Encountered issues and solutions
- Q&A

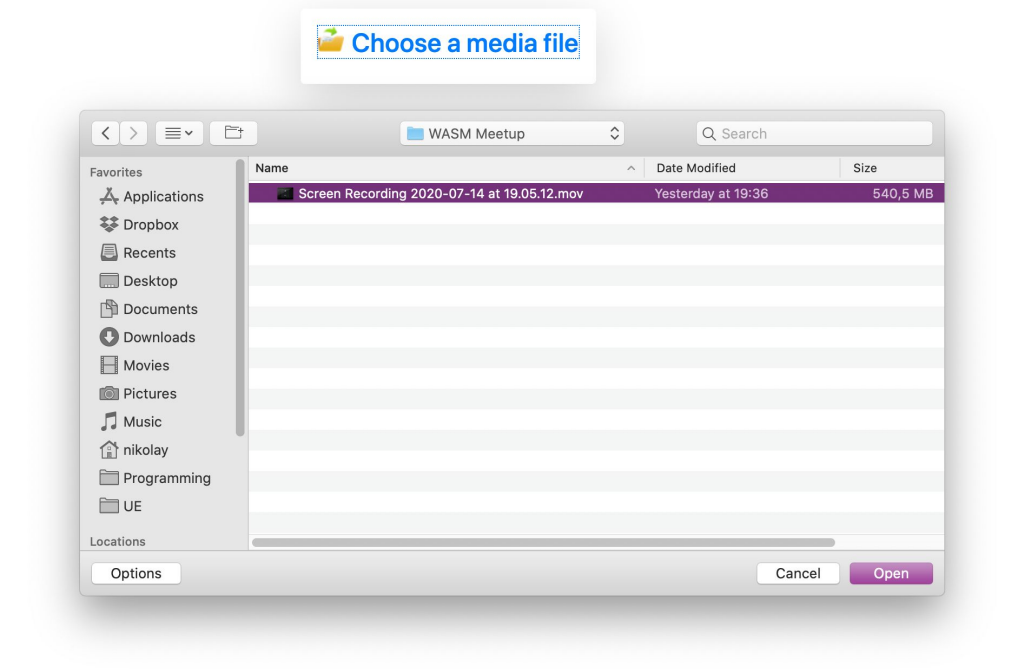
4. Describe the project

- Web-based utility to inspect media file metadata
- There are a lot of them...
- But developers like to reinvent the wheel

Let’s start...

- ffprobe - https://ffmpeg.org/ffprobe
- mediainfo - https://mediaarea.net/MediaInfo

5. Online metadata explorer: How it works?



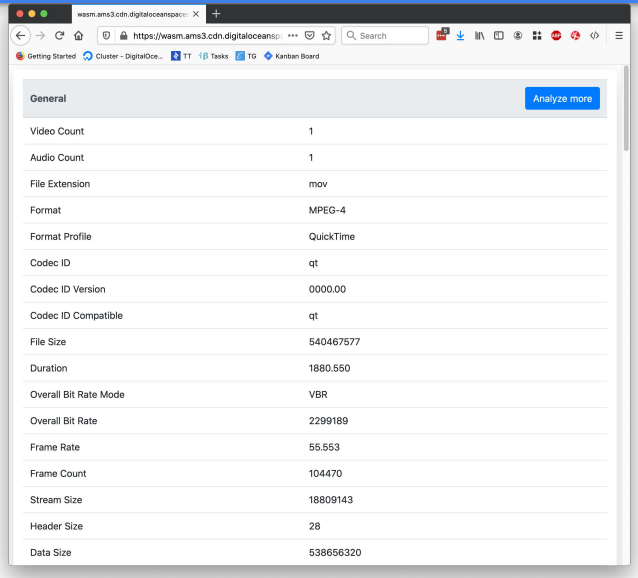
6. Online metadata explorer: How it works?

- Run mediainfo as a usual CLI tool in the web worker context

- ```
  module.callMain(["--Output=JSON", `/${file.name}`]);
  ```

- Grab its output and return to the browser context
  ```

7. Online medata explorer: how it works?



Video Count	1
Audio Count	1
File Extension	mov
Format	MPEG-4
Format Profile	QuickTime
Codec ID	qt
Codec ID Version	0000.00
Codec ID Compatible	qt
File Size	540467577
Duration	1880.550
Overall Bit Rate Mode	VBR
Overall Bit Rate	2299189
Frame Rate	55.553
Frame Count	104470
Stream Size	18809143
Header Size	28
Data Size	538656320

8. What the heck is WebAssembly/Emscripten

- Open standard
- Portable binary-code format for executable programs
- Sandbox/security/native performance

- LLVM-based compiler (IR)
- asm.js, WASM
- Based off C/C++ code

9. Fastcomp vs. Upstream

Fastcomp	Upstream
Fork of LLVM in 2013	Always up-to-date
Gonna be gone for good soon	New trend-setter
No modern features	All LLVM IR features

Upstream is far ahead in:

- Linking
- Bundle size
- Speed and performance

10. Emscripten fairytale (in theory)

```
$ emconfigure ./configure

$ emmake make

$ cp ./program ./program.o

$ emcc ./program.o -o program.js

$ ls -l program.js

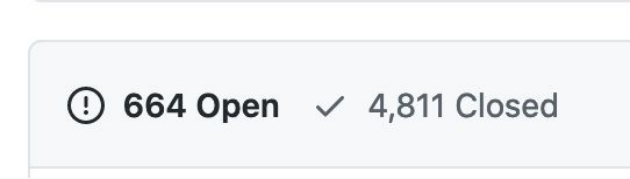
$ ls -l program.wasm
```

… and we're good

11. But in practice

12. What could go wrong?

- [Emscripten] API limitations
- [Emscripten] Environment + 3rd-party libraries
- [Emscripten] Cross-compilation
- [Emscripten] Repeatable builds
- [WASM] Bundle size
- [JS Wrapper] Handling files in the browser
- [JS Wrapper] Modularization



13. [Emscripten] API limitations

- Networking
  - Only async calls
- File systems (sandbox, JS wrapper)
- Conclusion: check what the app uses and be ready to make adjustments

14. [Emscripten] Environment + 3rd-party libraries

- All dependencies have to be compiled as static dependencies
- No closed-source libraries (no silver bullet to WASMify ‘em)
- You may need to fix your dependency versions (emsdk etc.)
- The most recent toolchain may have regressions

15. [Emscripten] Cross-compilation

- Zlib
- Compilation fails on MacOS: https://github.com/madler/zlib/issues/331

- Build on Linux and
- Be Docker with you, Luke!
- Con: Performance penalty
- Pro: Cached Docker images for fast builds

16. [WASM] Bundle size

- `wasm-opt` is your friend (easy-peasy to use)

- `wasm-opt -Oz ./src/worker/mediainfo-api.wasm -o ./src/worker/mediainfo-api.wasm`

- In theory it shrinks by **20%**: **5.2Mb** vs **4.8 Mb** (**~7%** effectively *for me*)

17. [JS Wrapper] Handling files in the browser

- Supported file systems:
  - MEMFS (client/server)
  - NODEFS (full access to the local filesystem)
  - IDBFS (browser)
  - WORKERFS
- No sync IO in the browser
- WebWorkers save us!

18. [JS Wrapper] Handling files in the browser

```
// use a custom read function

  node.stream_ops.read = function stream_ops_read(

    stream: { node: { blob: Blob } },

    buffer: Uint8Array,

    offset: number,

    length: number,

    position: number

  ): number { … }
```

19. [JS Wrapper] Handling files in the browser

```
const blob = stream.node.blob;
 if (position >= blob.size) {
   return 0;
 }
 const size = Math.min(blob.size - position, length);
 const slicedBlob = blob.slice(position, position + size);
 const data = new Uint8Array(fileReader.readAsArrayBuffer(slicedBlob));
 for (let index = 0; index < data.length; index++) {
   buffer[offset + index] = data[index];
 }
 return size;
```

20. [JS Wrapper] Modularization

- Global JavaScript object that can be used to control code execution and access exported methods.

21. [JS Wrapper] Modularization

```
emcc ./mediainfo.o \
  -o mediainfo-api.js \
  --bind <DEPENDENCIES>\
  -s EXTRA_EXPORTED_RUNTIME_METHODS='["FS", "callMain"]' \
  -s FORCE_FILESYSTEM=1 \
  -s EXIT_RUNTIME=1 \
  -s ENVIRONMENT=worker \
  -s INVOKE_RUN=0 \
  -s MODULARIZE=1 \
  -s ALLOW_MEMORY_GROWTH=1
```

22. [JS Wrapper] Modularization

```
const module = CreateMediaInfoInstance({
  noExitRuntime: false,
  print: (str: string) => {
    stdout.push(str);
  },
  printErr: (str: string) => {
    stderr.push(str);
  },
  onRuntimeInitialized: () => {
    loadedResolve();
  },
  onExit: (status: number) => {
    finishedResolve(stdout.join(""));
  },
  wasmBinary: wasmBinary,
});
```

23. Q&A

Thanks a lot for having me!