WebAssembly System Interface (WASI)

Stability: 1 - Experimental

The WASI API provides an implementation of the WebAssembly System Interface specification. WASI gives sandboxed WebAssembly applications access to the underlying operating system via a collection of POSIX-like functions.

```
import { readFile } from 'fs/promises';
import { WASI } from 'wasi';
import { argv, env } from 'process';

const wasi = new WASI({
  args: argv,
  env,
  preopens: {
    '/sandbox': '/some/real/path/that/wasm/can/access'
  }
});

// Some WASI binaries require:
//   const importObject = { wasi_unstable: wasi.wasiImport };
const importObject = { wasi_snapshot_preview1: wasi.wasiImport };

const wasm = await WebAssembly.compile(
  await readFile(new URL('./demo.wasm', import.meta.url))
);
const instance = await WebAssembly.instantiate(wasm, importObject);

wasi.start(instance);
```

```
'use strict';
const { readFile } = require('fs/promises');
const { WASI } = require('wasi');
const { argv, env } = require('process');
const { join } = require('path');

const wasi = new WASI({
  args: argv,
  env,
  preopens: {
    '/sandbox': '/some/real/path/that/wasm/can/access'
  }
});

// Some WASI binaries require:
//   const importObject = { wasi_unstable: wasi.wasiImport };
```

```
const importObject = { wasi_snapshot_preview1: wasi.wasiImport };

(async () => {
  const wasm = await WebAssembly.compile(
    await readFile(join(__dirname, 'demo.wasm'))
  );
  const instance = await WebAssembly.instantiate(wasm, importObject);

  wasi.start(instance);
})();
```

To run the above example, create a new WebAssembly text format file named `demo.wat`:

```
(module
    ;; Import the required fd_write WASI function which will write the given io vectors to s
    ;; The function signature for fd_write is:
    ;; (File Descriptor, *iovs, iovs_len, nwritten) -> Returns number of bytes written
    (import "wasi_snapshot_preview1" "fd_write" (func $fd_write (param i32 i32 i32 i32) (res

    (memory 1)
    (export "memory" (memory 0))

    ;; Write 'hello world\n' to memory at an offset of 8 bytes
    ;; Note the trailing newline which is required for the text to appear
    (data (i32.const 8) "hello world\n")

    (func $main (export "_start")
        ;; Creating a new io vector within linear memory
        (i32.store (i32.const 0) (i32.const 8))  ;; iov.iov_base - This is a pointer to the
        (i32.store (i32.const 4) (i32.const 12))  ;; iov.iov_len - The length of the 'hello

        (call $fd_write
            (i32.const 1) ;; file_descriptor - 1 for stdout
            (i32.const 0) ;; *iovs - The pointer to the iov array, which is stored at memory
            (i32.const 1) ;; iovs_len - We're printing 1 string stored in an iov - so one.
            (i32.const 20) ;; nwritten - A place in memory to store the number of bytes writ
        )
        drop ;; Discard the number of bytes written from the top of the stack
    )
)
```

Use wabt to compile `.wat` to `.wasm`

```
$ wat2wasm demo.wat
```

The `--experimental-wasi-unstable-preview1` CLI argument is needed for this example to run.

Class: `WASI`

The `WASI` class provides the WASI system call API and additional convenience methods for working with WASI-based applications. Each `WASI` instance represents a distinct sandbox environment. For security purposes, each `WASI` instance must have its command-line arguments, environment variables, and sandbox directory structure configured explicitly.

`new WASI([options])`

- `options` {Object}
  - `args` {Array} An array of strings that the WebAssembly application will see as command-line arguments. The first argument is the virtual path to the WASI command itself. **Default:** `[]`.
  - `env` {Object} An object similar to `process.env` that the WebAssembly application will see as its environment. **Default:** `{}`.
  - `preopens` {Object} This object represents the WebAssembly application's sandbox directory structure. The string keys of `preopens` are treated as directories within the sandbox. The corresponding values in `preopens` are the real paths to those directories on the host machine.
  - `returnOnExit` {boolean} By default, WASI applications terminate the Node.js process via the `__wasi_proc_exit()` function. Setting this option to `true` causes `wasi.start()` to return the exit code rather than terminate the process. **Default:** `false`.
  - `stdin` {integer} The file descriptor used as standard input in the WebAssembly application. **Default:** `0`.
  - `stdout` {integer} The file descriptor used as standard output in the WebAssembly application. **Default:** `1`.
  - `stderr` {integer} The file descriptor used as standard error in the WebAssembly application. **Default:** `2`.

`wasi.start(instance)`

- `instance` {WebAssembly.Instance}

Attempt to begin execution of `instance` as a WASI command by invoking its `_start()` export. If `instance` does not contain a `_start()` export, or if `instance` contains an `_initialize()` export, then an exception is thrown.

`start()` requires that `instance` exports a `WebAssembly.Memory` named `memory`. If `instance` does not have a `memory` export an exception is thrown.

If `start()` is called more than once, an exception is thrown.

`wasi.initialize(instance)`

- `instance` {WebAssembly.Instance}

Attempt to initialize `instance` as a WASI reactor by invoking its `_initialize()` export, if it is present. If `instance` contains a `_start()` export, then an exception is thrown.

`initialize()` requires that `instance` exports a `WebAssembly.Memory` named `memory`. If `instance` does not have a `memory` export an exception is thrown.

If `initialize()` is called more than once, an exception is thrown.

`wasi.wasiImport`

- {Object}

`wasiImport` is an object that implements the WASI system call API. This object should be passed as the `wasi_snapshot_preview1` import during the instantiation of a `WebAssembly.Instance`.