Facilitating deterministic distributed computation with WASI

FOSDEM 2020

JAKUB KONKA
Who am I?
“Who Are You”

My name is Jakub Konka

R&D Researcher at Golem Factory

Regular contributor to Wasmtime and WASI, and one of the authors of wasi-common library

Member of WebAssembly CG

@kubkon

kubkon@golem.network

kubkon@jakubkonka.com

@kubkon
What is WASI?
What is WASI?

01

WASI - WebAssembly System Interface

02

Standardisation led by Bytecode Alliance

03

Capability-based security - safe and portable access to host’s resources

Source: https://wasi.dev
<table>
<thead>
<tr>
<th>Allowed</th>
<th>✔</th>
<th>Forbidden</th>
<th>✗</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>File::create(&quot;/workspace/new&quot;)</code>?;</td>
<td></td>
<td><code>File::open(&quot;/dev/null&quot;)</code>?;</td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>rand::thread_rng();</code></td>
<td></td>
<td><code>let now = SystemTime::now();</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is the setting?
Meet the Golem Network

REQUESTOR
of computing resources
demand side of the market

DEVELOPER

PROVIDER
of computing resources
supply side of the market
Meet the Golem Network

REQUESTOR

CHOOSE AN APP

+ 

WASM APPS

DEVELOPER

DEVELOP AN APP

NETWORK
Wasm sandbox in Golem

C/C++/Rust binary

wasm32-unknown-emschipten

Wasm module

SpiderMonkey-based sandbox

In host's memory
Verification by redundancy
Is WASI deterministic?
Sources of nondeterminism in WASI

Access to random device
- Provided by `random_get`
- Will get its own module
- Will require a capability

```
unsafe fn random_get(
    buf: *mut u8,
    buf_len: Size,
) -> Result<(), Errno> {
    // call `getrandom` to access
    // host's entropy source, and
    // populate input `buf`
}
```
Sources of nondeterminism in WASI

Access to system clocks

- Provided by `clock_time_get`
- Will get its own module
- Will require a capability

```rust
def unsafe fn clock_time_get(id: Clockid, precision: Timestamp) -> Result<Timestamp, Errno> {
    // call `clock_gettime` to
    // get current host's time
    // etc.
}
```
Sources of nondeterminism in WASI

03

File atim/mtim/ctim stats
- Part of `Filestat` struct
- Inherently set by the host when file is created/modified
- Can be read by a module via `fd_filestat_get` or `path_filestat_get`

```rust
struct Filestat {
    dev: Device,
    ino: Inode,
    filetype: Filetype,
    nlink: Linkcount,
    size: Filesize,
    atim: Timestamp,
    mtim: Timestamp,
    ctim: Timestamp,
}
```

```rust
unsafe fn fd_filestat_get(
    fd: Fd
) -> Result<Filestat, Errno> {
    // call `fstat` to
    // get info on the underlying
    // host's fd
}
```
Sources of nondeterminism in WASI

Listing contents of a directory
- Provided by `fd_readdir`
- Order of entries dependent on the host and the filesystem used

```rust
unsafe fn fd_readdir(
    fd: Fd,
    buf: *mut u8,
    buf_len: Size,
    cookie: Dircookie,
) -> Result<Size, Errno> {
    // call `readdir` iteratively
    // to get enough dir entries
    // starting from `cookie` to
    // fully populate `buf`
}
```
Sources of nondeterminism in WASI

05

And the list goes on!
Encourage you to join the ongoing discussion here:

WebAssembly/WASI/issues/190
Can WASI be made deterministic though?
The model

Input WASI file descriptor

Exported "compute" Wasm function

```
fn compute(in: Fd, out: Fd)
```

Out: Fd

Output WASI file descriptor

The only rights we provide is reading or in WASI terms:
```
rights::fd_read
```

The only rights we provide is writing or in WASI terms:
```
rights::fd_write
```
What is WASI file descriptor?

<table>
<thead>
<tr>
<th>WASI Fd</th>
<th>0</th>
<th>...</th>
<th>11</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>Stdin</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
struct Entry {
    // ...
    os_handle: OsHandle,
    rights_base: Rights,
    rights_inheriting: Rights,
}
```
WASI Fd rights?

- Rights::fd_read
  - `fd_read(fd, iovs)`?
  - `fd_fdstat_get(fd)`?

- Rights::fd_write
  - `fd_write(fd, ciovs)`?
  - `fd_fdstat_get(fd)`?

But nothing else!
Have we just achieved determinism?
Almost! But not quite there yet...

You can still invoke these, since they are `Fd` independent

```c
poll_oneoff(...);
random_get(...);
environ_get(...);
clock_time_get(...);
```

Good news is, they will all get their own module and require a capability
Time for examples!
Everything’s on Github!

01 ————
Examples + description on Github:  
  kubkon/wasi-compute

02 ————
3 examples to play with:
  1. hello-compute – read from `in`, uppercase, write to `out`
  2. test-compute – verify that `in` and `out` have only `fd_read` and `fd_write` respectively
  3. flite-compute – plug in a text-to-speech `flite` engine into model

03 ————
Fork, play with, break, extend...
In general, have fun!
Any questions?

Have more questions about Wasm, WASI and Golem?
Contact me direct on

🐦 @kubkon
✉️ kubkon@golem.network
✉️ kubkon@jakubkonka.com
🔗 @kubkon