Building Ephemeral Virtual Filesystems

For WebAssembly

Daniel Phillips
@d_philla

About me - Dan Phillips

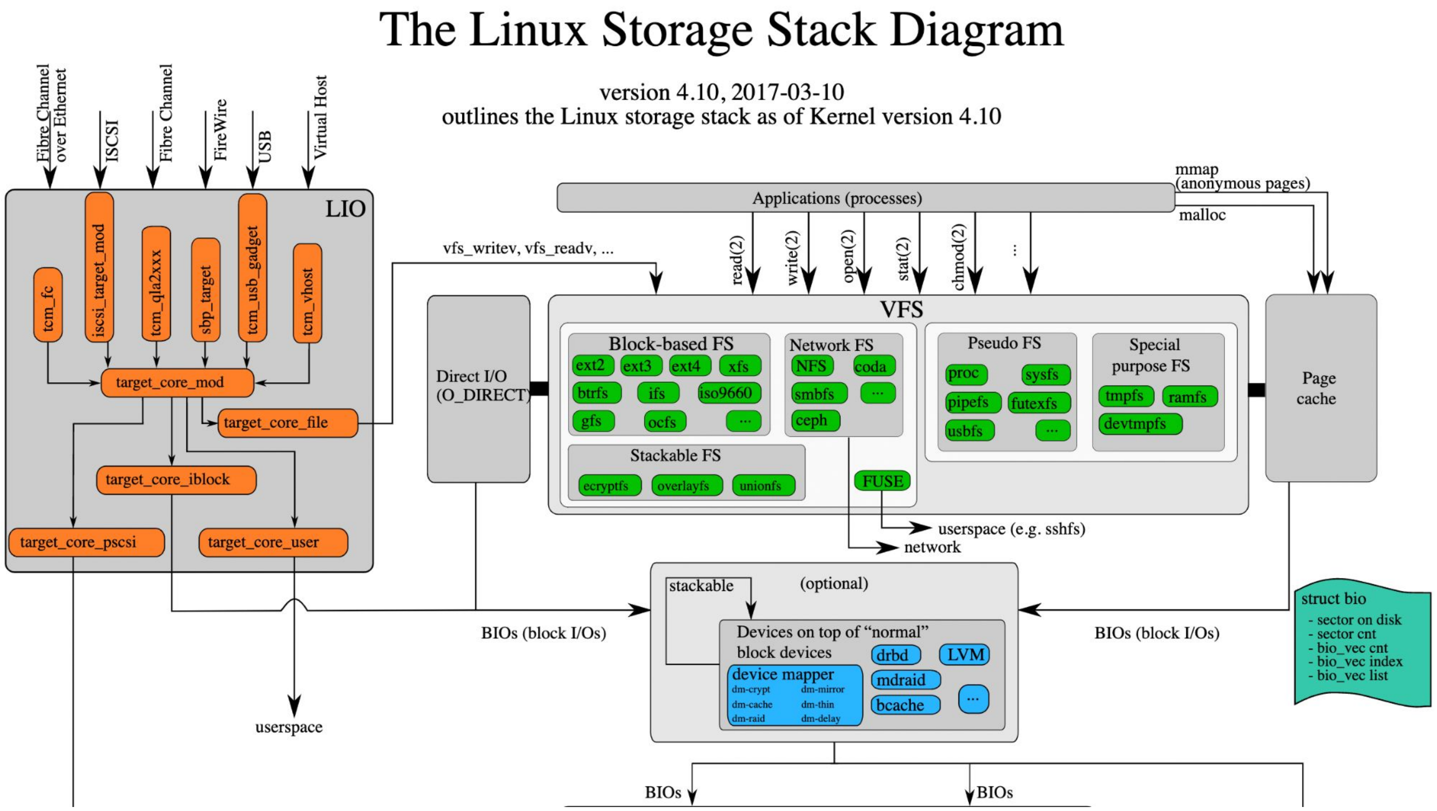
- Engineer / Wasm Lead @ Loophole Labs
- WasmChicago Group - wasmchicago.org
- @d_philla

The Problem: What to do about Filesystems?

- Applications need them; Wasm alone doesn't have them
- Using FSs with Bare Metal -> VMs -> Containers… -> Wasm?
- "Why not"- driven development

POSIX, and current, non-Wasm state of affairs

- Defines the interface
- Libc + Syscalls
- Virtual Filesystems in linux
  - an abstraction layer between userspace and the underlying file systems
- Underlying Filesystems, Networked, block devices, etc.



Source: Werner Fischer, Georg Schönberger - http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram

Current Approaches

- WASI
  - Provides scoped access to host filesystem resources
  - Interface: wasi-libc
- Emscripten VFS
  - Web-first, VFS on top of JS, can be backed by storage options like indexedDB or local storage
  - Interface: adapted musl libc
- wasi-vfs
  - In-memory, Wasm-embedded, read-only VFS
  - Interface: WASI, hooks WASI syscalls

Why not go a step further?

Thinking about Abstractions

- User space vs Kernel space
- libcs, Syscalls, context switching, etc
- So, what’s possible? Let’s question abstractions all the way down

Exploration Story: a Wasm libc + Wasm-first VFS

Why?

- Many, many, many programs rely on a libc interface for File i/o
- Freedom from underlying hosts

(non-rhetorically) Why not?

- It’s a lot to reinvent
- Constraints of the environment
- Alternatives already exist, specifically advances in WASI

🚨**DISCLAIMER** 🚨

I don't really know anything about Filesystems 😅

A libc is needed - but which?

- Really just need to satisfy the libc interfaces, everything else is fair game
- i.e. why port abstractions from a system with different constraints?
    - The abstractions imposed by Linux conventions may need not apply
- Keeping it simple, to start
    - open, read, write, close

An FS – what could it look like?

- As plain as possible to start, as universal (as possible)
  - Elements: implement basic fs syscalls, structure, RW, Permissions, etc.
- Another FUSE?
  - Try not to borrow abstractions from previous constraints, e.g. Kernel vs Userland
- Virtual, only be virtue of being in a Wasm Environment
  - Literally: syscalls on storage within limits of Wasm module, all under a libc
  - The fewer abstractions that leak, the better
- Wasm 🤝 ‘Serverless’ functions
  - maybe a slim, “ephemeral,” Wasm-first FS could be a better fit in some situations?

FS operations: Scale Function (scale.sh)

WIP: Persistence: A Snapshot model?

Possibility #1

- A Module loads state on instantiation from a centralized source of truth, even another VFS/FS on the host

Possibility #2

- A stateful VFS WASI module (component?), that exposes a resource handle to Wasm-only application modules instances, signaling on application instance shutdown to update state on underlying platform
    - like kernel-implemented VFSs? Standardize underlying storage operations

How “ephemeral” does a stateful component need to be?

- For example, a stateful layer could span several related or recurring application instances
- More exploration needed

The Future

So…

Why not?

- In-memory dbs already used: E.g. Redis as primary dbs, memcached, etc, many more of these could be “Wasm-first” as components
- VFSs already help standardize storage interactions in kernel implementations, why not in this paradigm?
- Linking to a persistent storage layer gets pushed to problem space of distributed computing, if this path is pursued

More challenges

- There are several flavors of libc,

  which wrap even further varieties of syscalls,

  which interact with the vast types of “real”/virtual/non-Wasm Filesystems,

  on even more types of platforms 😱

  How could this be managed?

Unified Tooling

- **strace**-like tool for an arbitrary Wasm module, outside of scope of this talk, but a WIP

- Generate bespoke and unique VFS + Libc-like implementations for novel, arbitrary Wasm modules, regardless of platform they were generated on, etc. – *only implementing what is needed for a give module*

- Allows for "Virtual Platform Layering" which, when contrasted with the POSIX/Syscall kernel/user-space abstractions, can use only what is needed for a specific instance of a module, and nothing more

Within the Component Model

- Virtualization Layers – (Luke Wagner's recent Wasm Day talk)
- Resource / handle types could mean that the only glue code needed is a libc interface, and
- resources are 'directly' usable, so minimal Syscall cost, perf
- Tooling for virtualization, supporting interface + Wasm-first Filesystem generation for components could make for a very promising future
- WebAssembly as an implementation detail

Thanks!

- Thanks to the ByteCode Alliance, WebAssembly Community Group for letting me be a fly on the wall in meetings
- Special thanks to Yuta Saito, Shivansh Vij for reviewing this talk
- Sergio for organizing

daniel@loopholelabs.io - Loophole Labs

Twitter - @d_philla

WasmChicago Group - wasmchicago.org