SibylFS

Formalize serious system (FS)
- POSIX

Using spec
- Testing
- Spec invariants

Influenced POSIX.

Challenge: non-determinism

Q: What's the non-det in spec?
Why? What does it enable?

Non-determinism

- Error codes → many possible errors OK. Choice by dev of FS.
- # bytes from read → perf choices.

readdir → not atomic, under-constrained spec.

concurrency

Non-det: check many impl's ⇐

Maybe too lax.

POSIX

Unix spec: common "man pages".

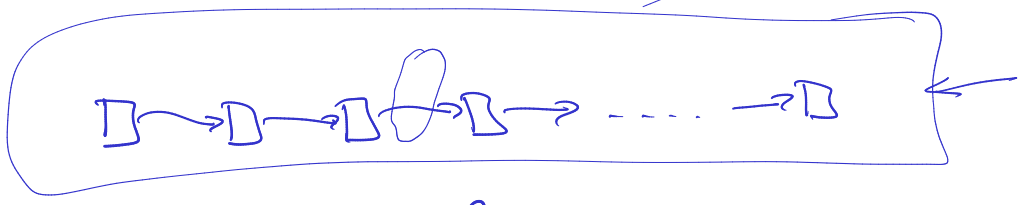
1980s: Many Unix OSes
- Similar, but minor diffs.
- Closed-source
- Portable apps.

Plan: experts from Unix vendor.
- Result: formalized/specified common core.
- Room for OS diffs

Influential

FS testing

Workload → large files
→ many files
→ concurrent ops
→ crashes.



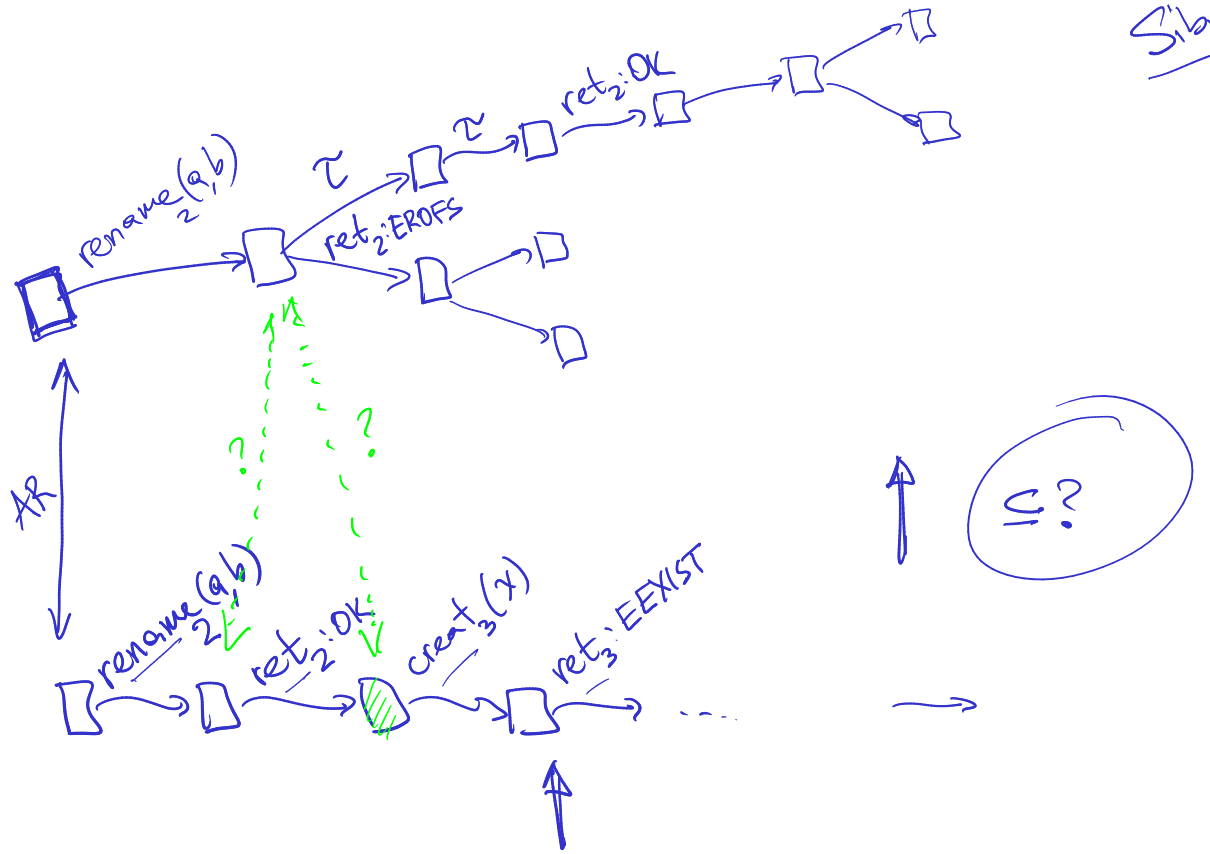
What's correct?
- FS behavior complex, diff by impl
- POSIX informal.

★ – No kernel panics.

★ – App behavior.

Popular app. should not error out

Overview



SibylFS model:

Lem DSL for specs.

↳ Abstract state ⟦⟧

↳ Transition $\xrightarrow{\text{label}}$

Workload ⟹ FS impl (Linux, MacOS, ...)

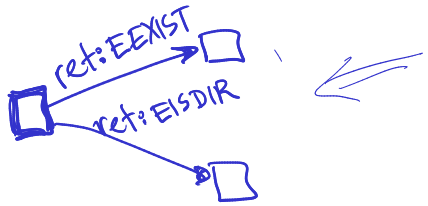
↳ Drive FS into corner cases.

No expected result. ⟸

Test gen: explores all possible syscalls.

Non-determinism

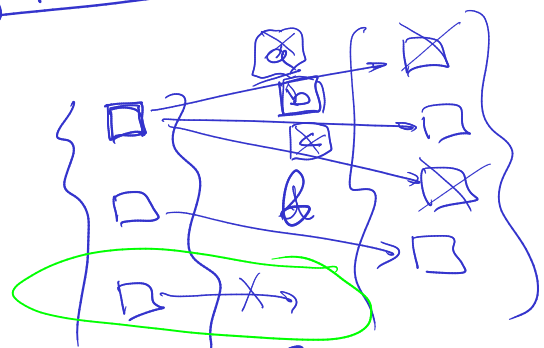
Plan: next-state enumeration.



Lem DSL: a ||| b

a: return EEXIST |||
b: return EISDIR

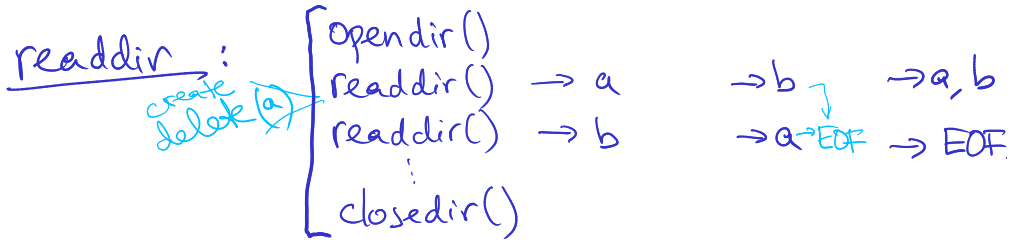
Many possible states



State explosion?

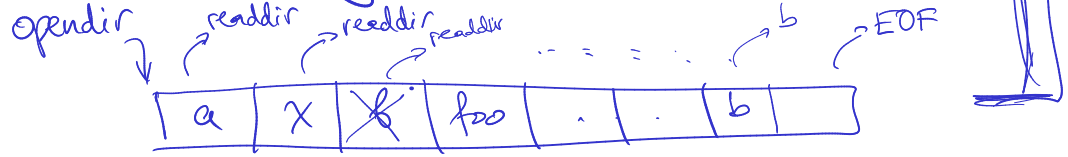
Dead-end states.
Match labels w/ impl trace. (prune)
Zero matching labels ⇒ bug/inconsistency.

readdir:



- POSIX allows any order
- Concurrent creates/deletes.

Spec?
→ Unmodified file in dir → must appear.
Concur deleted → may appear. → UB
Creates → may appear. → UB
Same file only appears once (?)



Model: dir → {"must"}, {"may"}.

Value?
- check many impl.
- captures core of readdir that devs can rely on

app.

SibylFS model

os_label: call (pid, cmd) → syscall name, args.
ret (pid, val)
create (pid)
destroy (pid)
τ

os_state: { Proc: { FID table, PID table }, FS: { Directories, Files. }

os_trans: os_state → label → { os_state OR special }

special → POSIX UB or unspec. ..

→ function calls → path lookup
→ dir ops
→ proc ops
:

(!!!)