Sibyl FS

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 input'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/specification
common core.
Room for OS diffs

Influential

FS testing
large files
Workload
many files
concurrent ops
crashes

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

A: No kernel panics.
A: App behavior
Popular app should not error
Overview

SlayFS model:
- Lem DSL for specs.
- Abstract state
- Transition
  - label

Spec:

Code:

Workload \rightarrow \text{FS impl (Linux, MacOS, ...)}

\rightarrow \text{Drive FS into corner cases.}
\rightarrow \text{No expected result.}
\rightarrow \text{Test gen: explores all possible syscalls.}
Non-determinism
Plan: next-state enumeration.

Lem DSL: \[ a \parallel b \]
\[ a: \text{return EXIST} \parallel \]
\[ b: \text{return EISDIR} \parallel \]
Many possible states

State explosion?
Dead-end states
Match labels w/ impl trace (prune)
Zero matching labels \( \Rightarrow \) bug/inconsistency.
Siby|PS model

OS_label: call (pid, cmd)
         ret (pid, val)
Create(pid)
destroy(pid)

OS_state: FID table
          Proc [PID table]
          FS [Directories
               Files]

OS_trans: OS_state \rightarrow \text{label} \rightarrow \{\text{os_state OR?}

\begin{itemize}
  \item function calls \rightarrow \text{path lookup}
  \item dir ops
  \item proc ops
\end{itemize}

\text{POSIX UB or unspec.}