Formally Specifying POSIX File Systems

Gian Ntzik, Pedro da Rocha Pinto and Philippa Gardner

Imperial College London

July 16, 2015

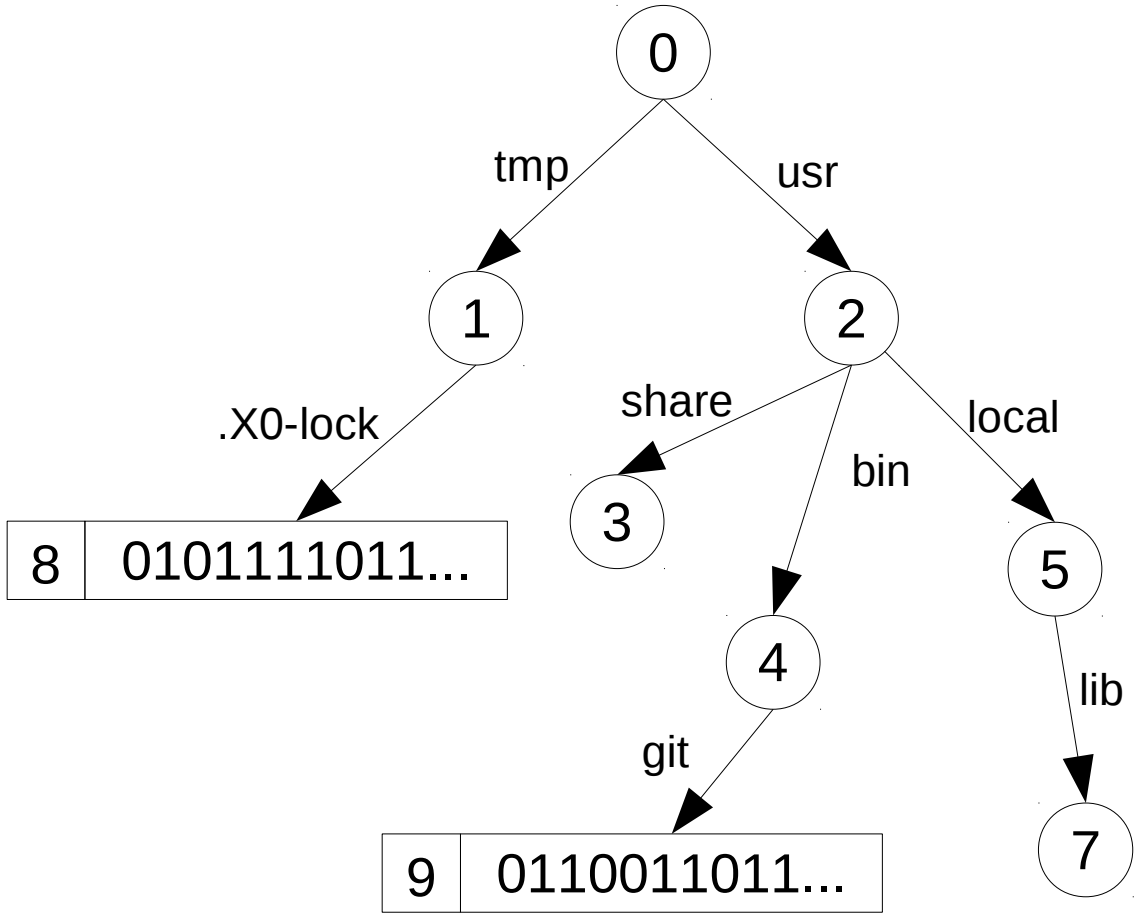
POSIX File Systems

- POSIX: Portable Operating System Interface
- Large part devoted to the file system
- English Specification
  - Underspecified
  - Ambiguous
  - Absence of proper memory model

Objectives

- Find formalism suitable for specifying POSIX file system operations
  - Suitable for clients and implementations
  - Client reasoning in a program logic
- Focus a core fragment:
  - Structural operations: `mkdir`, `rmdir`, `link`, `unlink`, `rename`, ...
  - IO: `open`, `read`, `write`, `lseek`, `close`, ...

File System Example

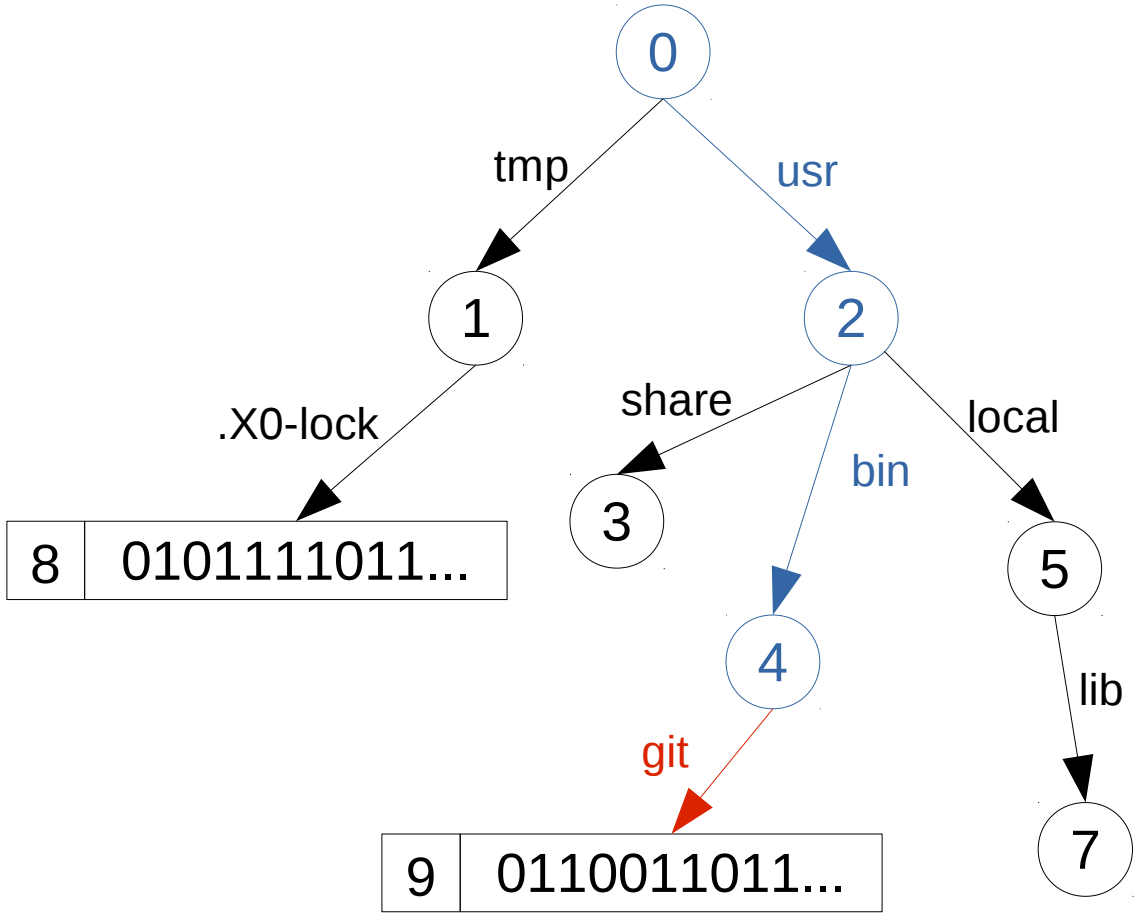


First Challenge: Atomicity

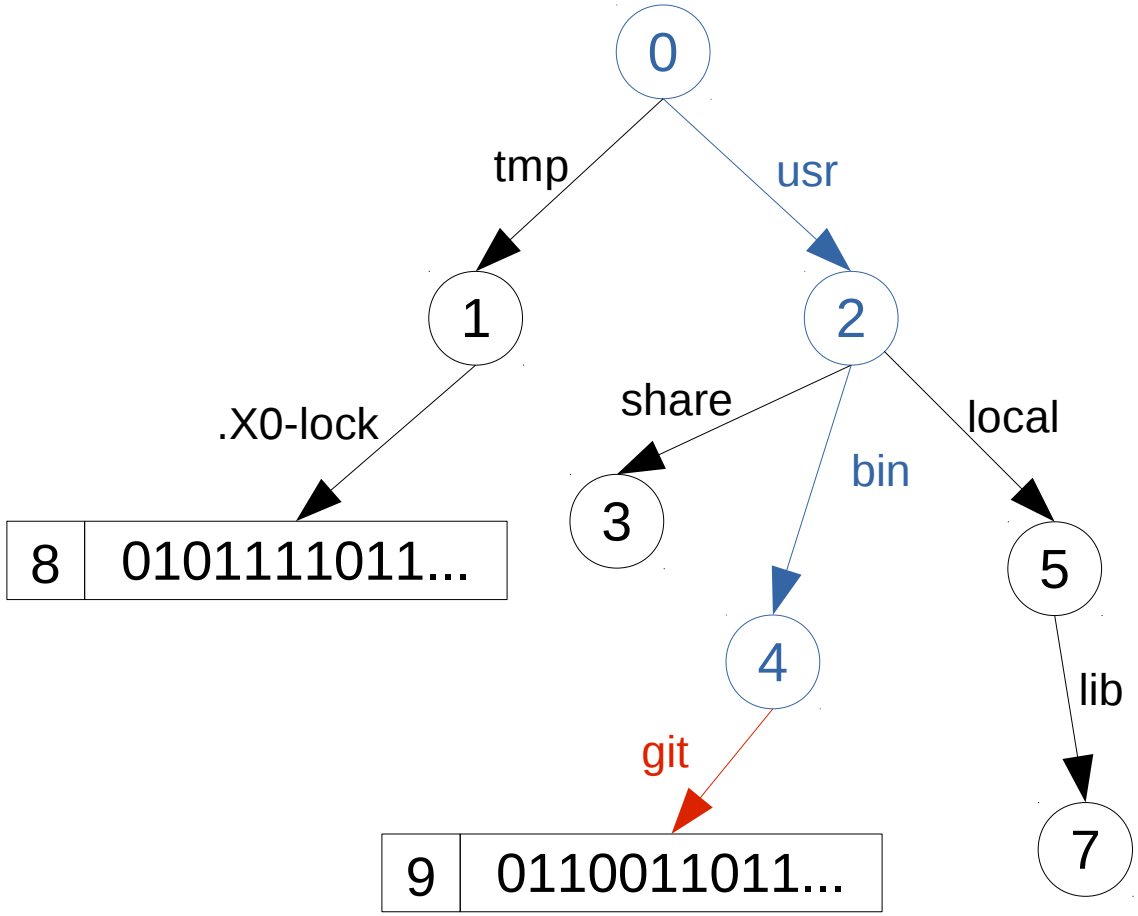
Example: `unlink`

- `unlink(p)` : Atomically remove the file identified by path `p`.
- "Atomic" has an unusual meaning in POSIX.

unlink(/usr/bin/git)

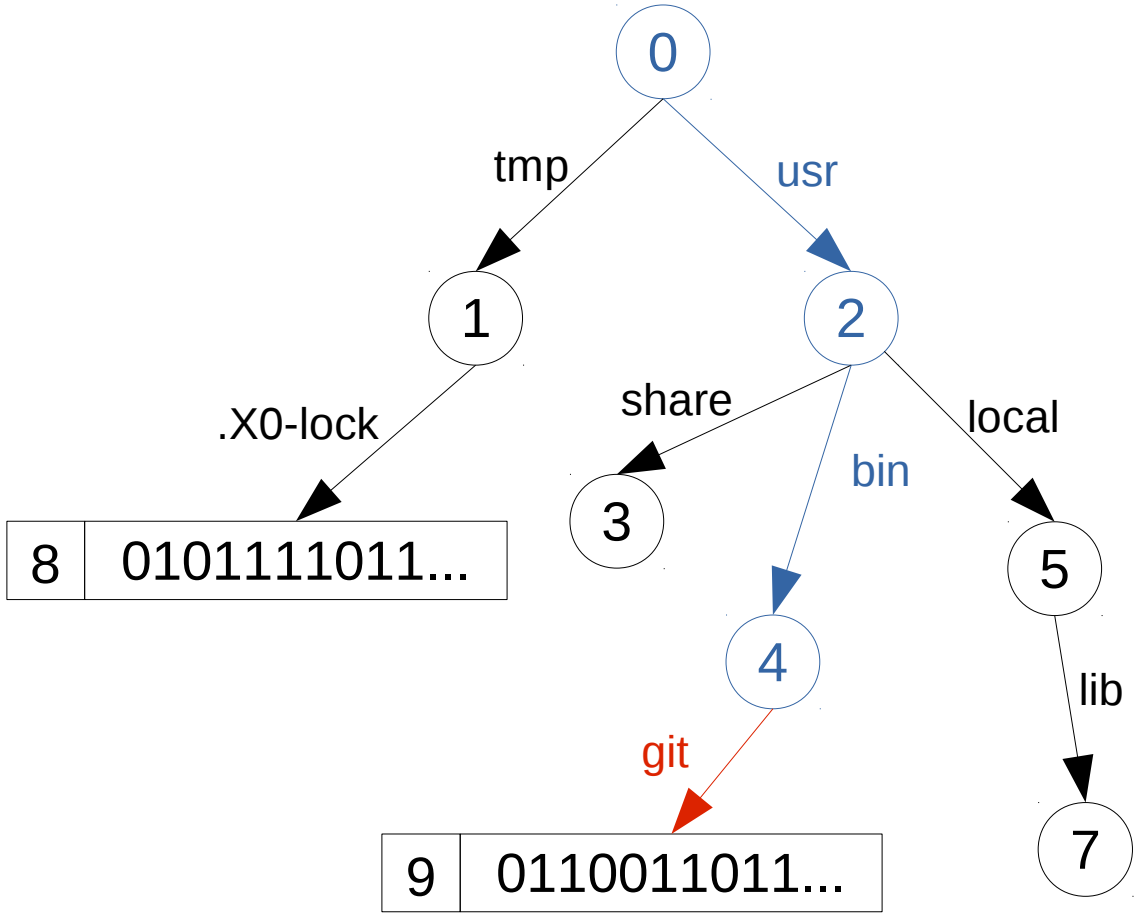


unlink(/usr/bin/git)



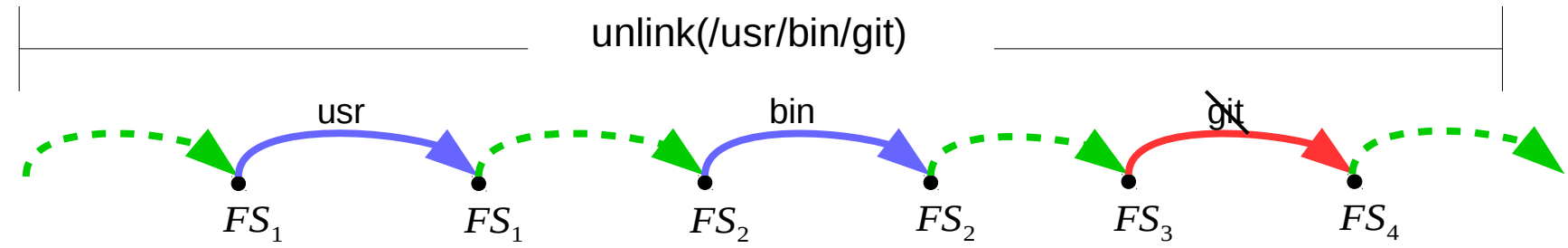
- Only removing `git` is required to be atomic.

unlink(/usr/bin/git)



- Only removing `git` is required to be atomic.
- Path resolution: a sequence of atomic reads.

Sequence of atomic actions



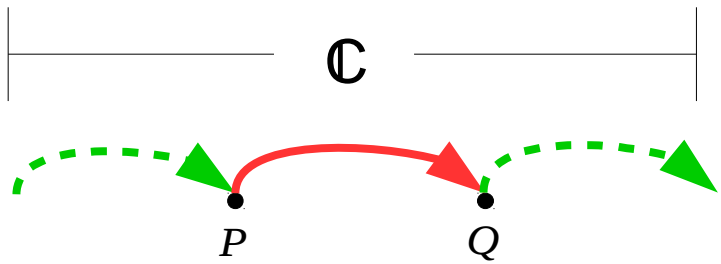
Environment: mupltiple atomic updates

Thread: single atomic read in path traversal

Thread: single atomic update

Single atomic step specification: Atomic Hoare Triple

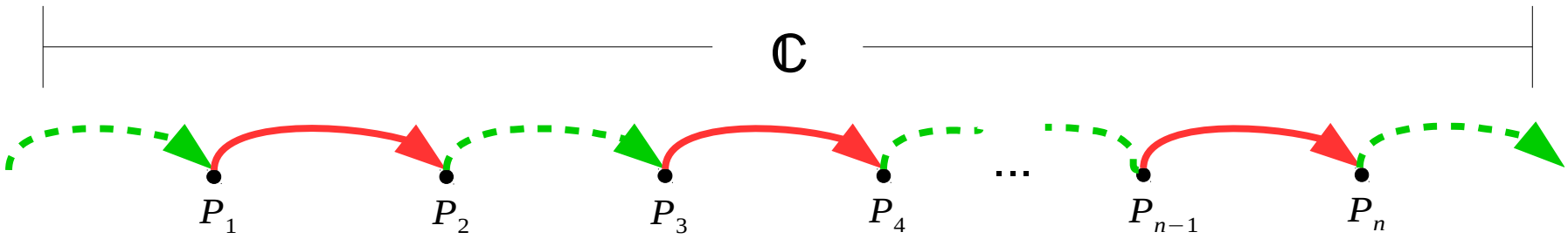
From TaDA, IRIS, ... we know how to specify a single atomic step.



$\langle P \rangle \, \mathbb{C} \, \langle Q \rangle$

Multi-atomic specifications

We extend to multiple atomic steps



$$\mathbb{C} \sqsubseteq \langle (P_1, P_2); (P_3, P_4); \ldots; (P_{n-1}, P_n) \rangle$$

Multi-atomic Program Logic

- Introduce multi-atomics:
  $\mathbb{C} \sqsubseteq \langle (P_1, P_2); (P_3, P_4); \ldots; (P_{n-1}, P_n) \rangle$
- Justified by an encoding in IRIS
- Extend reasoning rules for single atomic steps to multiple steps

Single Step Equivalence

SingleAtomic:

$$\mathbb{C} \sqsubseteq \langle (P, Q) \rangle \iff \langle P \rangle \, \mathbb{C} \, \langle Q \rangle$$

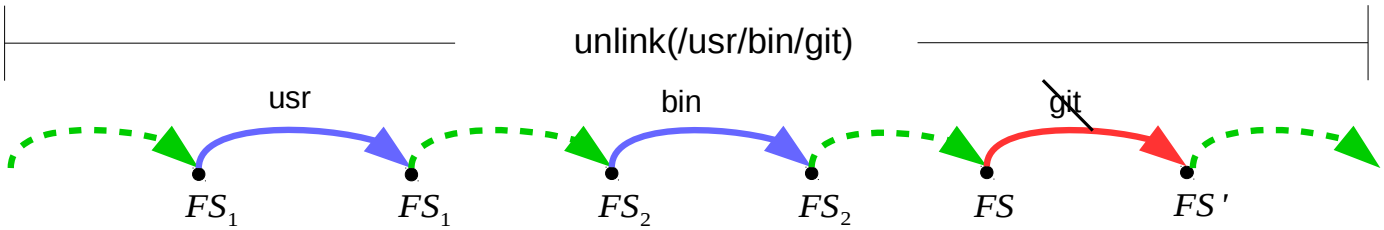
Sequence Rule

$$\frac{\mathbb{C}_1 \sqsubseteq \langle \ldots ; (P_1, Q_1) \rangle \quad \mathbb{C}_2 \sqsubseteq \langle (P_2, Q_2); \ldots \rangle}{\mathbb{C}_1 ; \mathbb{C}_2 \sqsubseteq \langle \ldots ; (P_1, Q_1); (P_2, Q_2); \ldots \rangle} \textsc{Multi-Seq}$$

Stuttering Rule

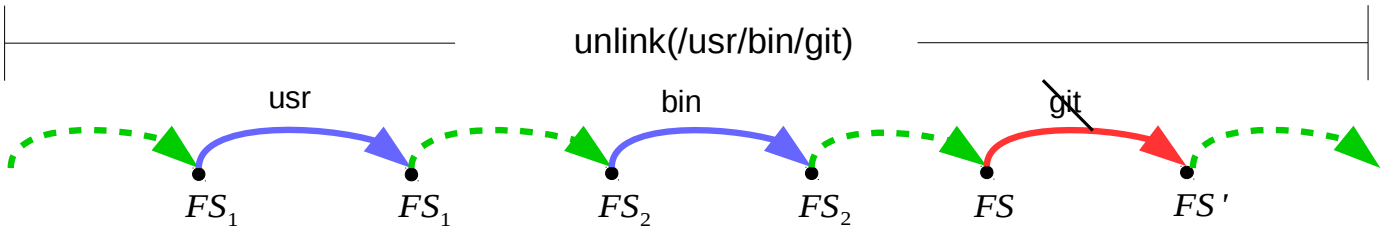
$$\frac{\mathbb{C} \sqsubseteq \langle \ldots; (P, P); (P, Q); \ldots \rangle}{\mathbb{C} \sqsubseteq \langle \ldots; (P, Q); \ldots \rangle} \textsc{Stutter}$$

`unlink`: Formal Specification



$$\texttt{unlink}(\texttt{/usr/bin/git}) \sqsubseteq \langle \mathsf{resolve}(\texttt{/usr/bin}, \iota_0, r); \ldots \rangle$$

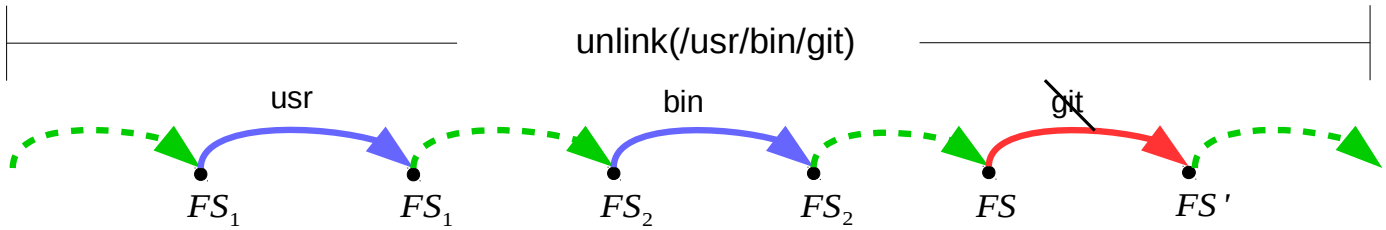
`unlink`: Formal Specification



$$\texttt{unlink}(\texttt{/usr/bin/git}) \sqsubseteq$$

$$\langle \mathsf{resolve}(\texttt{/usr/bin}, \iota_0, r); (\mathsf{fs}(FS), \mathsf{in}(FS, r, \texttt{git}) \Rightarrow \mathsf{rem}(FS, r, \texttt{git}) * \mathsf{ret} = 0) \rangle$$

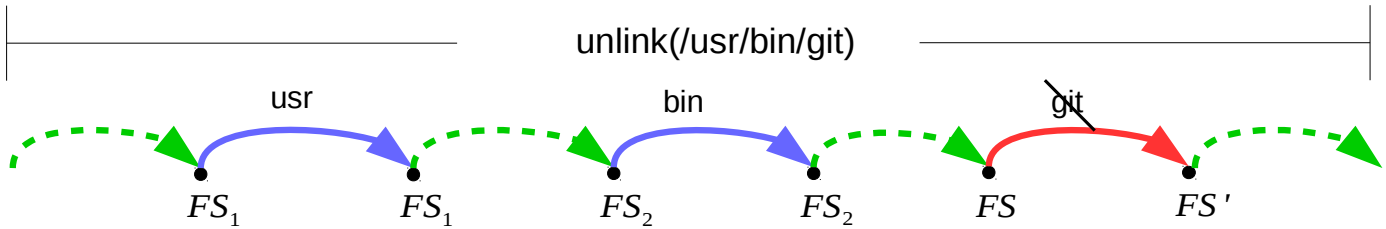
`unlink`: Formal Specification



$$\texttt{unlink}(\texttt{/usr/bin/git}) \sqsubseteq$$

$$\left\langle \text{resolve}(\texttt{/usr/bin}, \iota_0, r); \left(\text{fs}(FS), \begin{pmatrix} \text{in}(FS, r, \texttt{git}) \Rightarrow \text{rem}(FS, r, \texttt{git}) * \text{ret} = 0 \\ \wedge\ \text{out}(FS, r, \texttt{git}) \Rightarrow \text{fs}(FS) * \text{ret} = -1 * \texttt{errno} = \texttt{ENOENT} \end{pmatrix} \right) \right\rangle$$

`unlink`: Formal Specification



$$\texttt{unlink}(/\texttt{usr}/\texttt{bin}/\texttt{git}) \sqsubseteq$$

$$\left\langle \mathsf{resolve}(/\texttt{usr}/\texttt{bin}, \iota_0, r);\ \left(\mathsf{fs}(FS)\,,\ \begin{array}{l} r \in \textsc{In} \Rightarrow \begin{pmatrix} \mathsf{in}(FS, r, \texttt{git}) \Rightarrow \mathsf{rem}(FS, r, \texttt{git}) * \mathsf{ret} = 0 \\ \wedge\ \mathsf{out}(FS, r, \texttt{git}) \Rightarrow \mathsf{fs}(FS) * \mathsf{ret} = -1 * \texttt{errno} = \texttt{ENOENT} \end{pmatrix} \\ \wedge\ r \in \textsc{Err} \Rightarrow \mathsf{fs}(FS) * \mathsf{ret} = -1 * \texttt{errno} = r \end{array} \right) \right\rangle$$

Second Challenge: Unordered actions

Specifying unordered actions

- Example: $\texttt{rename}(\texttt{p}/\texttt{a}, \texttt{p}'/\texttt{b})$
- POSIX does not specify in which order $\texttt{p}$ and $\texttt{p}'$ are resolved

Specifying unordered actions

- Example: $\texttt{rename}(\mathtt{p}/\mathtt{a}, \mathtt{p}'/\mathtt{b})$
- POSIX does not specify in which order $\mathtt{p}$ and $\mathtt{p}'$ are resolved
- We can't do:

$$\texttt{rename}(\mathtt{p}/\mathtt{a}, \mathtt{p}'/\mathtt{b}) \sqsubseteq \langle \mathsf{resolve}(\mathtt{p}, \iota_0, r_1) \,;\, \mathsf{resolve}(\mathtt{p}', \iota_0, r_2) \,;\, \ldots \rangle$$

Specifying unordered actions

- Example: $\texttt{rename}(\mathrm{p}/\mathrm{a}, \mathrm{p}'/\mathrm{b})$
- POSIX does not specify in which order p and p' are resolved
- Solution:

$$\texttt{rename}(\mathrm{p}/\mathrm{a}, \mathrm{p}'/\mathrm{b}) \sqsubseteq \langle \mathsf{resolve}(\mathrm{p}, \iota_0, r_1) \parallel \mathsf{resolve}(\mathrm{p}', \iota_0, r_2) \,;\, \ldots \rangle$$

Parallel Rule

$$\frac{\begin{array}{c}\mathbb{C}_1 \sqsubseteq \langle (P_1, P_2); \ldots; (P_{n-1}, P_n) \rangle \\ \mathbb{C}_1 \sqsubseteq \langle (Q_1, Q_2); \ldots; (Q_{n-1}, Q_n) \rangle\end{array}}{\mathbb{C}_1 \parallel \mathbb{C}_2 \sqsubseteq \langle (P_1, P_2); \ldots; (P_{n-1}, P_n) \parallel (Q_1, Q_2); \ldots; (Q_{n-1}, Q_n) \rangle} \textsc{Multi-Par}$$

Client Applications

- Lock Files
- POSIX pipes
- POSIX Advisory Locks/Record Locking (on-going)
- Persistent Concurrent Queues (future)

Lock Files

- `lock(path)`: atomically create a non-existing lock file at `path`
- `unlock(path)`: remove the lock file identified by `path`
- Implemented similarly to spin locks
  - `open(path, O_CREAT|O_EXCL)` to try to lock
  - `unlink` to unlock

Lock File Implementation

```
function lock(path) {
  do {
    fd := open(path, O_EXCL|O_CREAT);
  } while (fd = −1);
  close(fd);
}

function unlock(path) {
  unlink(path);
}
```

Heap Based Lock Specification

- We know how to specify locks on the heap

Heap Based Lock Specification

- We know how to specify locks on the heap

$$\{\mathsf{emp}\}\ \texttt{makelock}()\ \{\mathsf{Lock}(\texttt{ret}, 0)\}$$

$$\langle \mathsf{Lock}(\mathtt{x}, v)\rangle\ \texttt{lock}(\mathtt{x})\ \langle \mathsf{Lock}(\mathtt{x}, 1) * v = 0\rangle$$

$$\langle \mathsf{Lock}(\mathtt{x}, 1)\rangle\ \texttt{unlock}(\mathtt{x})\ \langle \mathsf{Lock}(\mathtt{x}, 0)\rangle$$

Heap Based Lock Specification

- We know how to specify locks on the heap

$$\{\mathsf{emp}\}\ \texttt{makelock}()\ \{\mathsf{Lock}(\texttt{ret}, 0)\}$$

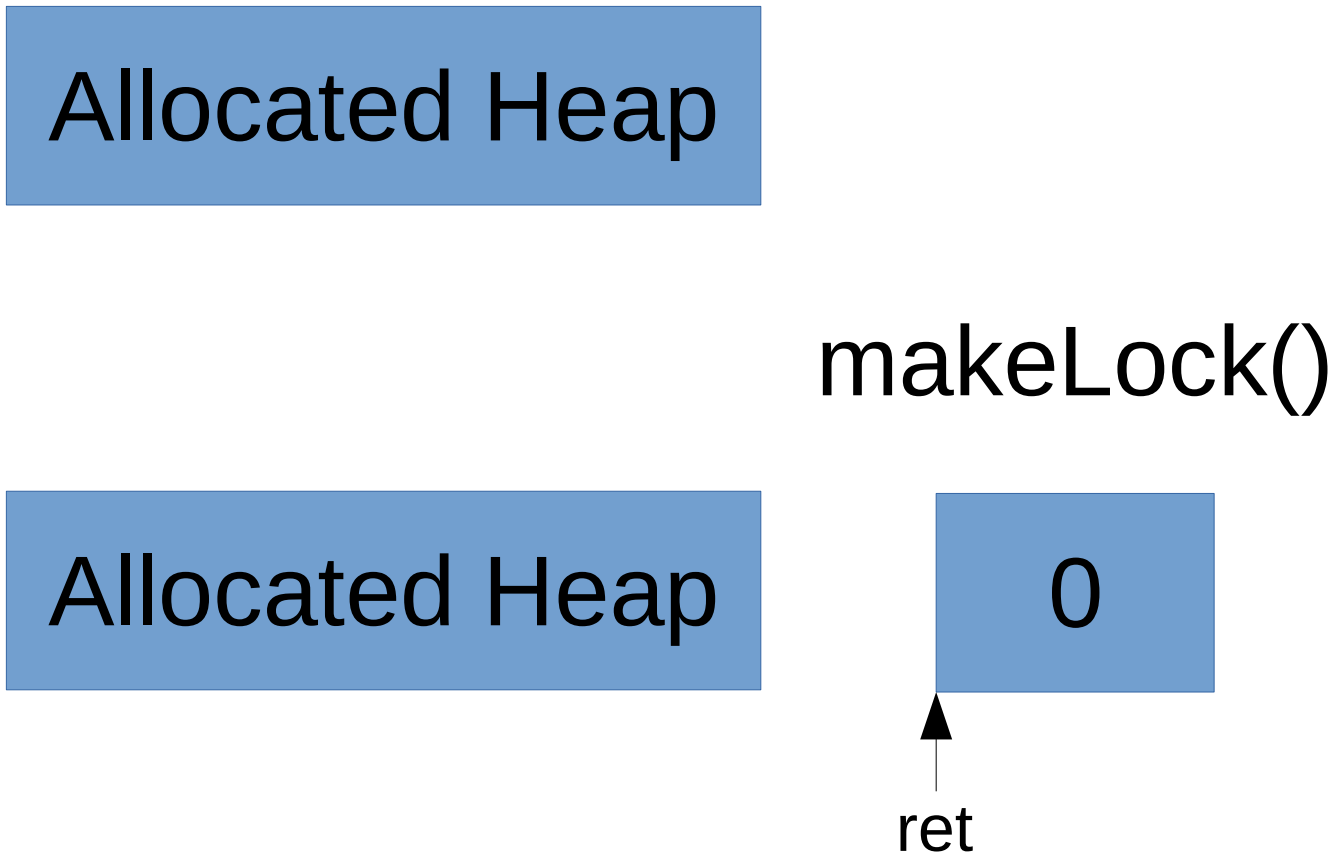
$$\langle \mathsf{Lock}(\texttt{x}, v)\rangle\ \texttt{lock}(\texttt{x})\ \langle \mathsf{Lock}(\texttt{x}, 1) * v = 0\rangle$$

$$\langle \mathsf{Lock}(\texttt{x}, 1)\rangle\ \texttt{unlock}(\texttt{x})\ \langle \mathsf{Lock}(\texttt{x}, 0)\rangle$$

- Ideally, we want the same specification for lock files
- Using `path` instead of heap address `x`

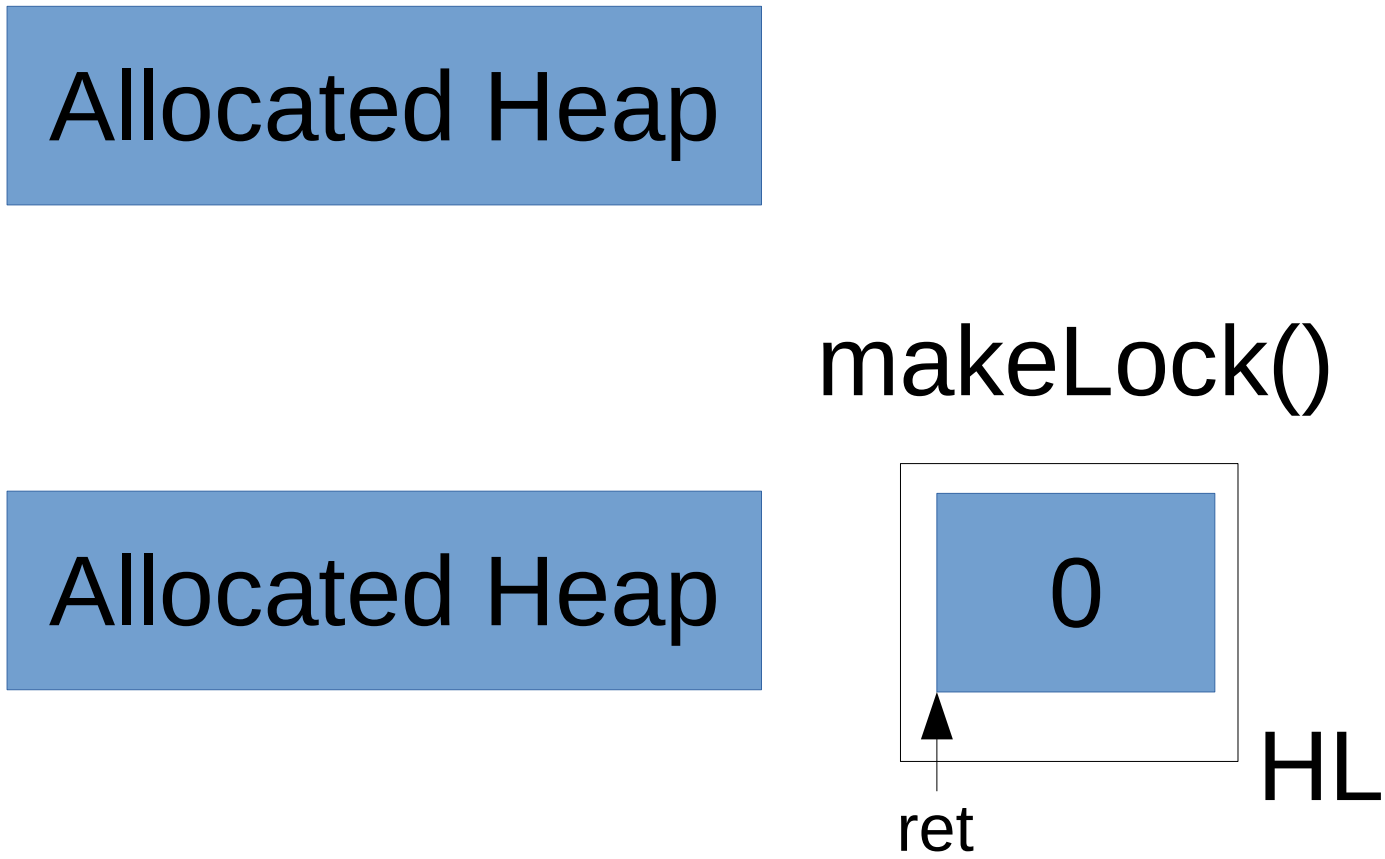
Third Challenge: Ownership

The Heap case



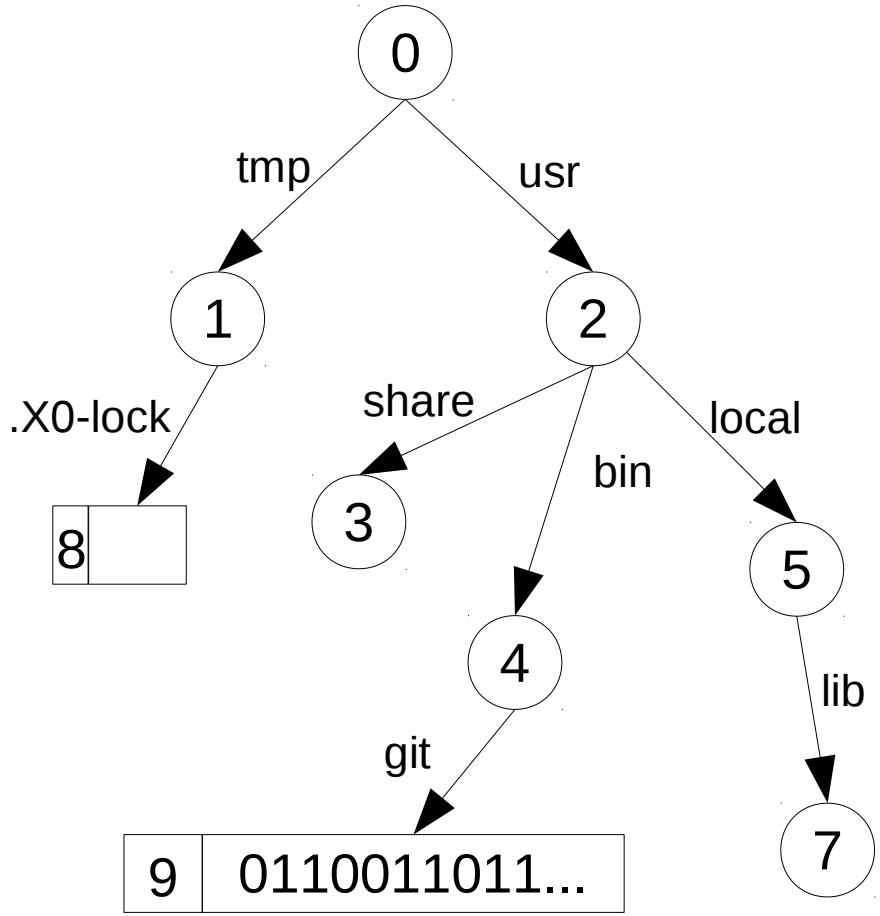
- The module owns the newly allocated memory

The Heap case

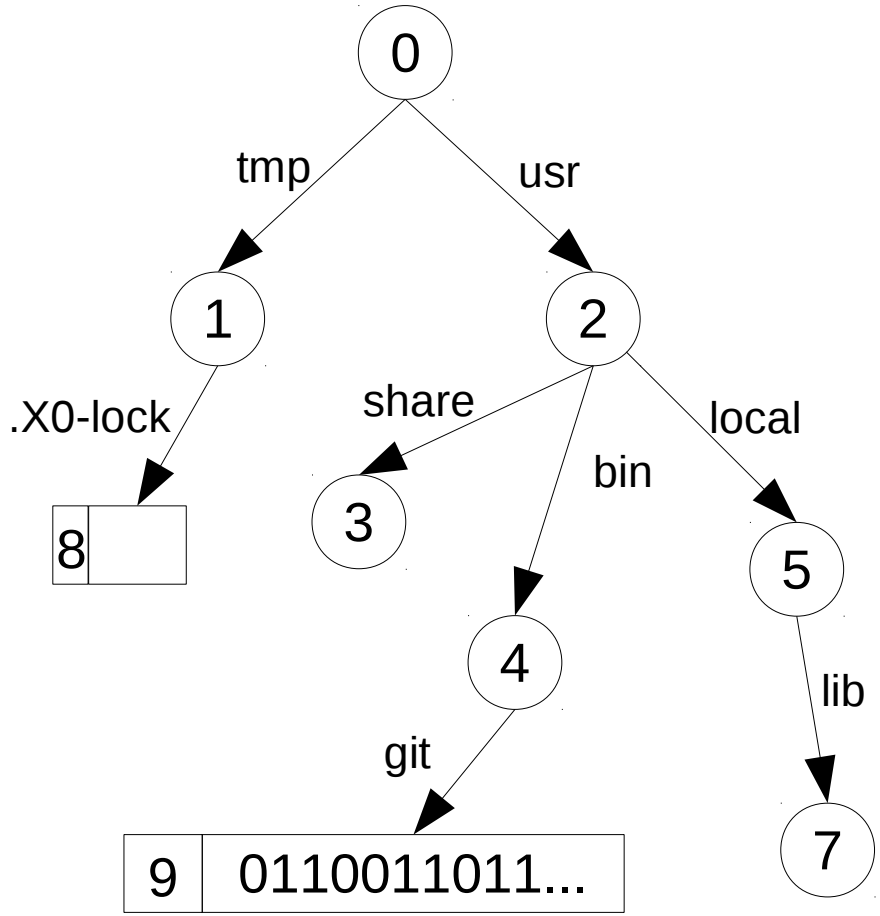


- The module enforces the sharing protocol

The File System case

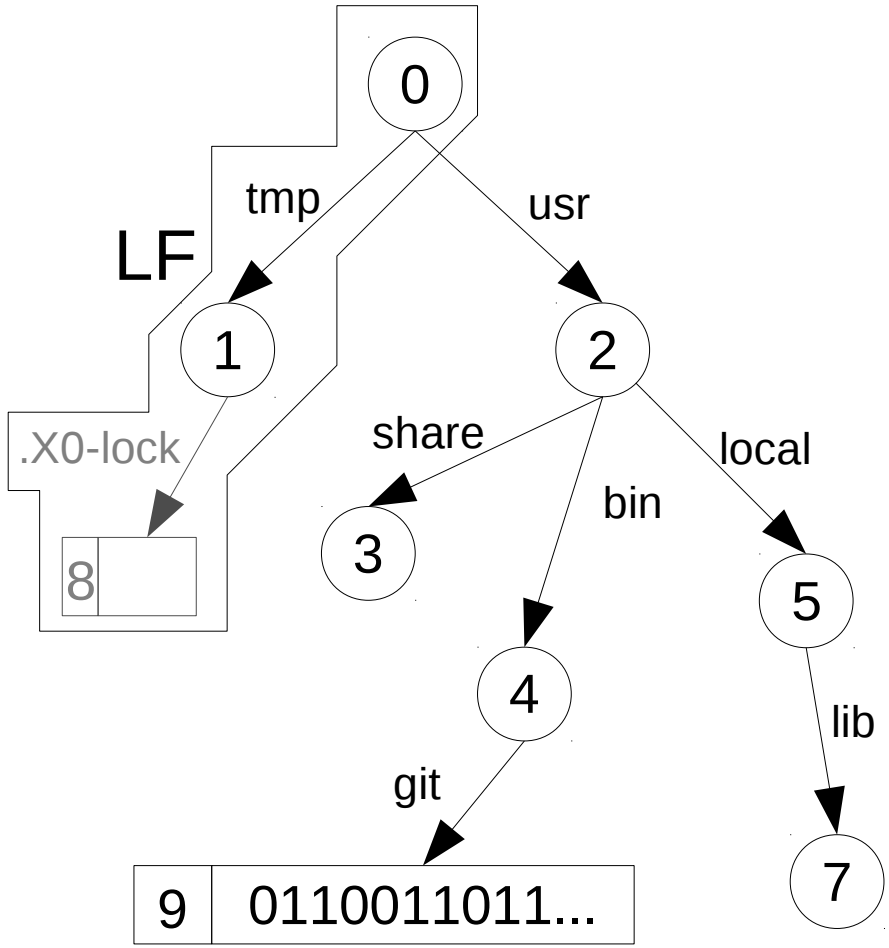


The File System case

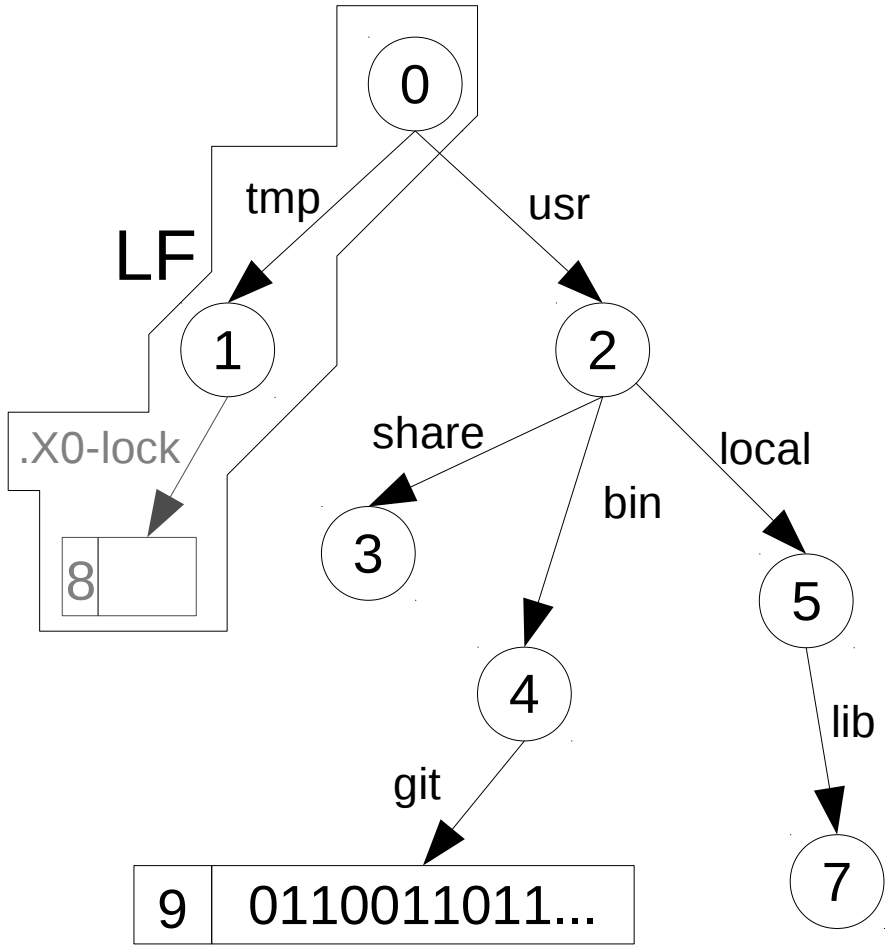


- No operation to extend with fresh path, unknown to environment
- Global path address space

The File System case

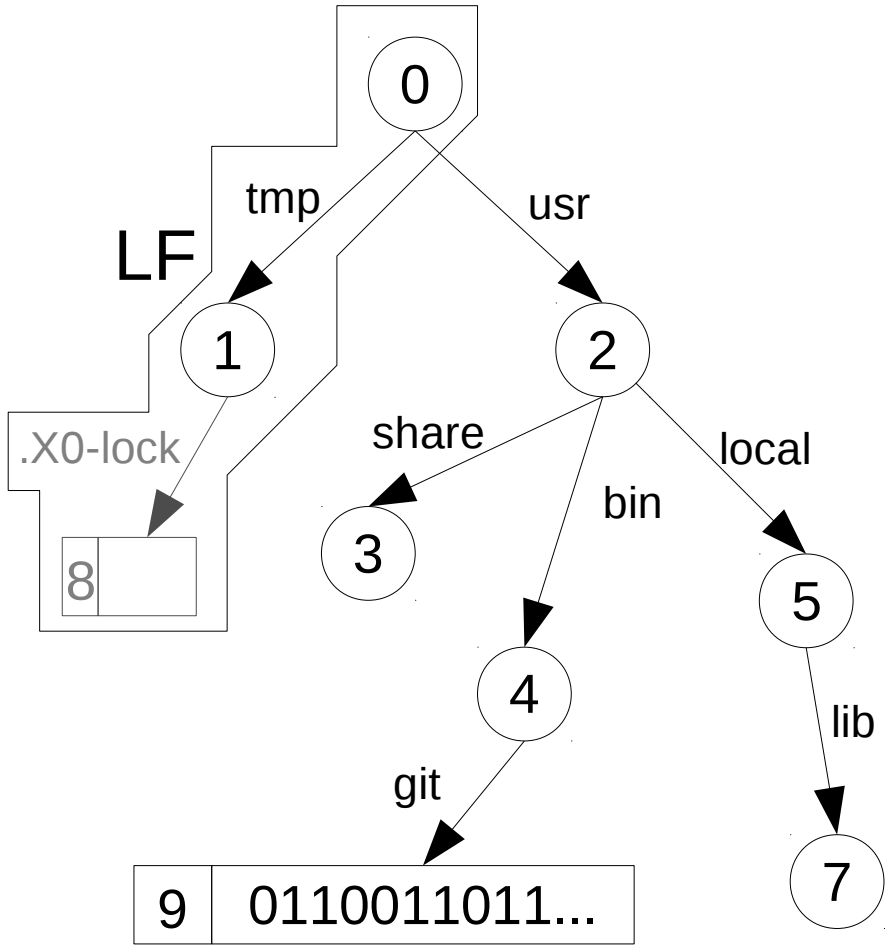


The File System case



- Clients must agree on sharing protocol

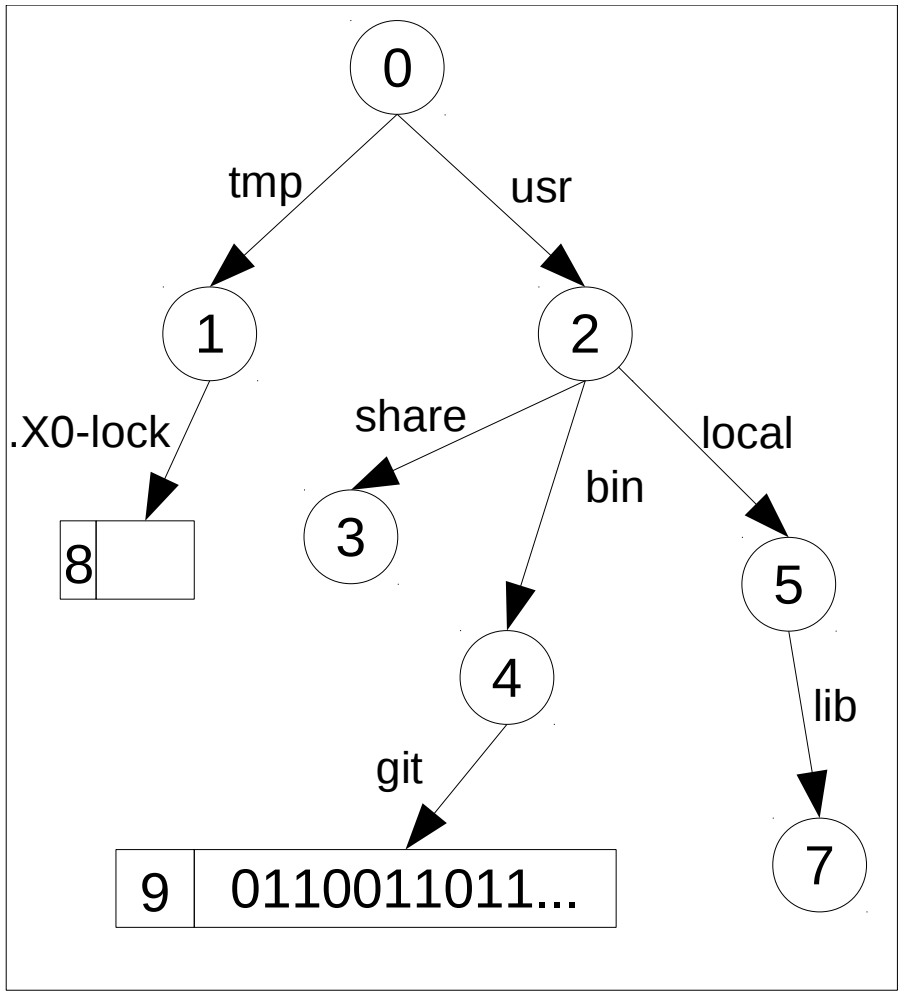
The File System case



- Clients must agree on sharing protocol
- *Cooporative ownership*

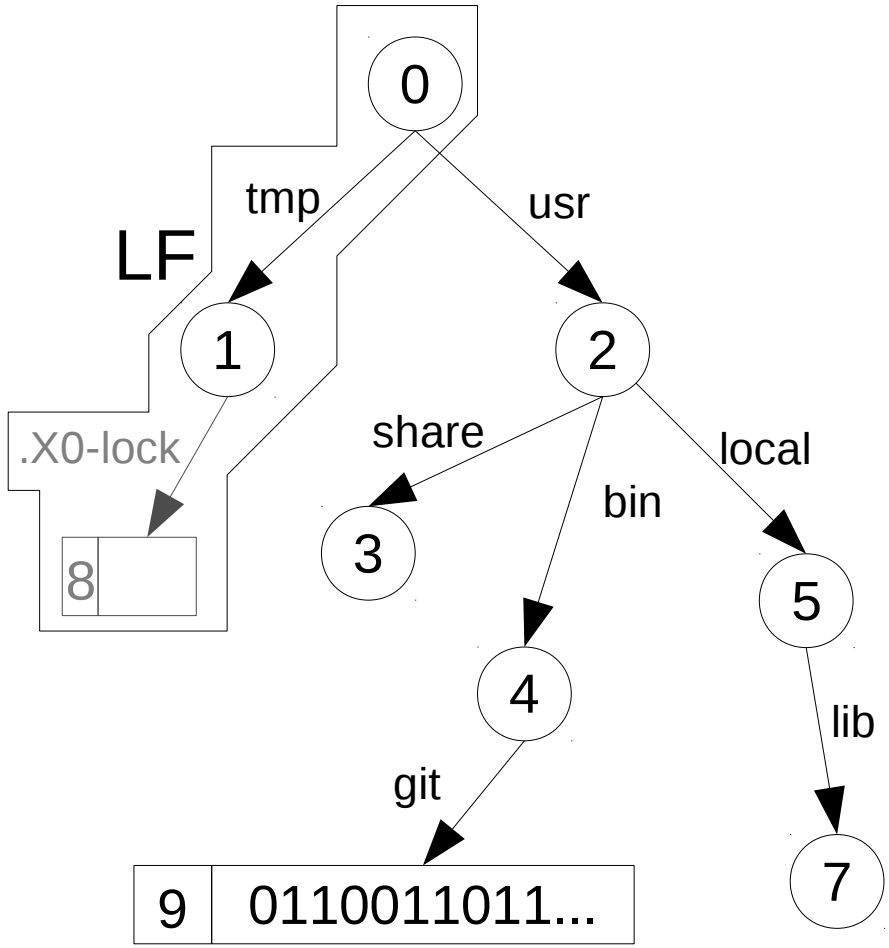
Lock file specification

If any client's protocols



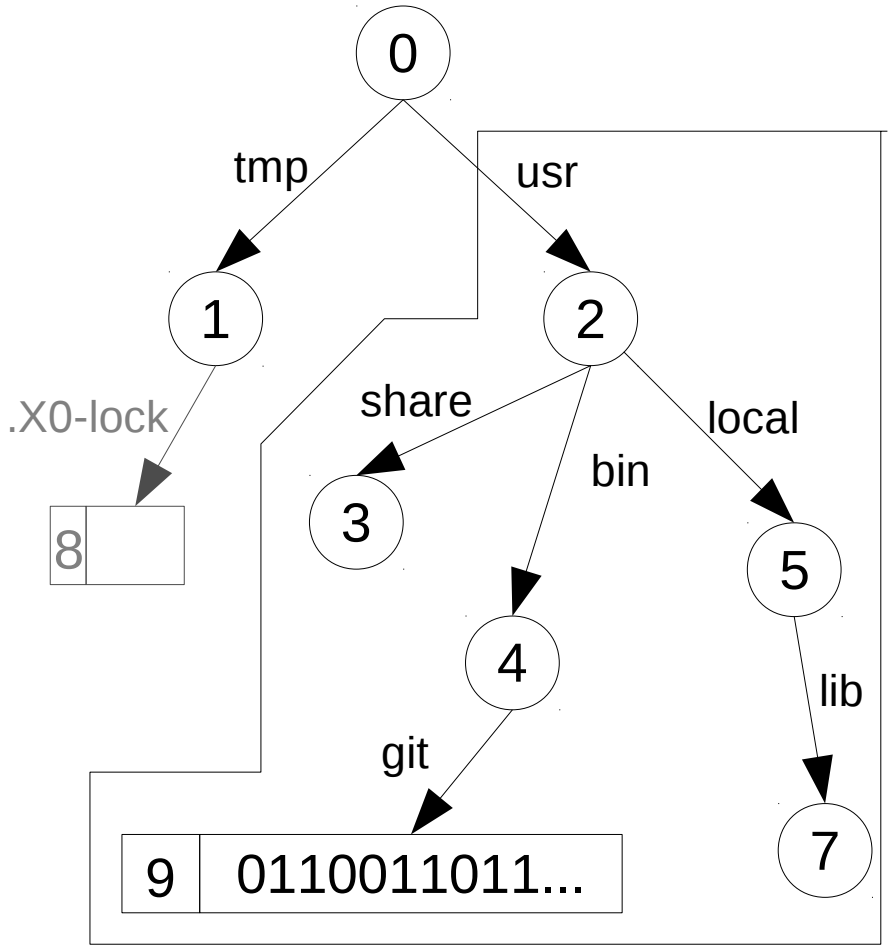
Lock file specification

contains the lock file protocol



Lock file specification

with other protocols



Lock File specification

then, we can use the lock file specification:

$$\texttt{lock}(\texttt{path}) \sqsubseteq \left\langle \mathsf{Lock}(\texttt{path}, v)\,, \mathsf{Lock}(\texttt{path}, 1) * v = 0 \right\rangle$$

$$\texttt{unlock}(\texttt{path}) \sqsubseteq \left\langle \mathsf{Lock}(\texttt{path}, 1)\,, \mathsf{Lock}(\texttt{path}, 0) \right\rangle$$

Lock File specification

$$\forall \texttt{path}, \boxed{P} \,.\, \mathsf{islock}\left(\texttt{path}, \boxed{P}\right) \Rightarrow$$

$$\texttt{lock}(\texttt{path}) \sqsubseteq \left\langle \mathsf{Lock}(\texttt{path}, v) \,, \mathsf{Lock}(\texttt{path}, 1) * v = 0 \right\rangle$$

$$\texttt{unlock}(\texttt{a}) \sqsubseteq \left\langle \mathsf{Lock}(\texttt{path}, 1) \,, \mathsf{Lock}(\texttt{path}, 0) \right\rangle$$

$$\mathsf{islock}\left(\texttt{path}, \boxed{P}\right) \triangleq \exists R. \boxed{P \iff \mathsf{LF}(\texttt{path}) * R}$$

Conclusions

- Introduced multi-atomic specifications
- Ordered sequences of atomic actions
- Unordered parallel atomic actions
- Formalised a fragment of POSIX file system operations
- Client reasoning
- Ownership in file systems is *cooperative*

Future Work

- Link with operational models & testing
- Verify implementations
- Explore connection with refinement & Hoare's algebraic laws
- Mechanisation