# POSIX Signals in Userspace

on the Redox μkernel

Jacob Lorentzon

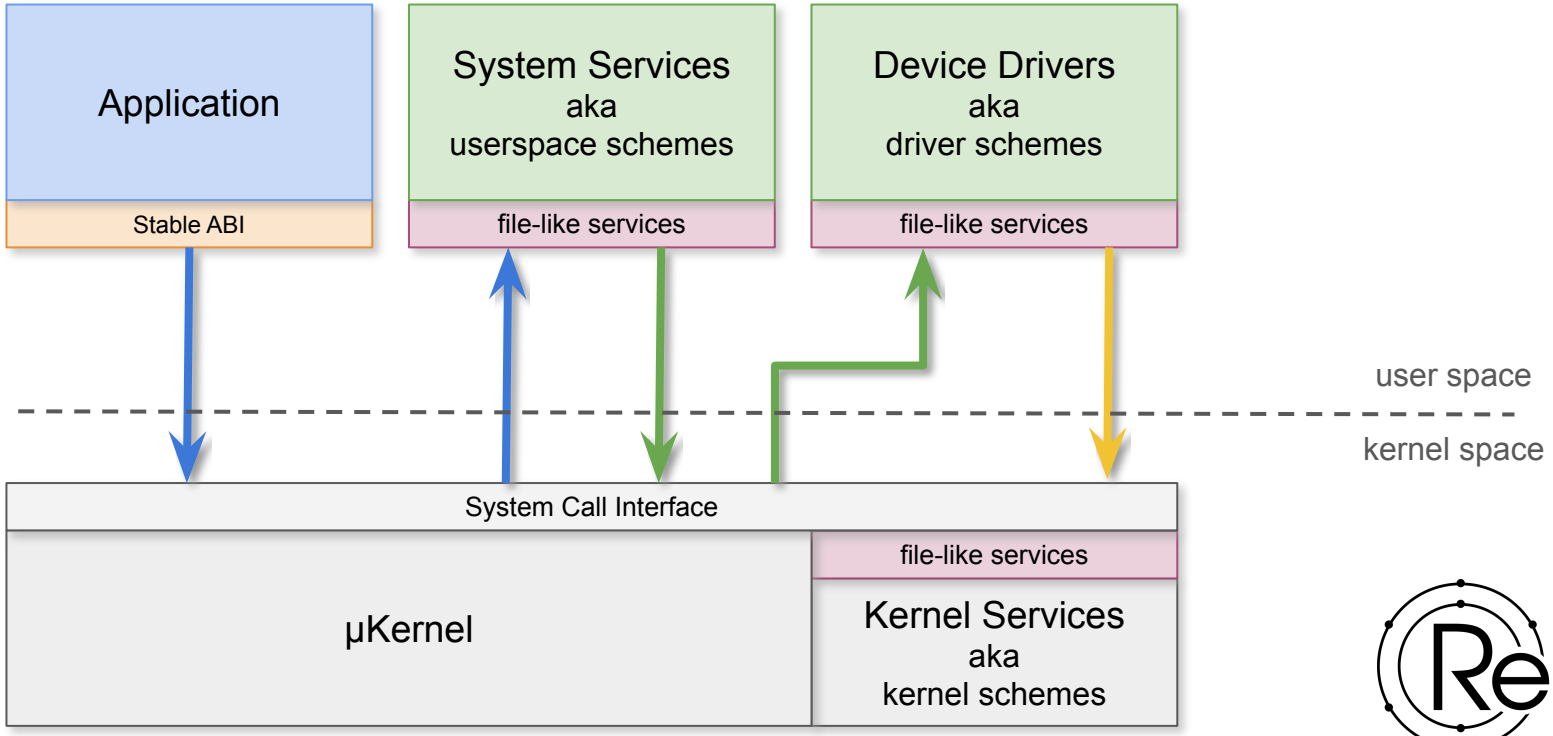
# Redox OS

- Unix-like userspace, on continuously shrinking µkernel
- Plan 9-inspired userspace filesystems
- Community-developed since 2015
- Written in Rust
  - Including our libc, relibc!
  - Some 3rd-party exceptions
- POSIX source-level compatibility
- Recent focus and progress on porting software
  - COSMIC apps
  - nushell
  - RustPython
  - GCC

# Me

- Redox contributor since 2019
- Redox Summer of Code 2020..=2023
  - I/O
  - Userspaceification of fork/execv
  - Demand paging implementation
- NLnet project (2024-2025)
  - Userspace signal handling
  - Userspace process management

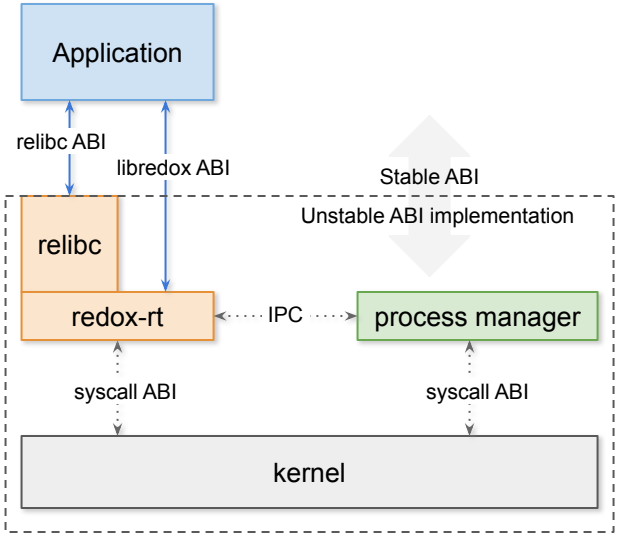
# Architecture




Redox OS

# Architecture

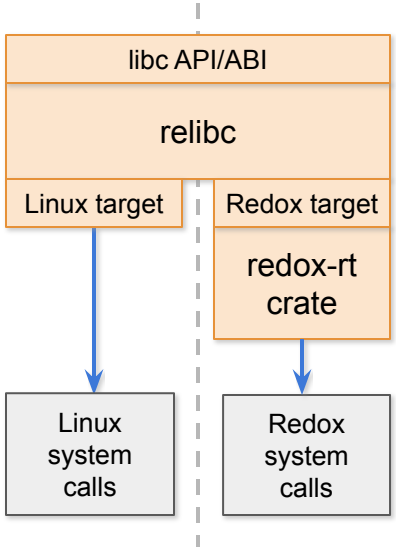
- Heavily file descriptor based, albeit some global state remains
- File syscalls are handled by “schemes”
- Syscall ABI intentionally kept unstable, shifting the stability layer to `redox-rt`
- POSIX and many other crucial parts moving incrementally to userspace libs
- Drivers run in separate userspace programs
- Kernel <30 kSLoC (~100k with dependencies)




Redox OS

# Relibc

- C library written in Rust
  - Even headers! (apart from macros, etc.)
  - Rustifying over time, reducing `unsafe {}`
- Redox and Linux support
- Focus on most of POSIX
- Source-level compatibility
- Two backends
  - `relibc` -> raw syscalls (Linux)
  - `relibc` -> `redox-rt` (Redox)




Redox OS

# Redox-rt

- Underlying relibc backend when targeting Redox
- Intended to become freestanding ABI
  - Backend for Rustix
  - WASI
  - Library “emulation”
- One of few components ‘allowed’ to syscall directly
- Handles most proc state and syscalls
- Fork/exec
- Signal handling!
- (Userspace proc manager WIP)

# POSIX signals

- Userspace analogue of interrupts
- Asynchronous
- ~64 of them
- Can mask/ignore
- Syscall interruption

# Problem

- Userspace needs state, and locks!
- Functions like open(3) are async safe, need sigprocmask :(
- In kernel mode, interrupts are almost always disabled
- Disabling/enabling ~20 cycles
- Userspace needs to enable/disable signals more often
- Syscall, ~200 cycles!

- What about… shared memory?

# Signals Protocol

- Goal: low cost `sigprocmask` ideally bypassing the kernel
- Goal: keep most state in, or accessible by, userspace
- Goal: provide a basic IPC cancellation primitive
- Solution: store signal state in kernel-shared page!
- Basic primitive very simple for the kernel
- Makes userspace slightly more complex

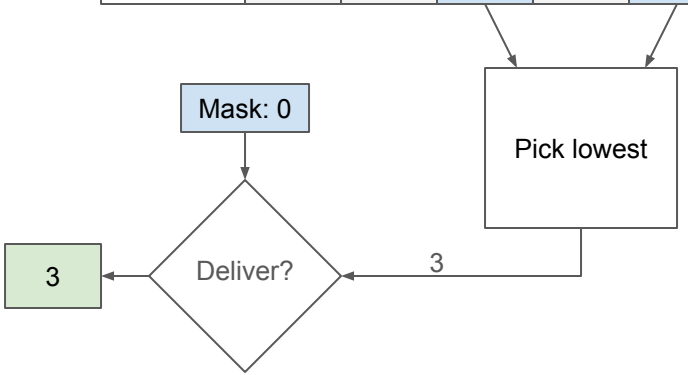
# Protocol

- Atomic ops!
- Per-thread: combined ‘allowset’ and ‘pending set’
- Logical AND gives the deliverable signals
- Thread-local flag for temporarily inhibiting signals, used internally
- Per-process pending set
- Kernel-accessible `siginfo_t`-like flags
- Ad-hoc exceptions for certain signals like SIGTSTP
- In-kernel queue for realtime signals, but lock-freedom theoretically possible (CMPXHG16B)

# Protocol

- Per-thread allowset
- Per-process and per-thread pending set (bitwise ORed)

| Signal | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Pending | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| Allow | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| Deliver | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |




Redox OS

# Signal sending

- Single-producer-multi-consumer
- Sender is kernel
- `pthread_kill`
  - Set pending
  - Check allowset, conditionally interrupt
  - Will clear bit once trampoline is entered
- `kill`
  - Set process-level pending
  - For all threads, check allowset
  - If unblocked, wake up thread
  - Spurious signals can occur, but only if actively enabled/disabled
  - Similarly, wait for first thread to clear bit

# Cancellation

# IPC

1. Userspace calls e.g. SYS_PREAD2
2. Kernel maps buffer virtually, queues request, yields thread
3. … Switch to server …
4. Scheme daemon handles request, queues response, calls kernel
5. … Switch to client …
6. Kernel returns from pread2

Synchronous, even when IO is non-blocking

# Cancellation

- Kernel checks process's and thread's masks before sleeping
- But how are synchronous calls interrupted?
- Interrupt results in cancellation request, scheme hopefully returns early
- Only SIGKILL can force-cancel an IPC syscall
- TODO: asynchronous syscalls or detaching POSIX calls from underlying IPC primitive

# Conclusion

- Increased (signal) POSIX coverage, for porting
  - Although standard is vague in certain areas (e.g. realtime sigs)
  - And “monolithic”!
- Ideally need further impl. testing
- Userspace process manager
- Dynamically linked relibc/redox-rt is close
- Improved mechanisms for moving state to userspace
- Further “userspaceification”!

# Thanks for listening!

Questions?

# Links

- https://redox-os.org/
- https://nlnet.nl/project/RedoxOS-Signals/
- https://fosdem.org/2025/schedule/event/fosdem-2025-5670-posix-signals-in-user-space-on-the-redox-microkernel/
- https://fosdem.org/2025/schedule/event/fosdem-2025-5973-redox-os-a-microkernel-based-unix-like-os/