3. System Calls

Outline

1. POSIX Standard
2. Essentials
3. Implementation

POSIX Standard

Portable Operating System Interface

- IEEE POSIX 1003.1 and ISO/IEC 9945 (latest standard: 2004)
- Many subcommittees

Portability Issues

- POSIX is portable and does not evolve much,
- ... but it is still too high level for many OS interactions
  E.g., it does not specify file systems, network interfaces or power management
- UNIX applications deal with portability with
  - C-preprocessor conditional compilation
  - Conditional and multi-target `Makefile` rules
  - GNU `configure` scripts to generate `Makefile`s
  - Shell environment variables (`LD_LIBRARY_PATH`, `LD_PRELOAD`)

System Calls Essentials

Return Values and Errors

- All system calls return an `int` (very rarely a `long`)

  $\geqslant 0$ if execution proceeded normally

  -1 if an error occurred

- When an error occurs, `errno` is set to the error code
  - Global scope, thread-local, `int` variable
  - It carries *semantical information* not available by any other mean
  - It is *not* reset to 0 before a system call
- `#include <errno.h>`

System Calls Essentials

Error Messages

- Print error message: `perror()` (see also `strerror()`)

Sample Error Codes

`EPERM`:	Operation not permitted
`ENOENT`:	No such file or directory
`ESRCH`:	No such process
`EINTR`:	Interrupted system call
`EIO`:	I/O error
`ECHILD`:	No child process
`EACCESS`:	Access permission denied
`EAGAIN`/`EWOULDBLOCK`:	Resource temporarily unavailable

System Calls Essentials

Standard Types

- `#include <sys/types.h>`
- Integral or pointer types in general, but portable

Examples

`clock_t`: clock ticks since last boot

`dev_t`: major and minor

`uid_t/gid_t`: user and group identifier

`pid_t`: process identifier

`ino_t`: inode

`mode_t`: access permissions

`off_t`: file offset)

`sigset_t`: set of signal masks

`size_t/ssize_t`: unsigned/signed size, signed allows to multiplex size and error condition in a return value

`time_t`: seconds since 01/01/1970

System Calls Essentials

Interrupted System Calls

- Deliverling a *signal* interrupts system calls
- Hardware interrupts do not interrupt system calls (the kernel supports nesting of control paths)
- Rule 1: fail if the call did not have time to produce any effect

  Typically, return `EINTR`
- Rule 2: in case of partial execution (for a call where it means something), do not fail but return information allowing to determine the actual amount of partial progress

  See e.g., `read()` and `write()`

System Call Implementation

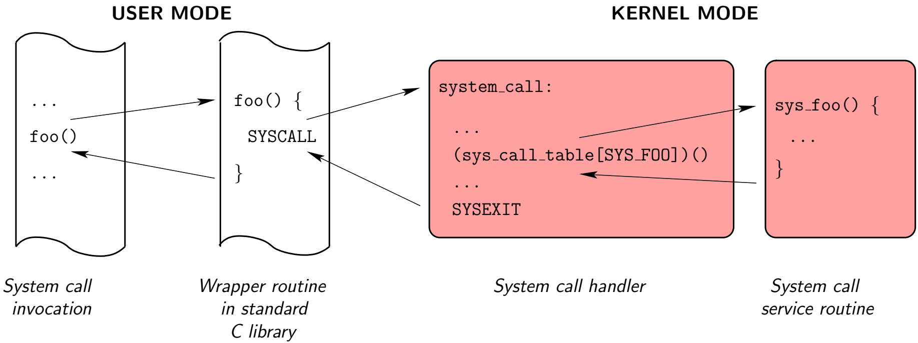
C Library Wrapper

- All system calls defined in OS-specific header file
  Linux: `/usr/include/sys/syscall.h` (which includes `/usr/include/bits/syscall.h`)
- System call handlers are numbered
- C library wraps processor-specific parts into a plain function

System Call Implementation

C Library Wrapper

- All system calls defined in OS-specific header file
  Linux: `/usr/include/sys/syscall.h` (which includes `/usr/include/bits/syscall.h`)
- System call handlers are numbered
- C library wraps processor-specific parts into a plain function



System Call Implementation

Wrapper's Tasks

1. Move parameters from the user stack to processor registers
   Passing arguments through registers is easier than playing with both user and kernel stacks at the same time
2. Switch to kernel mode and jump to the system call handler
   Call processor-specific instruction (`trap`, `sysenter`, ...)
3. Post-process the return value and compute `errno`
   Linux: typically negate the value returned by the service function

Handler's Tasks

1. Save processor registers into the *kernel mode stack*
2. Call the service function in the kernel
   Linux: array of function pointers indexed by system call number
3. Restore processor registers
4. Switch back to user mode
   Call processor-specific instruction (`rti`, `sysexit`, ...)

System Call Implementation

Verifying the Parameters

- Can be call-specific
  E.g., checking a file descriptor corresponds to an open file

- General (coarse) check that the address is outside kernel pages
  Linux: less than `PAGE_OFFSET`
- Delay more complex page fault checks to address translation time
  1. Access to non-existent page of the process
     ⟶ no error but need to allocate (and maybe copy) a page on demand
  2. Access to a page outside the process space
     ⟶ issue a segmentation/page fault
  3. The kernel function itself is buggy and accesses and illegal address
     ⟶ call `oops()` (possibly leading to “kernel panic”)