

9. Application/Kernel Interface

- POSIX Essentials
- Implementation

Application/Kernel Interface: System Calls

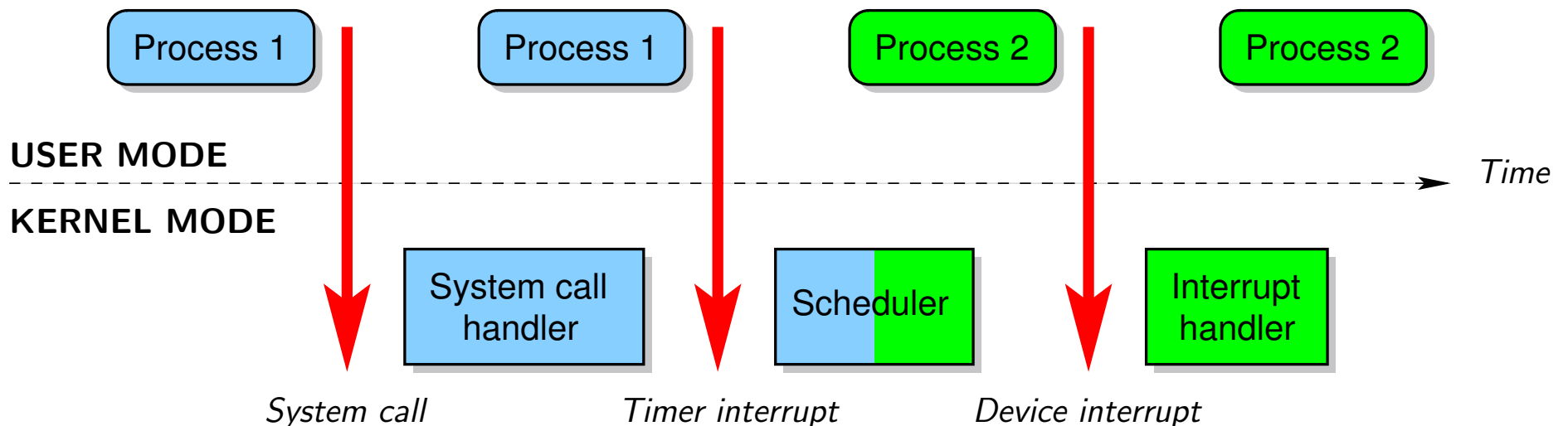
Challenge: Interaction Despite Isolation

- How to isolate processes (in memory)...
- ... While allowing them to request help from the kernel...
- ... To access resources (in compliance with security policies)...
- ... And to interact

System Call Principles

Solution: Privilege Levels

- User and kernel programs run in different privilege levels, or “modes”
 - ▶ *Kernel mode*: no restriction
 - ▶ *User mode*: restricted instructions and memory regions
- Processors provide instructions to switch between user and kernel modes
- User processes switch to kernel mode when requesting a service provided by the kernel: *system call*



9. Application/Kernel Interface

- POSIX Essentials
- 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
- C applications deal with portability with
 - ▶ C preprocessor: *conditional compilation*
 - ▶ Multi-target **Makefile** rules or GNU **configure** scripts to generate **Makefiles**
 - ▶ Shell environment variables (**LD_LIBRARY_PATH**, **LD_PRELOAD**), etc.

Should You Care About POSIX?

Modern Operating System Interfaces

- Java takes a different approach
 - ▶ A *Virtual Machine* (VM)
 - ▶ A *Standard Development Kit* (SDK)
- Yet UNIX commands and shell scripts are also normalized in POSIX
- Besides POSIX, Linux also sets its own “extended standard”
- Note: unlike GNU/Linux, Android is not a typical POSIX/UNIX system, but still uses Linux as a kernel

9. Application/Kernel Interface

- POSIX Essentials
- Implementation

System Call Implementation

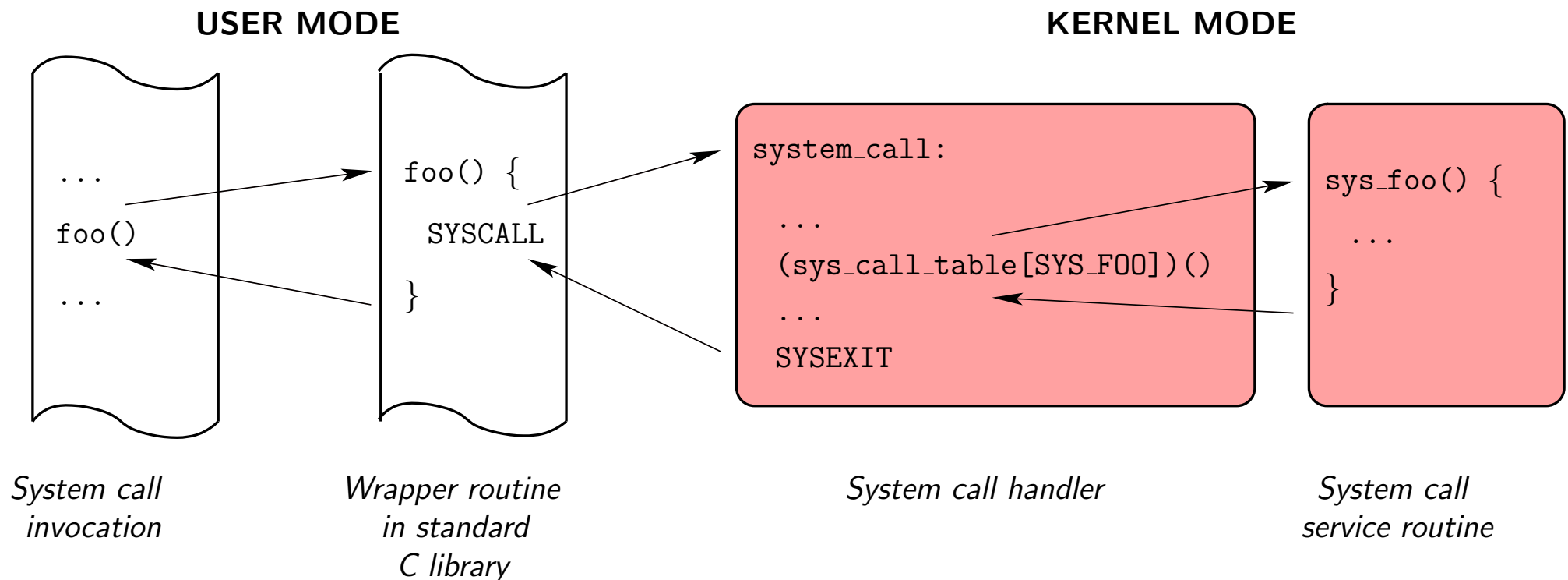
C Library Wrapper

- 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`

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 an illegal address
→ call `oops()` (possibly leading to “kernel panic”)

