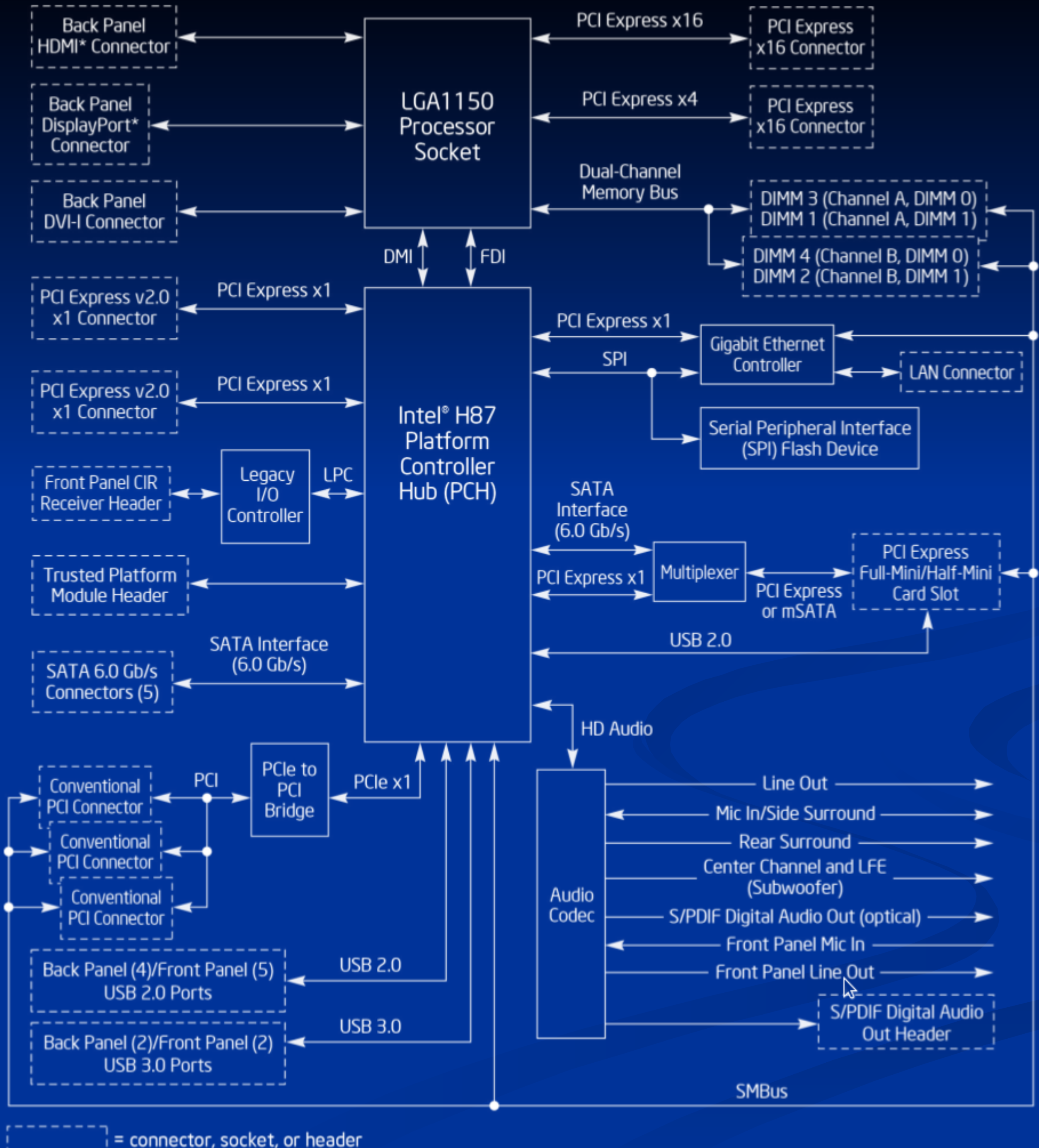
CS 25200: Systems Programming

Lecture 17: Execution Modes, Interrupts, and System Calls

Prof. Turkstra

Lecture 17

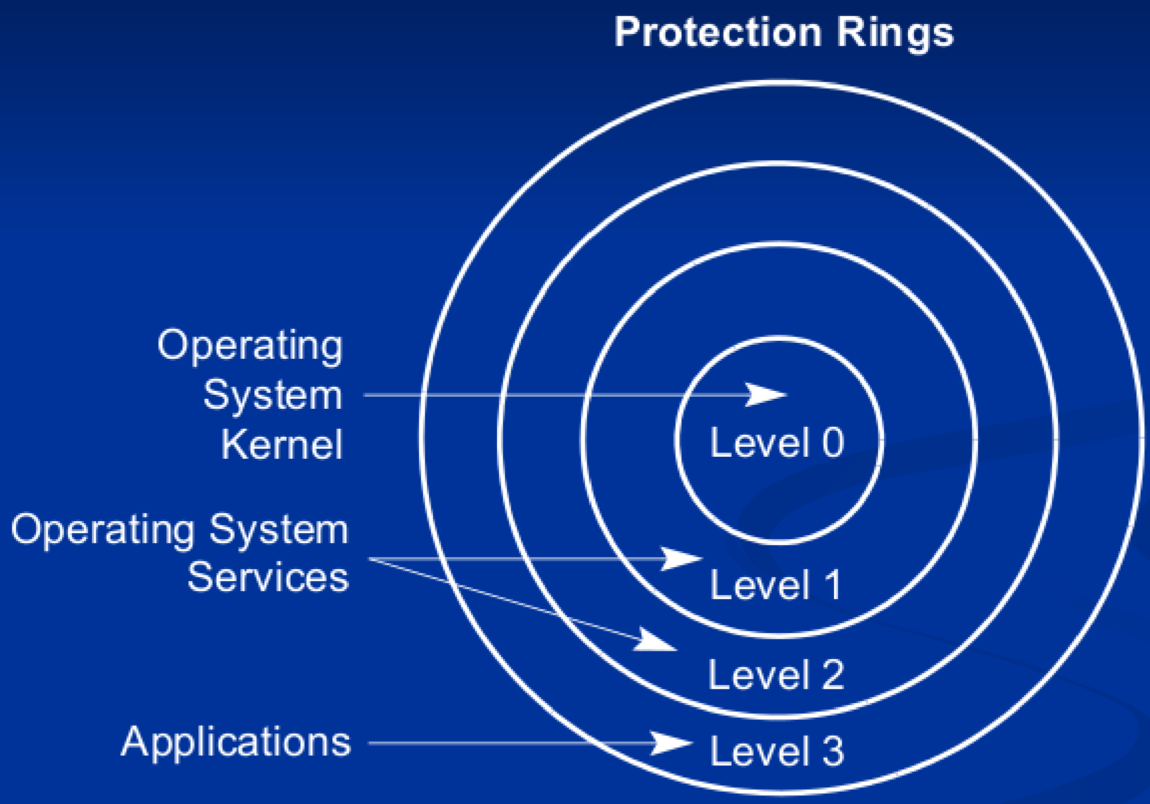
- Execution modes
- Interrupts
- System calls


Back Panel HDMI* Connector
Back Panel DisplayPort* Connector
Back Panel DVI-I Connector
LGA1150 Processor Socket
PCI Express x16
PCI Express x16 Connector
PCI Express x4
PCI Express x16 Connector
Dual-Channel Memory Bus
DIMM 3 (Channel A, DIMM 0)
DIMM 1 (Channel A, DIMM 1)
DIMM 4 (Channel B, DIMM 0)
DIMM 2 (Channel B, DIMM 1)
DMI
FDI
PCI Express v2.0 x1 Connector
PCI Express x1
PCI Express v2.0 x1 Connector
PCI Express x1
Intel® H87 Platform Controller Hub (PCH)
PCI Express x1
Gigabit Ethernet Controller
SPI
LAN Connector
Serial Peripheral Interface (SPI) Flash Device
Front Panel CIR Receiver Header
Legacy I/O Controller
LPC
SATA Interface (6.0 Gb/s)
Multiplexer
PCI Express x1
PCI Express or mSATA
PCI Express Full-Mini/Half-Mini Card Slot
Trusted Platform Module Header
USB 2.0
SATA Interface (6.0 Gb/s)
SATA 6.0 Gb/s Connectors (5)
HD Audio
Conventional PCI Connector
Conventional PCI Connector
Conventional PCI Connector
PCI
PCIe to PCI Bridge
PCIe x1
Audio Codec
Line Out
Mic In/Side Surround
Rear Surround
Center Channel and LFE (Subwoofer)
S/PDIF Digital Audio Out (optical)
Front Panel Mic In
Front Panel Line Out
S/PDIF Digital Audio Out Header
Back Panel (4)/Front Panel (5) USB 2.0 Ports
USB 2.0
Back Panel (2)/Front Panel (2) USB 3.0 Ports
USB 3.0
SMBus
= connector, socket, or header

Execution modes

- CPU hardware has several possible modes
  - At any one time, in one mode
- Modes specify
  - Privilege level
  - Valid instructions
  - Valid memory addresses
  - Size of data items
  - Backwards compatibility

Rings


Protection Rings
Operating System Kernel
Operating System Services
Applications
Level 0
Level 1
Level 2
Level 3

Ring -1

- Intel Active Management Technology
- Exists for other architectures as well
- Runs on the Intel Management Engine (ME)
  - Isolated and protected coprocessor
  - Embedded in all current Intel chipsets
  - ARC core
  - Out-of-band access
  - Direct access to Ethernet controller
- Requires vPro-enabled CPU/Motherboard/Chipset

Ring -1

- …if you can exploit it, you win.
  - CVE-2017-5689
- Go read about it

How to change between modes

- Automatic
  - Hardware interrupts
  - OS-specified handlers
- “Manual”
  - Initiated by software, typically OS
  - System calls, signals, and page faults
  - Sometimes mode can be set by application

Kernel mode

- Most OSs only use two rings, 0 and 3
- Ring 0 is kernel mode
  - Can run any instruction
  - Can modify any memory location
  - Can access and modify any registers
  - Full control of the computer

User mode

- Ring 3 is user mode
- Instructions are limited
- Only has access to defined sections of memory
  - ...cannot modify the page table or PTEs
- Can access a subset of registers
- Privileged actions must go through the OS

Booting

- CPU always starts in kernel mode
- OS is bootstrapped
  - Sets up interrupt vectors
  - Initializes devices
  - Sets up the first process (init/systemd)
  - Switches to usermode
- Begins executing first process
  - Executes daemons
  - Starts a login manager

Kernel and user mode

- User programs run in user mode
- System switches to kernel mode when resources or services are needed
- Interrupts also run in kernel mode
  - Interrupt vector
- Most CPU time is spent in user mode

Separation

- This separation gives us…
  - Security – can enforce restrictions and privileges on user mode requests
  - Robustness – provides protection between processes (invalid memory accesses, malicious system calls, etc)
    - Crashes impact only the process that crashed
    - User mode bugs do not impact other processes nor the kernel
  - Fairness – OS can ensure that resources are made available in a "fair" way

Kernel invocation

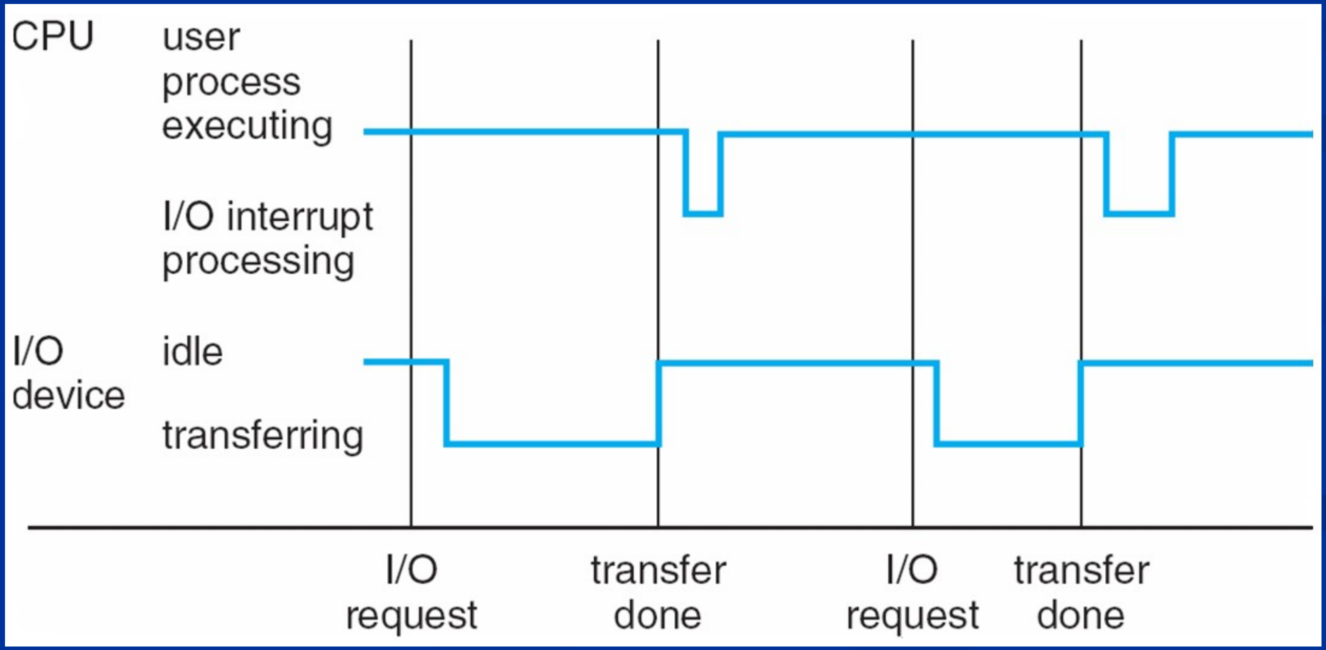
- What causes the switch to kernel mode?
  - System calls
  - Page faults
  - Signals
  - Hardware

Interrupts

- Signal to the processor that indicates an event needing immediate attention
  - Hardware interrupts – disk controller, keyboard, mouse, network card, etc
    - Often a wire with a signal that is asserted – interrupt-request line
  - Software interrupts – called traps or exceptions. E.g., INT and SYSCALL instructions, divide by zero, etc

Handling

- Interrupts halt the current execution stream and cause execution to resume somewhere else
  - Remember the program counter?



Interrupt service routine (ISR)

- Hardware sets privilege mode (kernel)
- Saves PC, maybe some registers
- Jumps to predefined entry point in the kernel
  - Interrupt vector table
- ISR then…
  - Often masks interrupts (edge vs. level triggered)
    - Nested interrupts
  - Saves any registers
  - Sets up a stack (maybe)
  - Does its work
  - Restores registers
  - IRET

Intel interrupts

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Interrupt vector table

- Indexed by interrupt number
- Holds the address of the appropriate ISR
- Can only be modified in kernel mode
  - Usually initialized during boot and driver loading

Device drivers

- Device drivers register ISRs
  - Sometimes the driver is the ISR
  - Other times it sets up a more involved process

Instruction restarting

- Sometimes the instruction that caused the interrupt should be retried
  - Page faults
- Sometimes interrupt handlers can cause interrupts (e.g., another page fault)
  - Double fault, triple fault

Polling

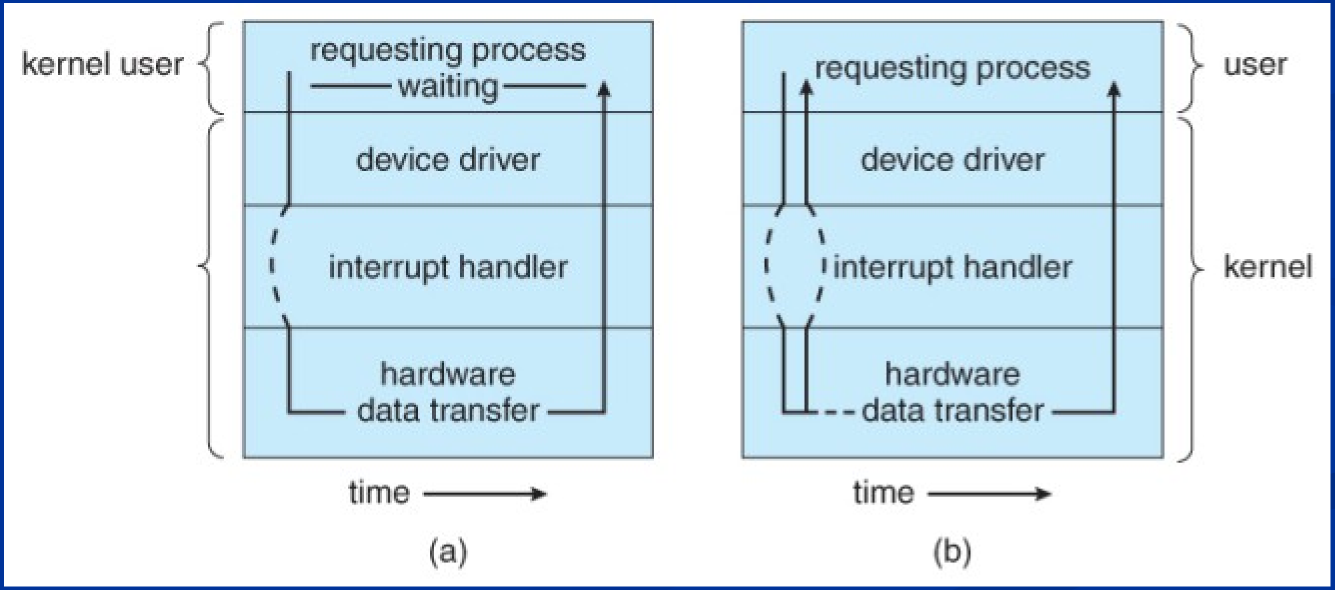
- What if we didn't have interrupts?

```
init_operation();
while (!is_operation_done());
// resume execution
```

- "Busy waiting" or "polling"
- Used rarely now – kprintf(), for example

Synchronous vs. Asynchronous

- Polling is considered to be synchronous in nature
- Interrupt handling is asynchronous



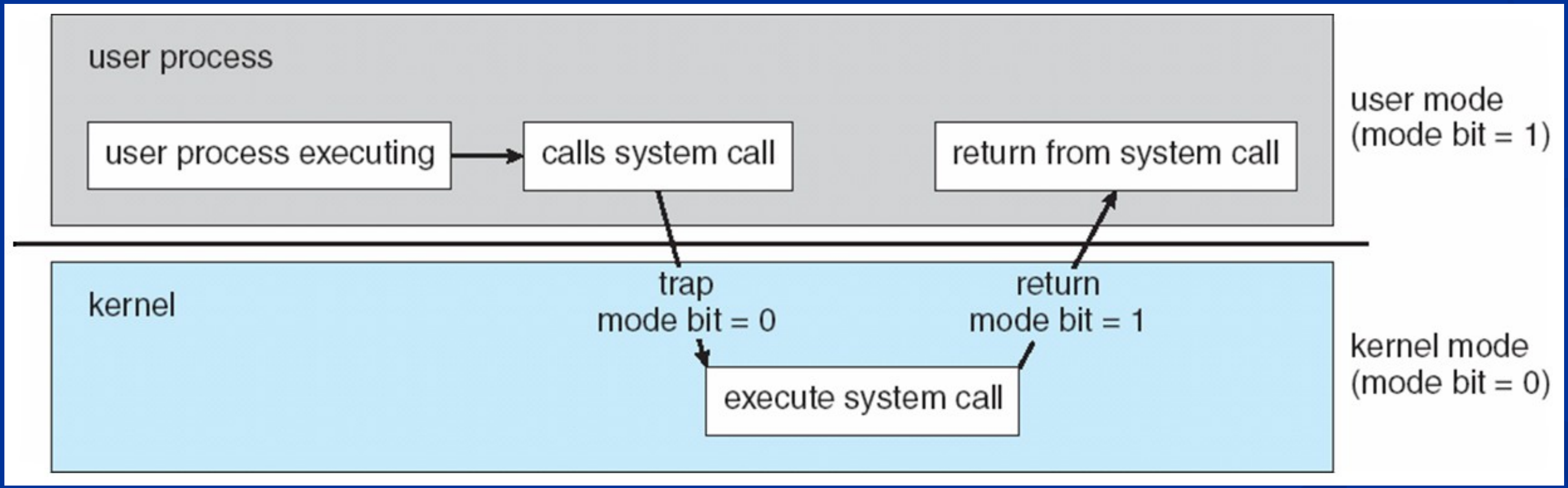
Types of interrupts

- Device interrupts
  - Mouse, keyboard, network interface card (NIC), hard drive, CD/DVD
- Math exceptions
  - Divide by zero
- Page faults – generated by the MMU
  - Invalid address
  - Swapped page
  - Invalid permissions

- Software interrupt – special instruction
  - E.g., INT and SYSCALL

System calls

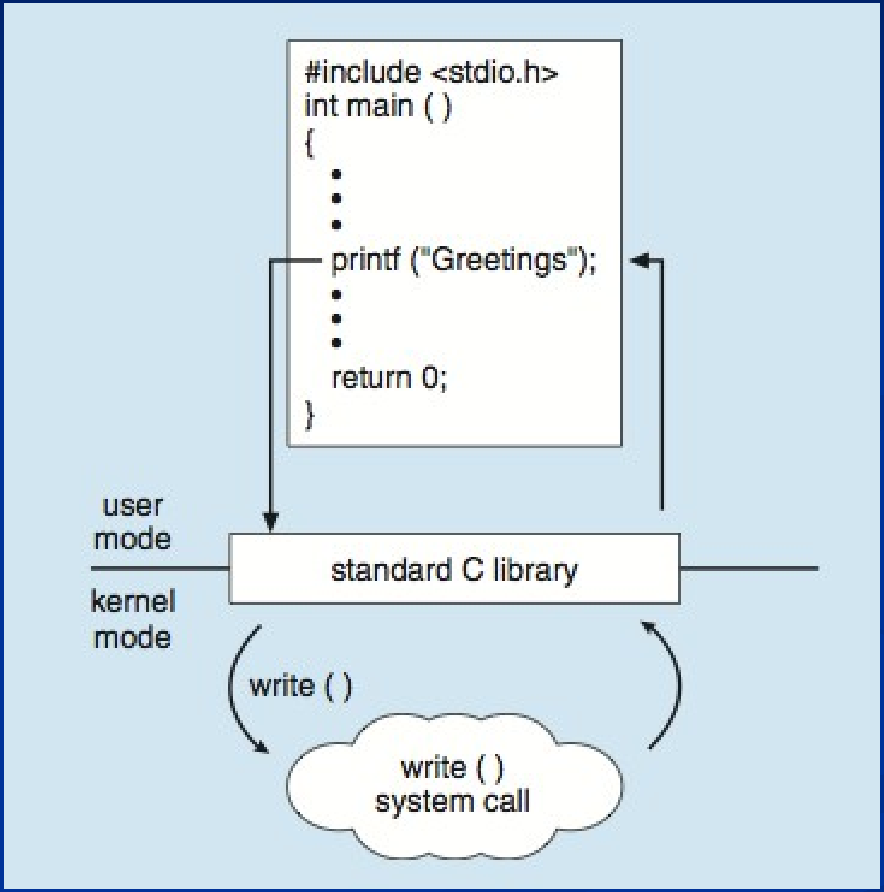
- System calls are the interface between processes and the OS kernel



System call types

- Process management
  - Create, terminate, execute, wait, etc
- File management
  - Create file, delete, open, close, read write, getattr, setattr, etc
- Device management
  - ioctl, read, write, etc
- Information management
  - getpid, alarm, sleep, etc
- Communication – between processes
  - pipe, shmget, mmap, etc

Standard C library



System V AMD64 ABI

- Used on Solaris, Linux, FreeBSD, and macOS
- RDI, RSI, RDX, RCX, R8, and R9
  - integer or pointer arguments
  - R10 instead of RCX for kernel
- XMM0-7
  - Floating point
- Additional arguments on the stack
- Return value in RAX and RDX

- Callee must save and restore RBP, RBX, R12-R15 if used
  - Not for system calls
- Lots of details
  - http://refspecs.linuxbase.org/elf/x86-64-abi-0.99.pdf

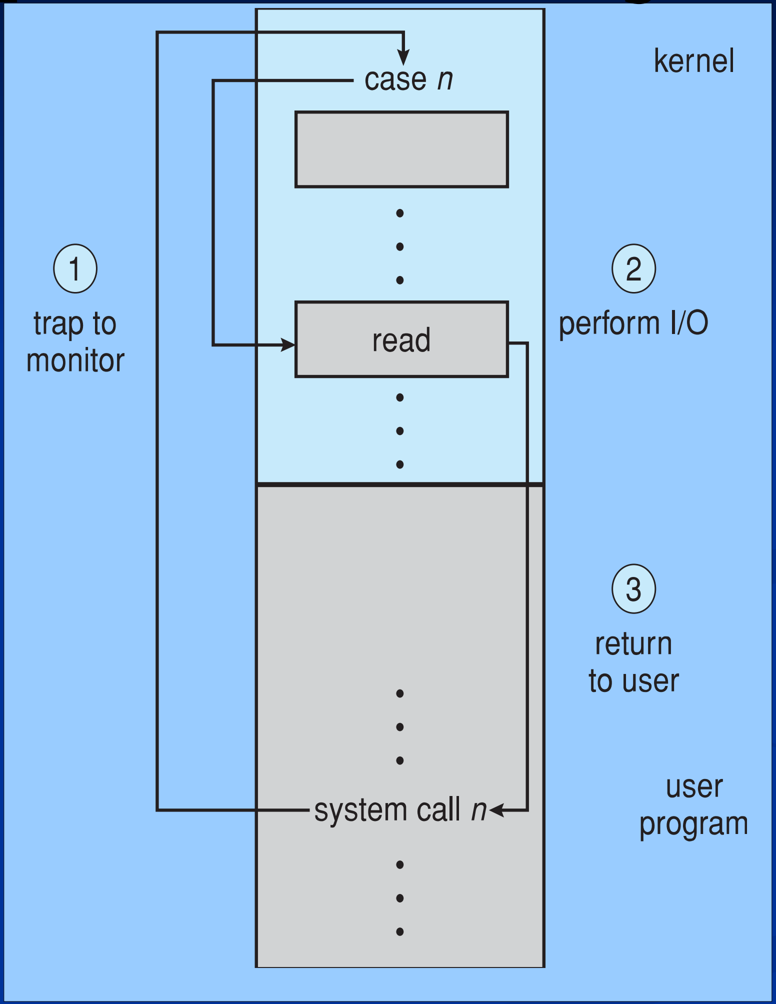
Why not function calls?

- System calls deal with privileged operations
  - Need privileged instructions
  - Access to devices and kernel data structures
  - Mechanism by which the kernel’s security policy is enforced

Libraries and wrappers

- Most code does not invoke system calls directly
- Done through libraries and wrapper functions
  - E.g., printf() and malloc()
- Try to do as much work in user mode as possible
  - Context switches are expensive

Example read system call



Synchronous write()

- Program executes the libc wrapper for write():
  write(fd, buff, n);
- libc places the arguments in the appropriate registers or stack locations
- libc then invokes syscall, which generates a software interrupt

- The OS interrupt handler checks the system call number and jumps to the appropriate location
- The handler verifies..
  - The fd is an open fd
  - Has the correct privileges (read/write/etc)
  - That [buff, buff+n-1] is a valid memory range
- Returns -1 and sets errno for failures

- OS determines the block(s) corresponding to the current file position by inspecting the inode
  - Also updates current file position
- OS sets up a DMA operation with the hard drive that takes the memory at buff, up to buff+n-1, and writes it to the appropriate block(s) address
- OS places current process in wait state

- OS switches to another process
- Disk completes write operation and generates a hardware interrupt
- OS jumps to appropriate ISR, writes the return value to rax and IRETs
- OS places process in the ready state
  - Available for scheduling

Security

- The checks that the kernel does on system call entry are critical
  - Never directly inspects user memory
- E.g., for open()…
  - Get file name and mode
  - Check if file exists
  - Verify permissions
  - Return fd

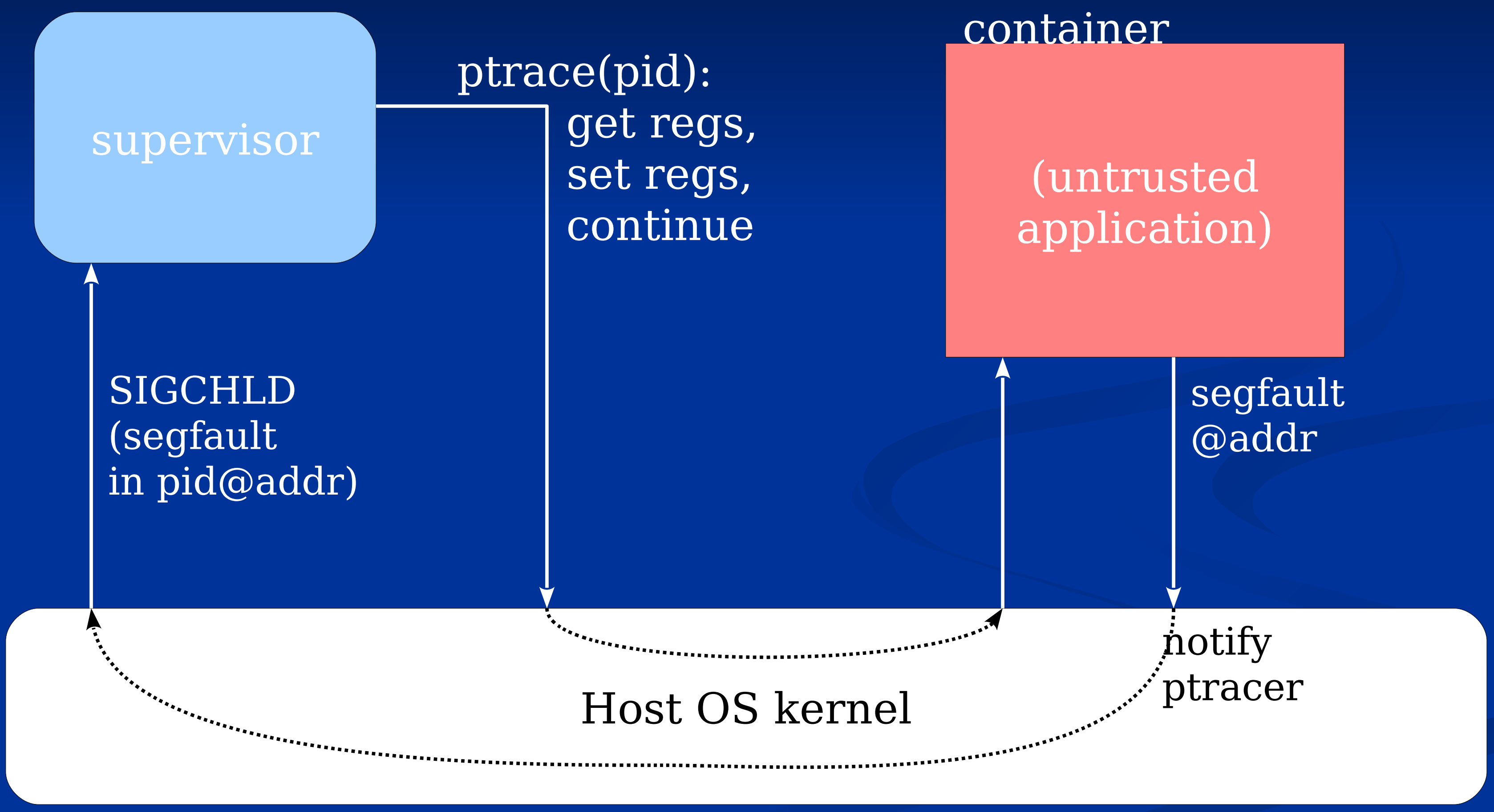
Errors

- Remember errno.h?
- System calls often return -1 and set errno
- man errno
- /usr/include/errno.h

strace

- Traces system calls and signals
  - Relies on the ptrace() system call
    - a “parent process” observes/controls another process
    - Can change child's core image and registers
    - Suspends child, wakes parent on *all* exceptional events

Handling a page fault



Interception slowdown

Type of container exception	Native (cycles)	Virtual (cycles)	Penalty
call `getpid()` (min)	786.0	59442.0	75.6x
(average)	1567.9	188051.6	119.9x
read fault	1329.5	90063.1	67.7x
write-after-read fault	3589.3	81826.3	22.8x
direct write double-fault	2924.4	170895.0	58.4x

Questions?