Architectural Support for Operating Systems

Jo, Heeseung

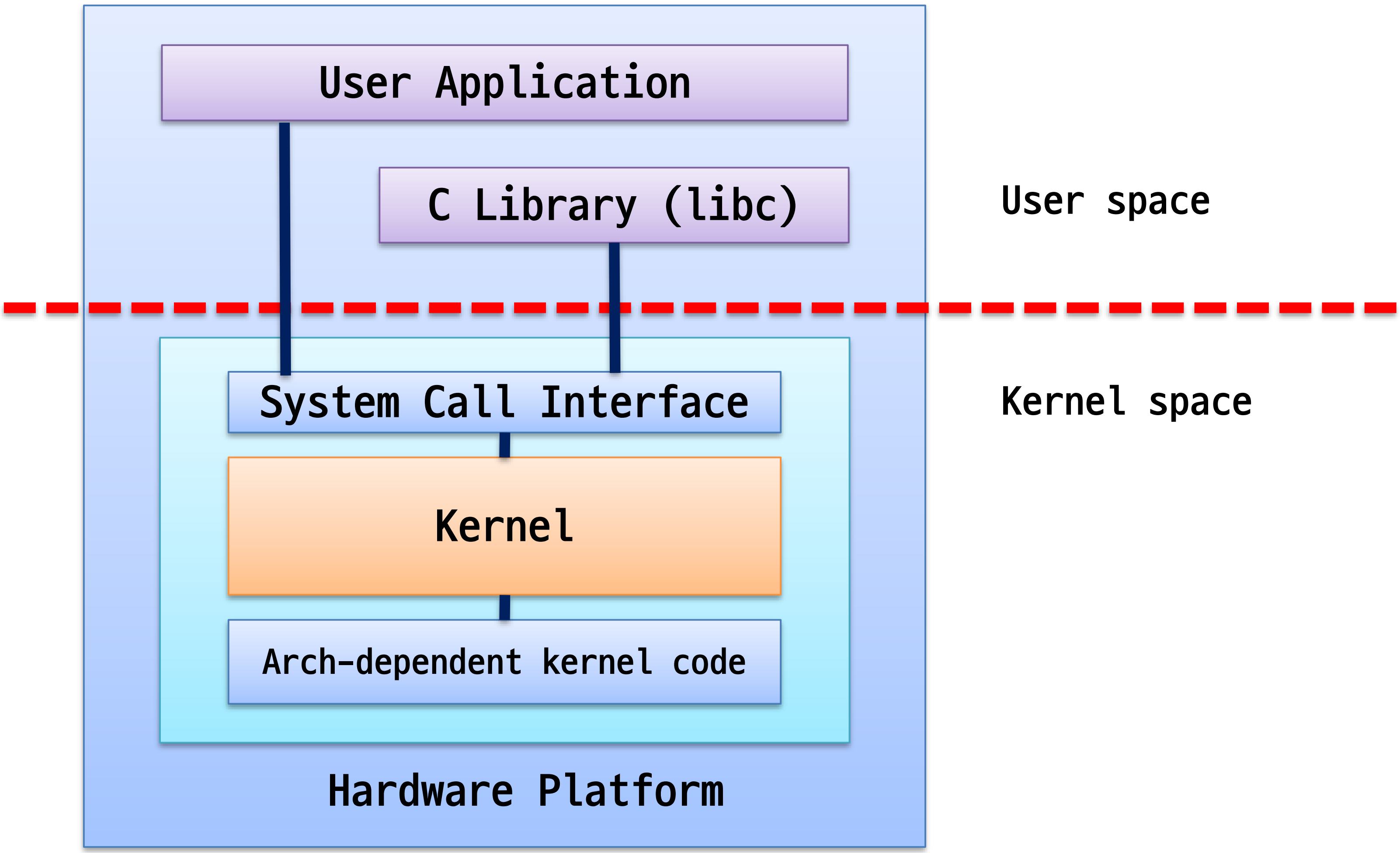
Today's Topics

Basic structure of OS

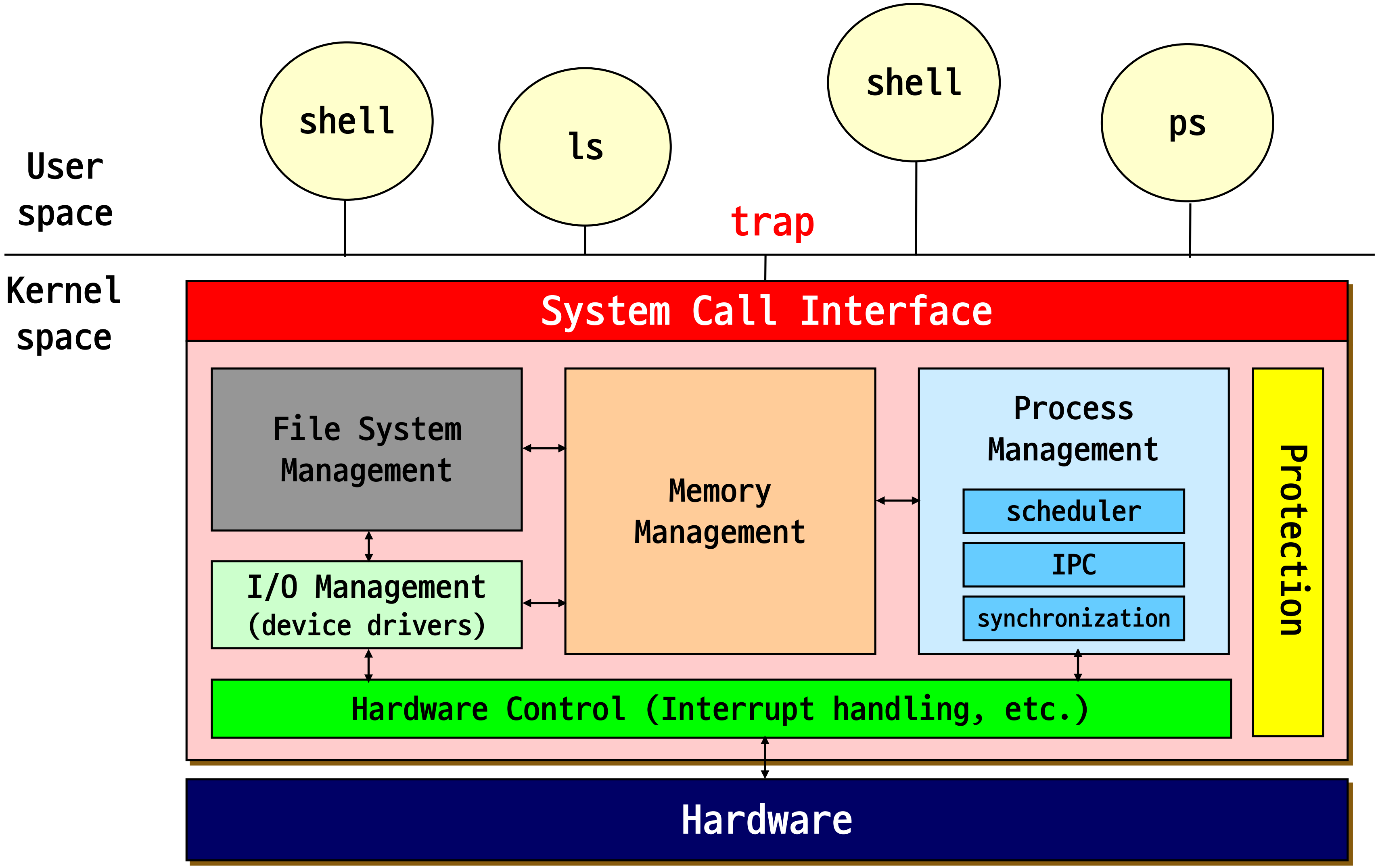
Basic computer system architecture

Architectural support for OS

OS Internals (1)

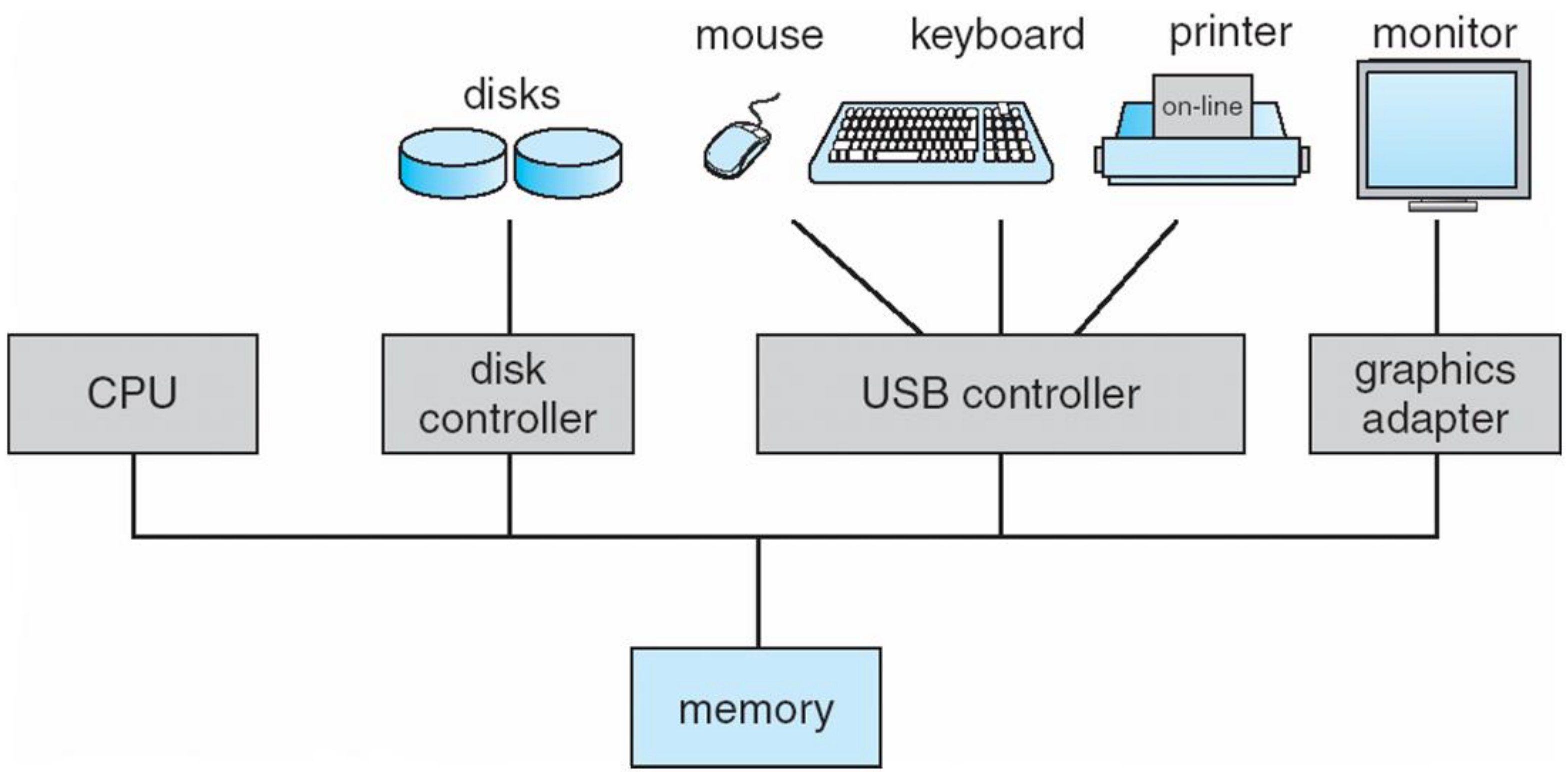


OS Internals (2)



Computer Systems (1)

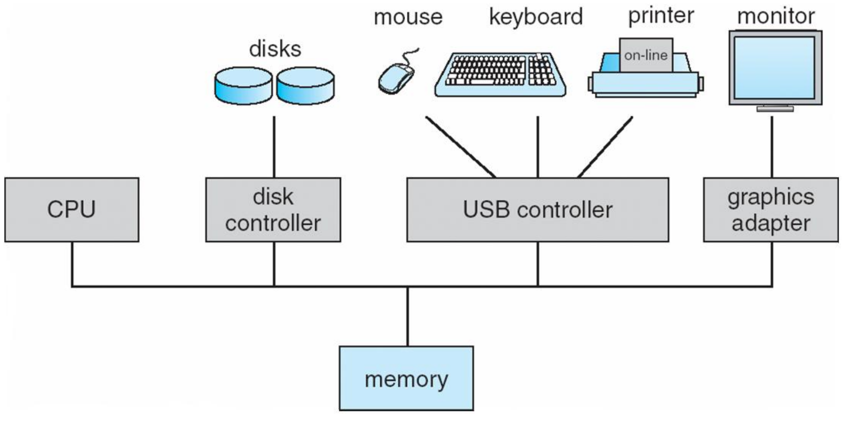
Computer system organization



Computer Systems (2)

Characteristics

- I/O devices and CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is data movements between main memory and the local buffer of controller
- CPU issues specific commands to I/O devices
- CPU should be able to know whether the issued command has been completed or not



OS and Architecture

Mutual interaction

- The functionality of an OS is limited by architectural features
  - Multiprocessing on DOS/8086?
- The structure of an OS can be simplified by architectural support
  - Interrupt, DMA, synchronization, Intel-VT/AMD-V, etc.
- Most proprietary OS's were developed with the certain architecture in mind
  - SunOS/Solaris for SPARC architecture
  - IBM AIX for Power/PowerPC architecture
  - HP-UX for PA-RISC architecture
  - ...

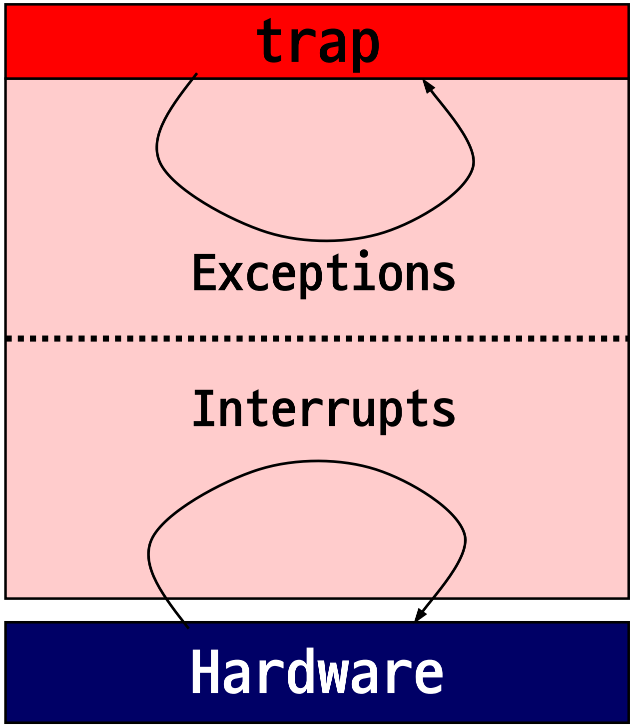
Interrupts (1)

Interrupts

- Generated by hardware devices
  - Triggered by a signal in INTR or NMI pins (x86)

Exceptions

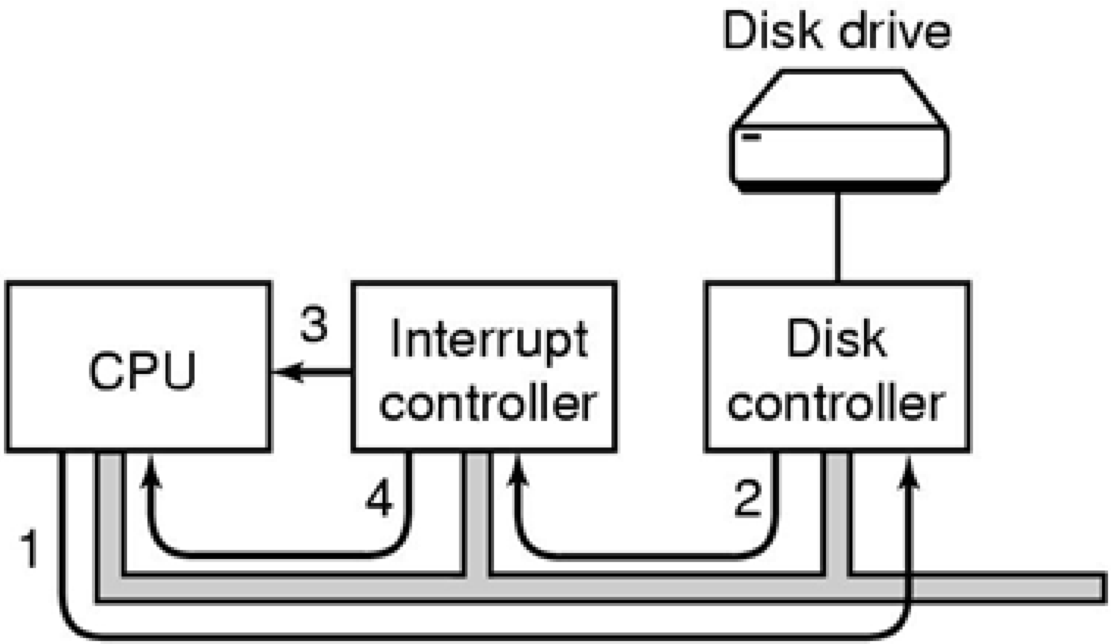
- Generated by software executing instructions
  - INT instruction in x86
- Exception handling is same as interrupt handling



Interrupts (2)

How does the kernel notice an I/O has finished?

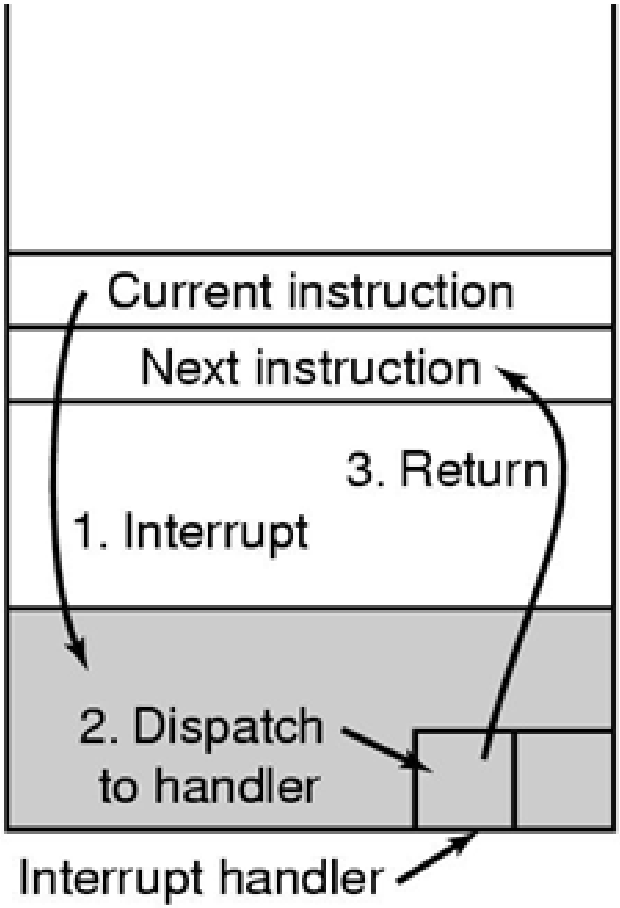
- Polling
- Hardware interrupt



Interrupts (3)

Interrupt handling

- Preserves the state of the CPU
  - In a fixed location
  - In a location indexed by the device number
  - On the system stack
- Determines the type
  - Vectored interrupt system
- Transfers control to the interrupt service routine (ISR) or interrupt handler



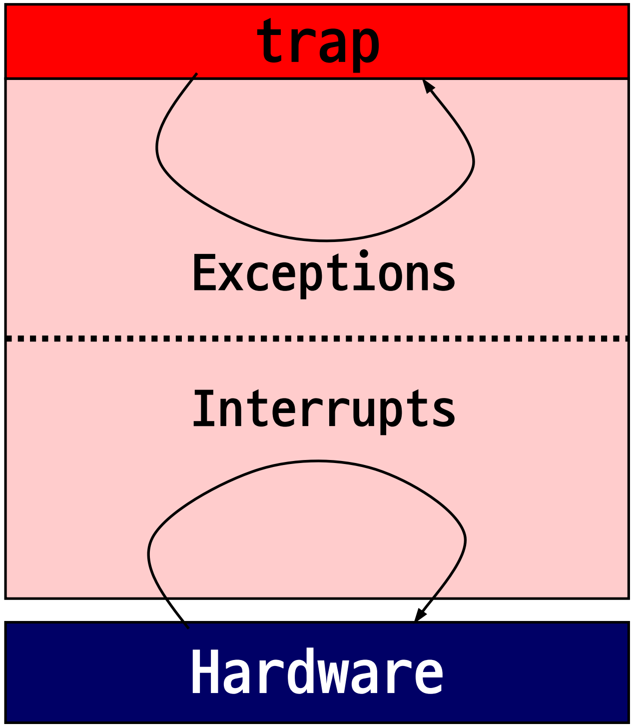
Exceptions (1)

Interrupts

- Generated by hardware devices
  - Triggered by a signal in INTR or NMI pins (x86)

Exceptions

- Generated by software executing instructions
  - INT instruction in x86
- Exception handling is same as interrupt handling



Exceptions (2)

Further classification of exceptions

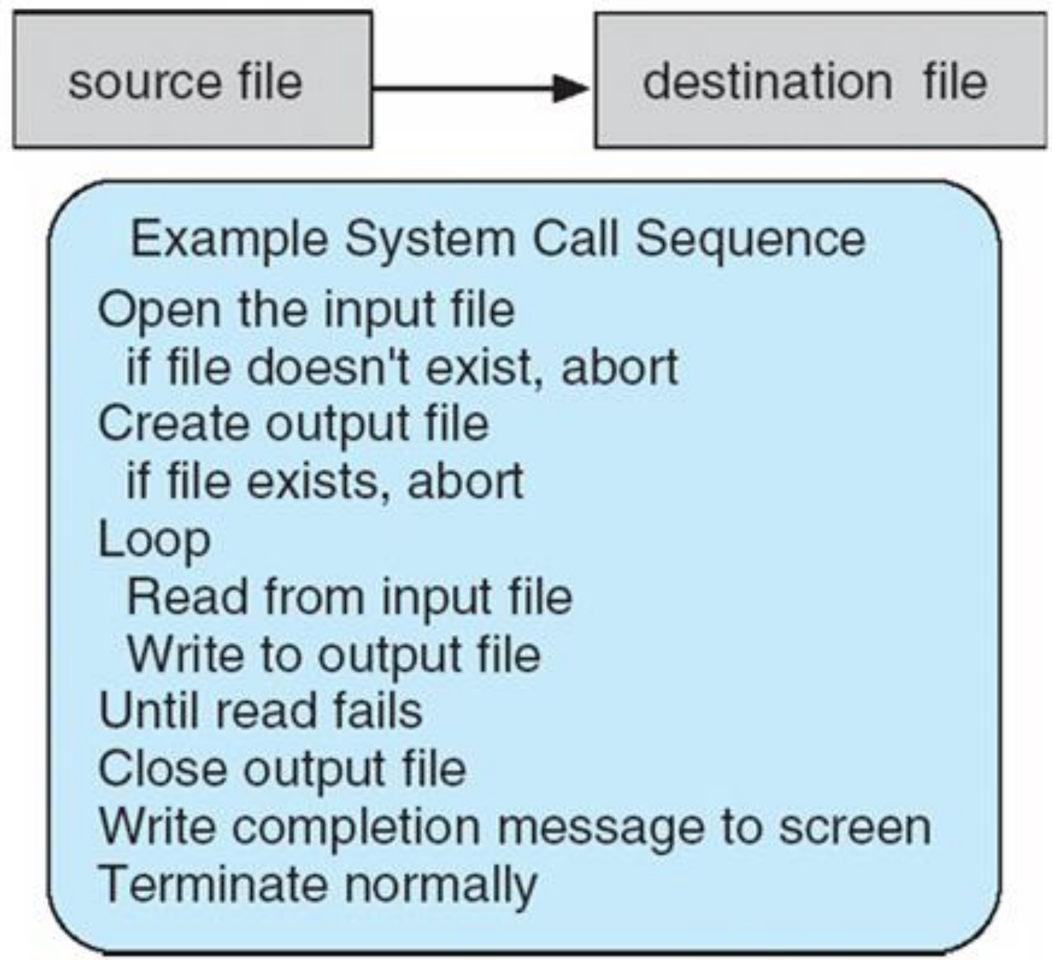
- Traps
  - Intentional
  - System calls, breakpoint traps, special instructions, ...
  - Return control to "next" instruction
- Faults
  - Unintentional but possibly recoverable
  - Page faults (recoverable), protection faults, ...
  - Either re-execute faulting ("current") instruction or abort
- Aborts
  - Unintentional and unrecoverable
  - Parity error, machine check, ...
  - Abort the current program

Exceptions (3)

System calls

- Programming interface to the services provided by OS
- e.g., system call sequence to copy the contents of one file to another

cp a.txt b.txt



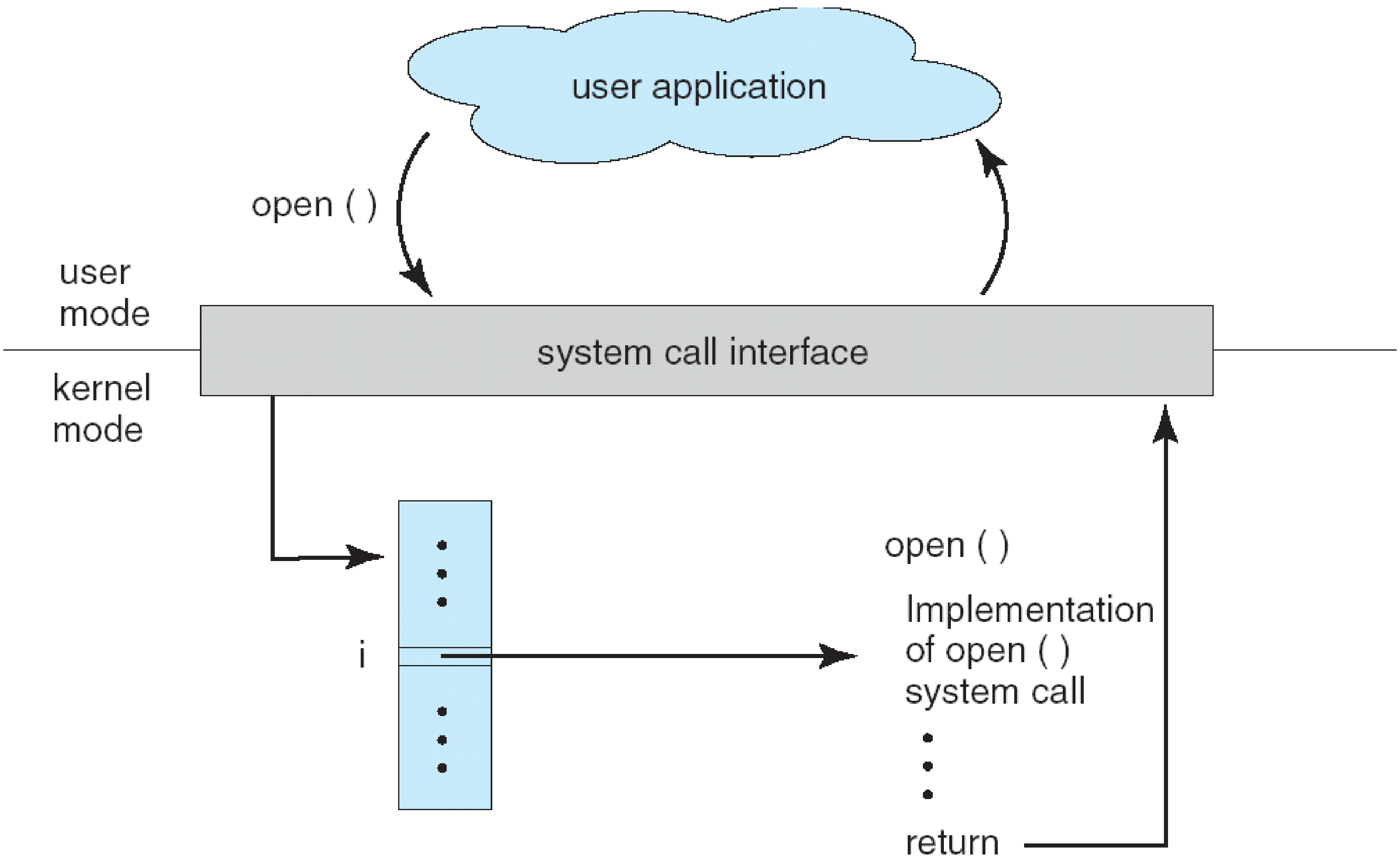
Exceptions (4)

Important system calls (POSIX & Win32)

Process Management	fork	CreateProcess	Create a new process
	waitpid	WaitForSingleObject	Wait for a process to exit
	execve	(none)	CreateProcess = fork + execve
	exit	ExitProcess	Terminate execution
	kill	(none)	Send a signal
File Management	open	CreateFile	Create a file or open an existing file
	close	CloseHandle	Close a file
	read	ReadFile	Read data from a file
	write	WriteFile	Write data to a file
	lseek	SetFilePointer	Move the file pointer
	stat	GetFileAttributesEx	Get various file attributes
	chmod	(none)	Change the file access permission
File System Management	mkdir	CreateDirectory	Create a new directory
	rmdir	RemoveDirectory	Remove an empty directory
	link	(none)	Make a link to a file
	unlink	DeleteFile	Destroy an existing file
	mount	(none)	Mount a file system
	umount	(none)	Unmount a file system
	chdir	SetCurrentDirectory	Change the curent working directory

Exceptions (5)

Implementing system calls

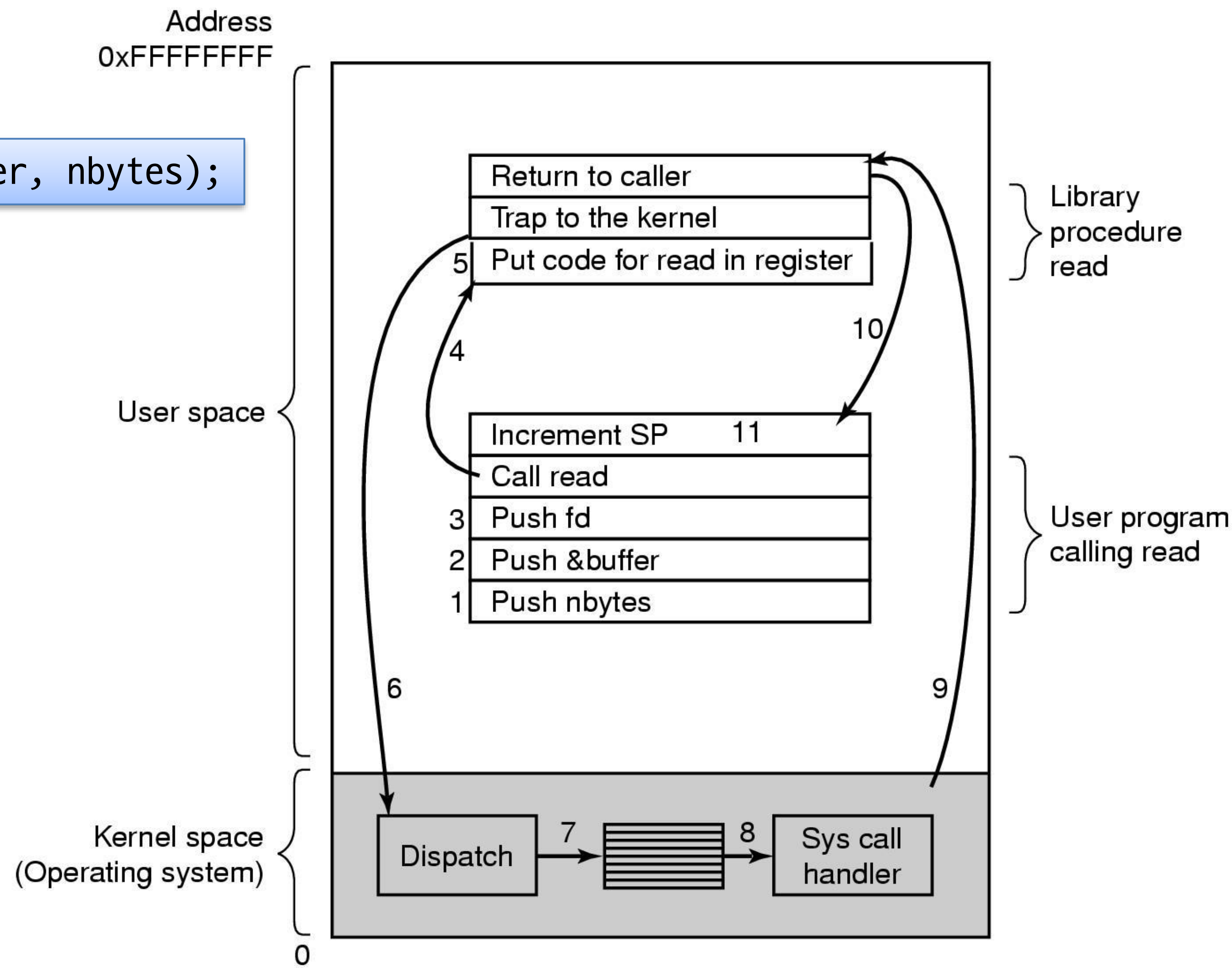


System call table

Exceptions (6)

Implementing system calls (cont'd)

```
count = read (fd, buffer, nbytes);
```



Signals (1)

Signals

- Standardized messages sent to a running program (process)
- To trigger specific behavior, such as quitting or error handling
- A limited form of inter-process communication (IPC)
- Asynchronous

Common uses

- To interrupt, suspend, terminate or kill a process or thread

```
ubuntu@DESKTOP-P53HG96:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

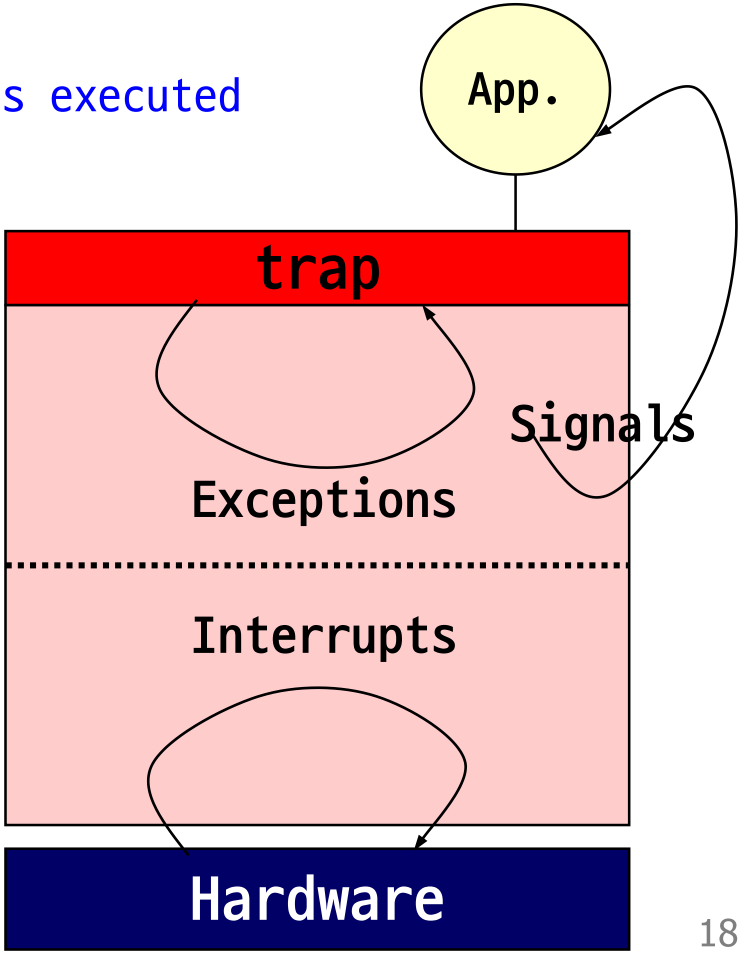
Signals (2)

When a signal is sent

- The operating system interrupts the target process' normal flow of execution to deliver the signal
- If the process has previously registered a signal handler, that routine is executed
- Otherwise, the default signal handler is executed

Signals vs. interrupts

- Interrupts are mediated by the hardware and handled by the kernel
- Signals are mediated by the kernel and handled by individual processes



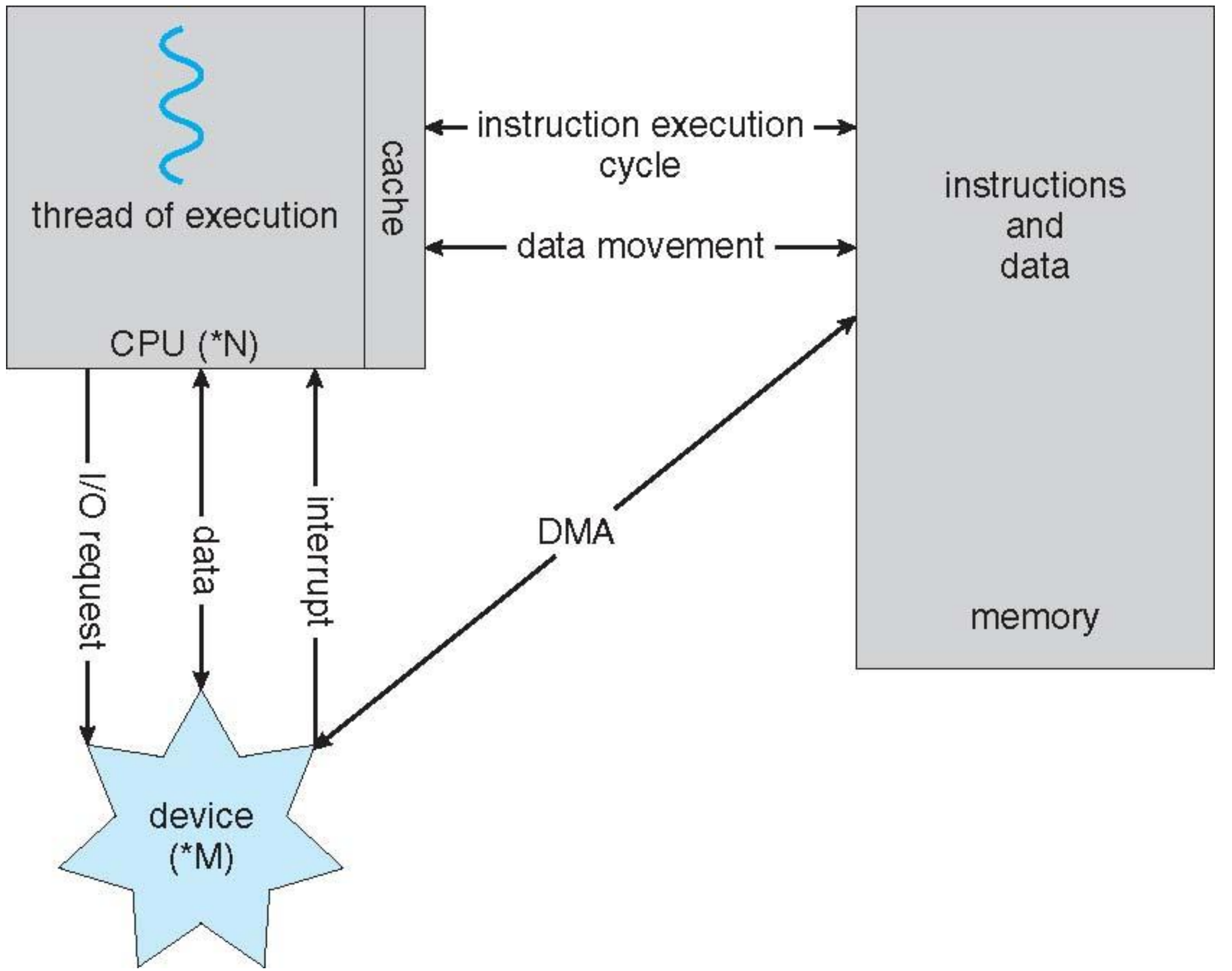
DMA (1)

Data transfer modes in I/O

- Programmed I/O (PIO)
  - CPU is involved in moving data between I/O devices and memory
  - By special I/O instructions vs. by memory-mapped I/O
- DMA (Direct Memory Access)
  - Used for high-speed I/O devices able to transmit information at close to memory speeds
  - Device controller transfers blocks of data from buffer storage directly to main memory
    - Without CPU intervention
  - Only an interrupt is generated per block

DMA (2)

Processing I/O requests



Timers

How does the OS take control of CPU from the running programs?

- Use a hardware timer that generates a periodic interrupt
- The timer interrupt transfers control back to OS
- The OS preloads the timer with a time to interrupt
  - 10ms for Linux 2.4, 1ms for Linux 2.6
  - Dynamic changing for current Linux
  - (cf.) time slice
- The timer is privileged
  - Only the OS can load it

Protected Instructions

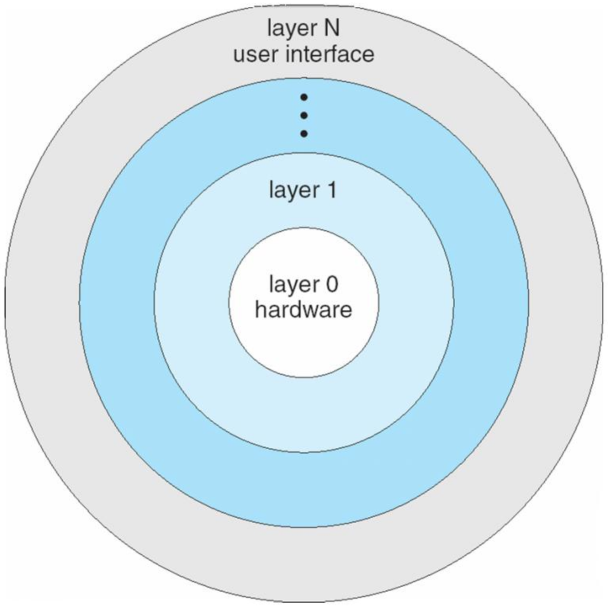
Protected or privileged instructions

- Direct I/O access
  - Use privileged instructions or memory-mapping
- Memory state management
  - Page table updates, page table pointers
  - TLB loads, etc.
- Setting special "mode bits"

OS Protection (1)

How does the processor know if a protected instruction should be executed?

- The architecture must support at least two modes of operation: kernel and user mode
  - 4 privilege levels in IA-32: Ring 0 > 1 > 2 > 3
- Mode is set by a status bit in a protected processor register
  - User programs in user mode, OS in kernel mode
  - Current Privilege Level (CPL) in IA-32: CS register
- Protected instructions can only be executed in the kernel mode



OS Protection (2)

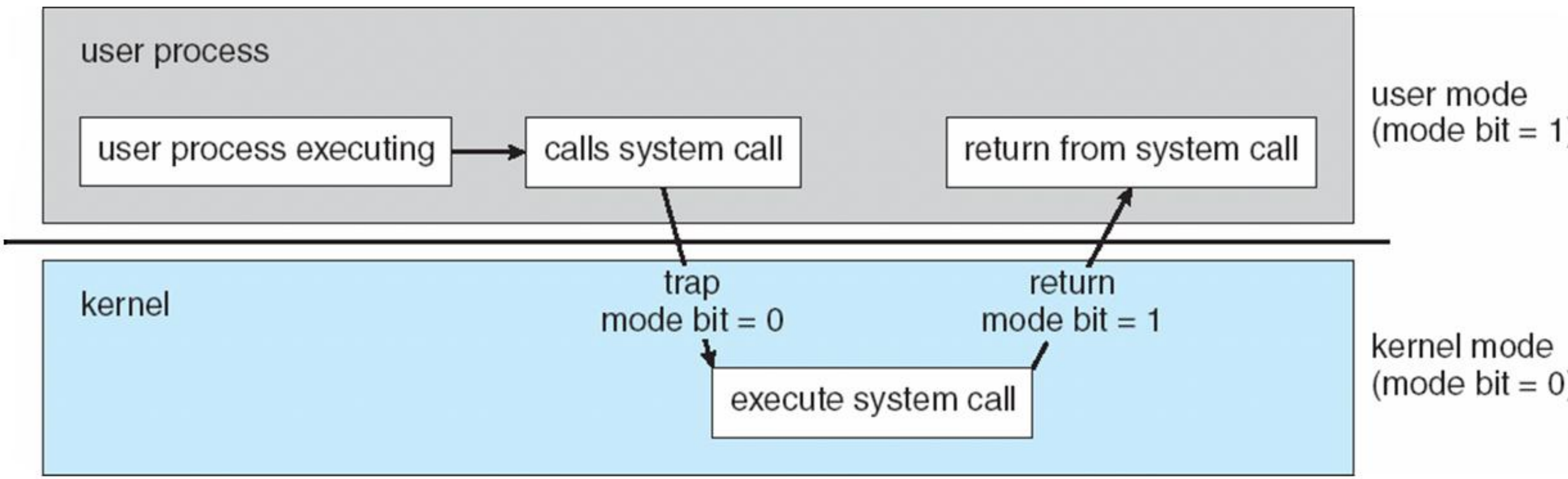
Crossing protection boundaries

- User programs must call an OS to do something privileged
  - OS defines a sequence of system calls (system call table)
- There must be a system call instruction that:
  - Causes an exception, which invokes a kernel handler
  - Passes a parameter indicating which system call to invoke
  - Saves caller's state (registers, mode bits) so they can be restored
  - OS must verify caller's parameters (e.g. pointers)
  - Must provide a way to return to user mode when done

OS Protection (3)

Making a system call

- System call changes mode to kernel
- Return from system call resets it to user



Memory Protection (1)

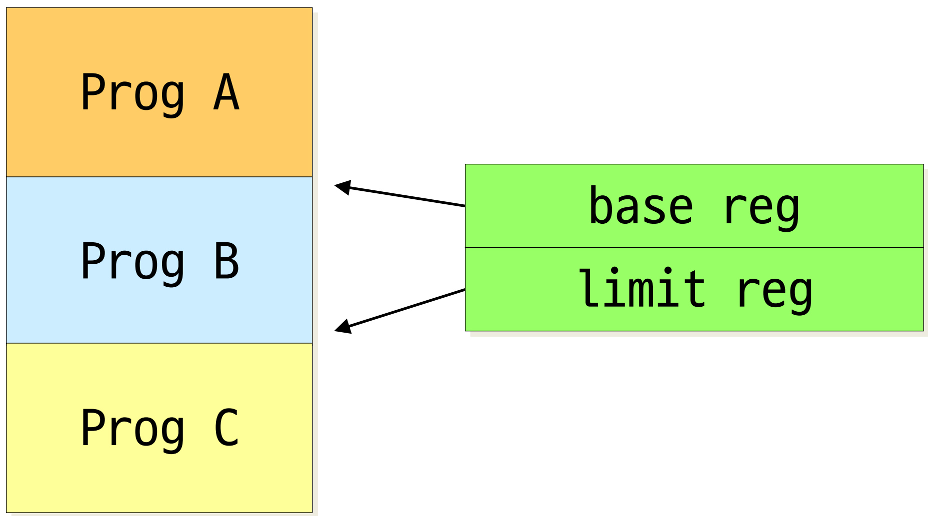
Requirements

- OS must protect user programs from each other
  - Malicious users
- OS must also protect itself from user programs
  - Integrity and security

Memory Protection (2)

Simplest scheme

- Use base and limit registers
- Base and limit registers are loaded by OS before starting a program



Memory Protection (3)

MMU (Memory Management Unit)

- Memory management hardware provides more sophisticated memory protection mechanisms
  - Base and limit registers
  - Page table pointers, page protection, TLBs
  - Virtual memory
  - Segmentation
- Manipulation of memory management hardware are protected (privileged) operations

Synchronization

Problems

- Interrupt can occur at any time and may interfere with the interrupted code
- OS must be able to synchronize concurrent processes

Synchronization

- Turn off/on interrupts
- Use a special atomic instructions
  - read-modify-write (e.g., INC, DEC)
  - test-and-set
  - LOCK prefix in IA32
  - LL (Load Locked) & SC (Store Conditional) in MIPS