System Architecture and Structure

Gabriel Parmer
csci3411: Operating Systems

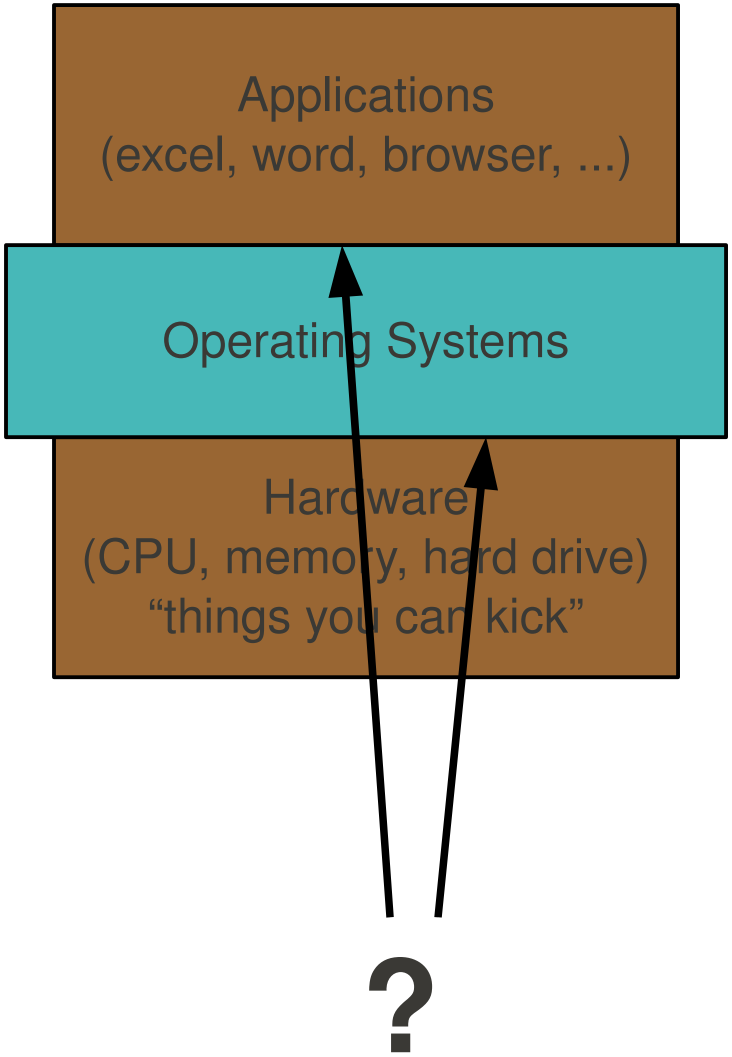
Lecture 2

Some content modified from Silberschatz etal, and West

More Administrative Details

- Piazza.com – join ASAP
- C linked list – homework due in a week.
  - Homework is really about setting up your environment...
  - vmware/virtualbox
- Labs today in Tompkins 211
- Blackboard for homework
  - Posting – Thursday night
  - Submission – Saturday by Noon

System Architecture and Structure



- How does hardware interact with the OS?
- How do applications and the OS interact?
- OS goals
  - provide desirable abstractions to applications
  - while controlling the hardware
- OS implementation/org styles

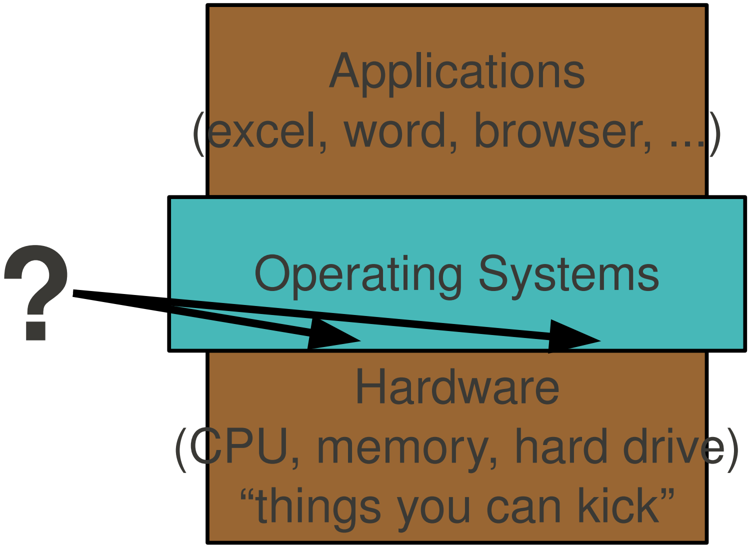
I/O Management/Communication

- Each device controller is in charge of a device type

- Each device controller has a buffer/control regs

- Protocol: To get a byte of data from a device
  - I/O is from the device to local buffer of controller
  - CPU sets registers in controller with command to read e.g. character from keyboard
    - CPU waits while data is moved from controller buffer to memory

- I/O devices and the CPU can execute concurrently
  - But CPU must actively wait as data is transferred...byte...by...byte
  - *Better way???*

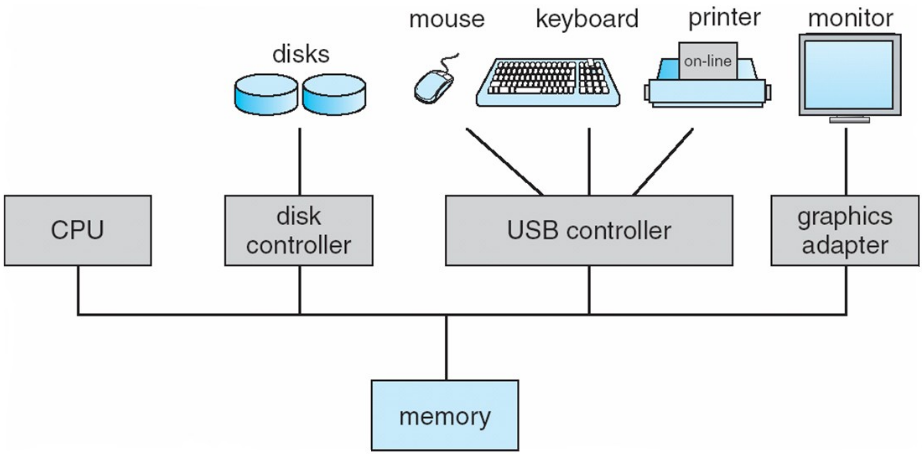


?

Applications
(excel, word, browser, ...)

Operating Systems

Hardware
(CPU, memory, hard drive)
“things you can kick”



Interrupts

- Transfer control (instruction pointer) to interrupt service routine (ISR)
  - ISR identified by address in interrupt vector
- Interrupt architecture (HW) must save address of the interrupted instruction
- After servicing interrupt, CPU resumes execution at previously interrupted address

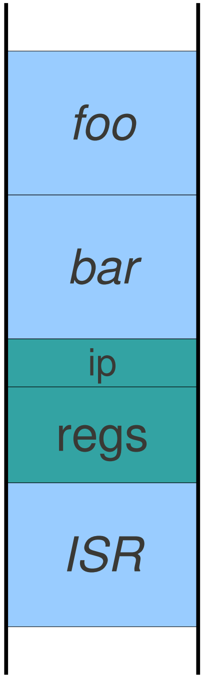
- *What about other registers?*
- *Where are they saved?*

Interrupts

- Transfer control (instruction pointer) to interrupt service routine (ISR)
  - ISR identified by address in interrupt vector
- Interrupt architecture (HW) must save address of the interrupted instruction
- After servicing interrupt, CPU resumes execution at previously interrupted address

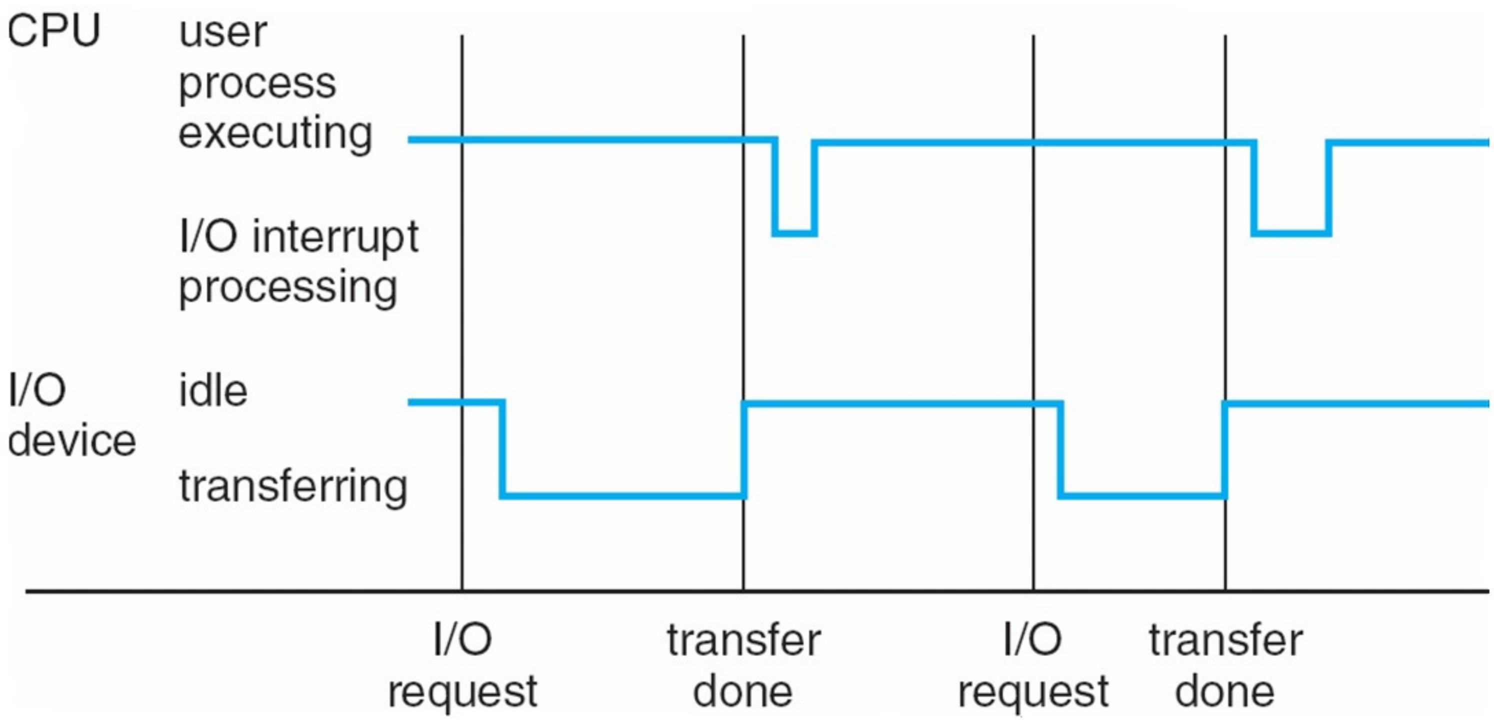
- *What about other registers?*
- *Where are they saved?*

Execution
Stack
Grows
Down (x86)

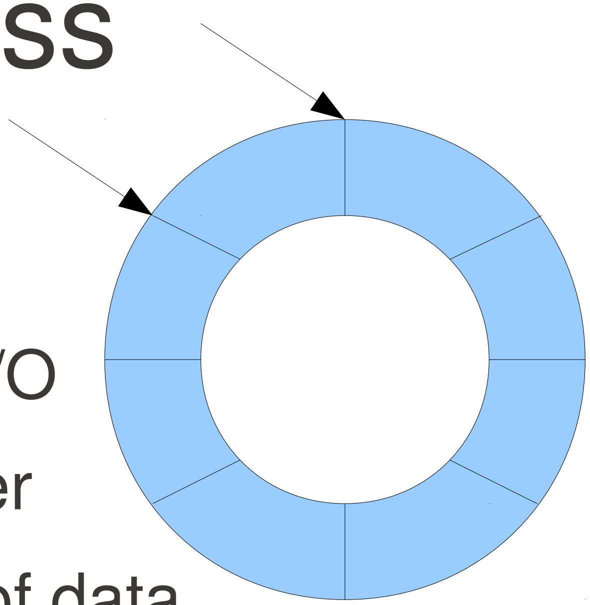


```
int foo(void) {
        bar();
}
void bar(void) {
        …
        // interrupt triggered here
        …
}
```

CPU/Device Interaction: Interrupts



Direct Memory Access



- CPU
  - Sets up (large) buffers in memory *before* I/O
  - Asks device controller to transfer into buffer
  - Receives single *interrupt* for whole buffer of data
- Device controller
  - When device I/O complete (transferred to controller's local buffers), transfer/copy data directly into memory
  - Send *interrupt* when transfer complete
  - Avoids CPU work for data-movement
- *What if transferred data is always a single byte?*

Direct Memory Access and Interrupts

- Keyboard device doesn't cause many interrupts
    - Interrupt per key press: say 100 interrupts/second
    - ISR overhead of 1 microseconds → 1/10000th CPU time

- *How about a networking card?*
    - 1 GB/second

Polling vs. Interrupts

- Polling: CPU repeatedly checks status of I/O
  - Read a device controller register
  - Has an I/O request finished, or not?
- If I/O has completed, CPU reads it into memory
- Frequency of polling impacts latency and throughput of I/O
  - So should we simply poll at the highest possible frequency?
  - *Is polling ever better than interrupts?*

Example: Polling vs. Interrupts I

- Office hours – one professor, multiple students
- Explanation to group of students takes 10 min
- Professor can do one of two things:
  1) Have students knock (interrupt); tell them to wait, or come in – 1 minute/interrupt
  2) Have students wait outside; Prof checks (polls) if students waiting every 10 minutes – 1 minute/check

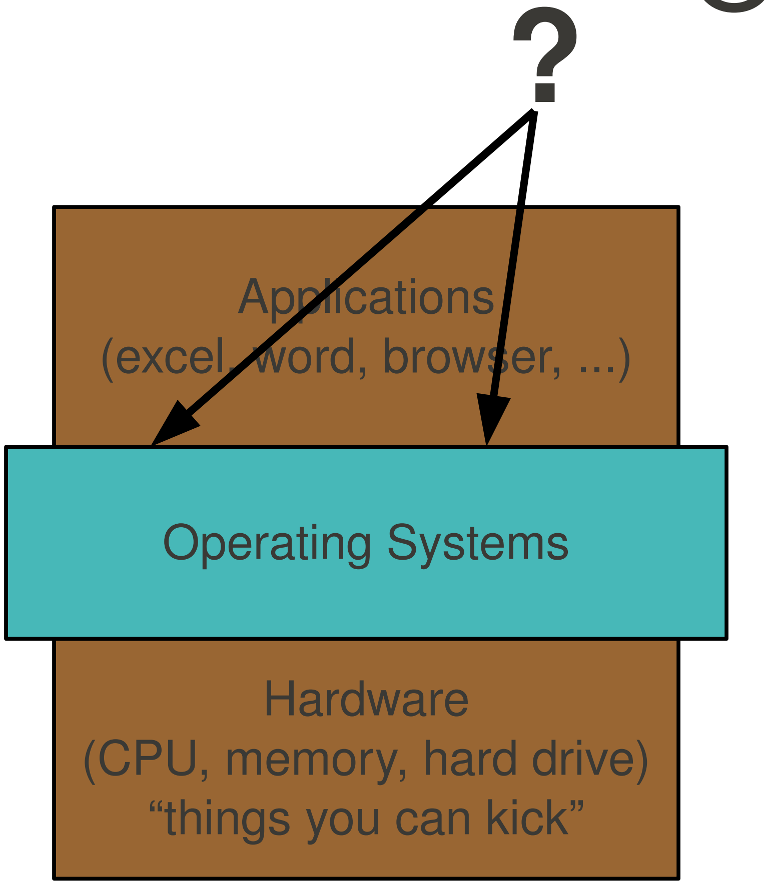
Example: Polling vs. Interrupts II

- Normal office hours: 0.5 students/hour
- Pre-test: 12 students/hour
- Ask students to write a new OS: 60 students/hour

- Which policy is best for each?
  - Minimize amount of time prof spends on polling/ints
  - Minimize amount of time students spend waiting

- Not a perfect analogy, but you get the gist

OS Services

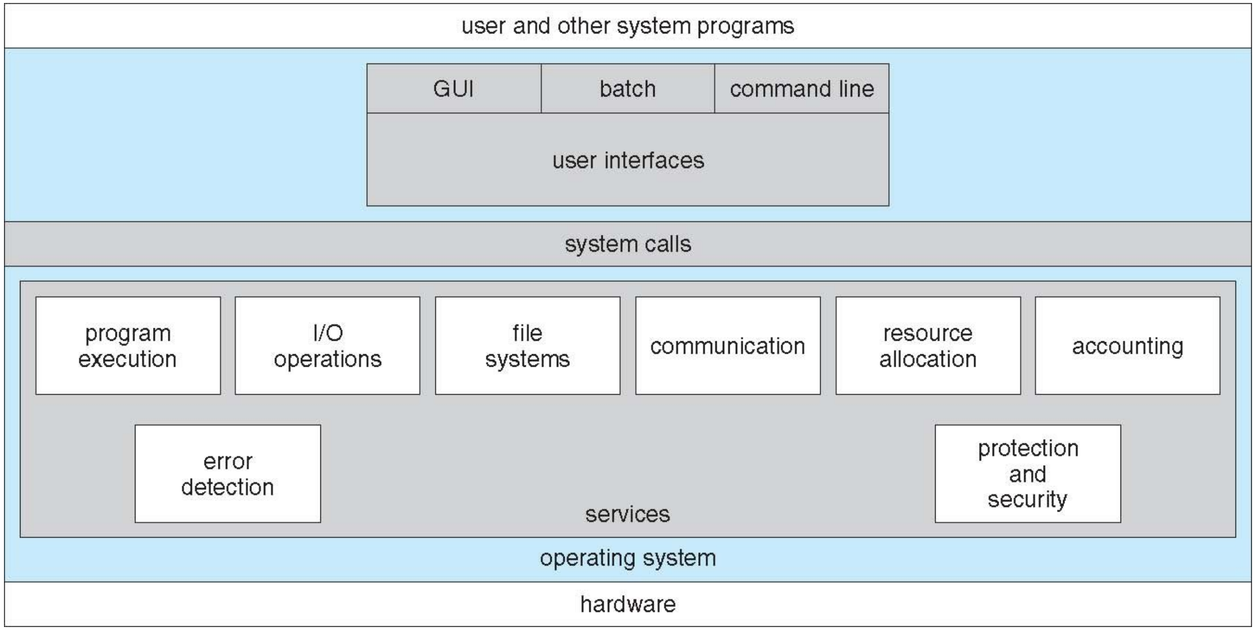


?

Applications
(excel, word, browser, ...)

Operating Systems

Hardware
(CPU, memory, hard drive)
“things you can kick”



user and other system programs

GUI | batch | command line

user interfaces

system calls

program execution | I/O operations | file systems | communication | resource allocation | accounting

error detection | protection and security

services

operating system

hardware

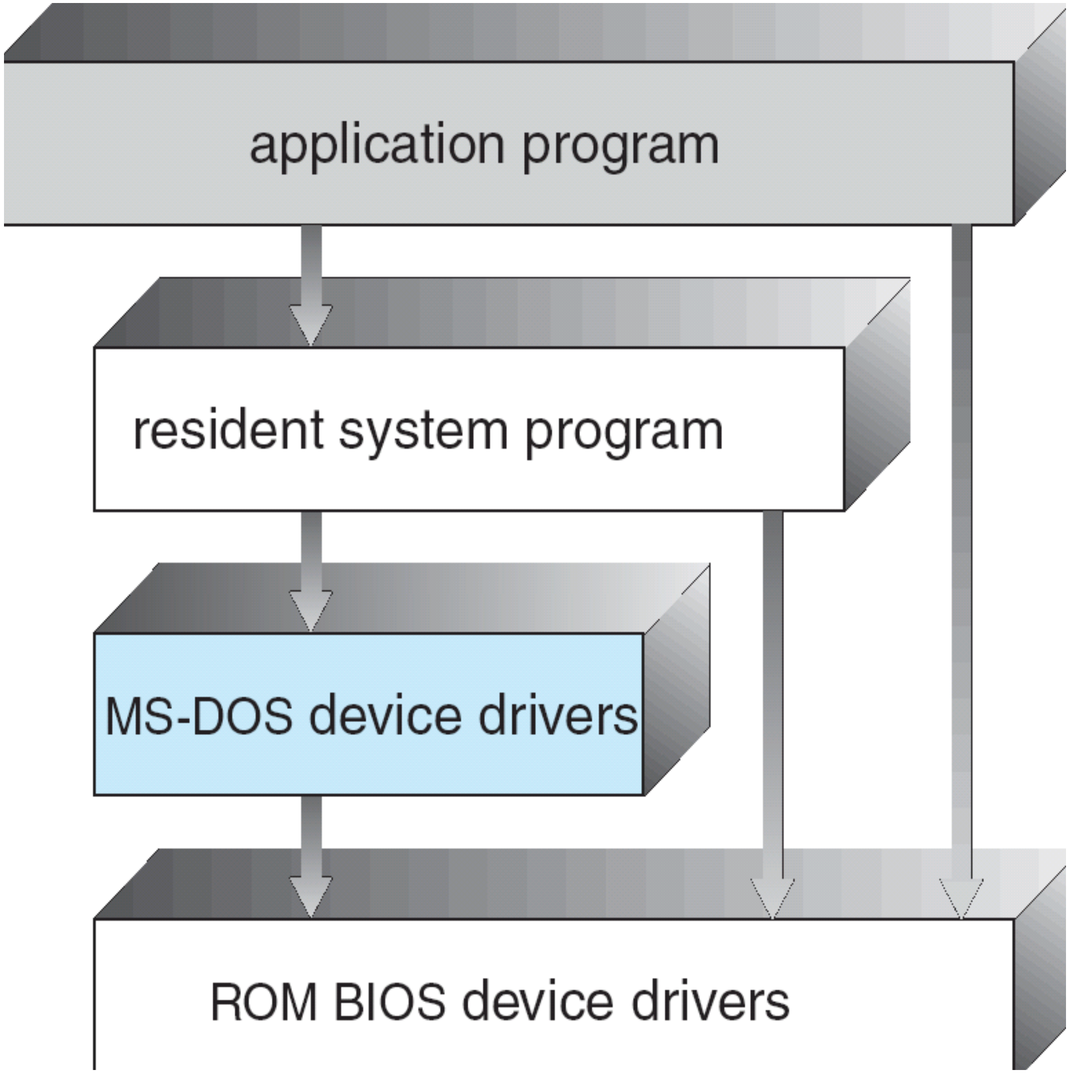
Interrupts, exceptions, and traps – OH MY

- Interrupts thus far: Device ↔ kernel
- Software-triggered events
  - Application state saved (as for interrupt) and can be resumed
  - Exceptions
    - Program faults (divide by zero, general protection fault, segmentation fault)
    - Not requested by executing application
  - Traps/Software Interrupts
    - Requested by application by executing specific instruction: `sysenter` or `int %d` on x86

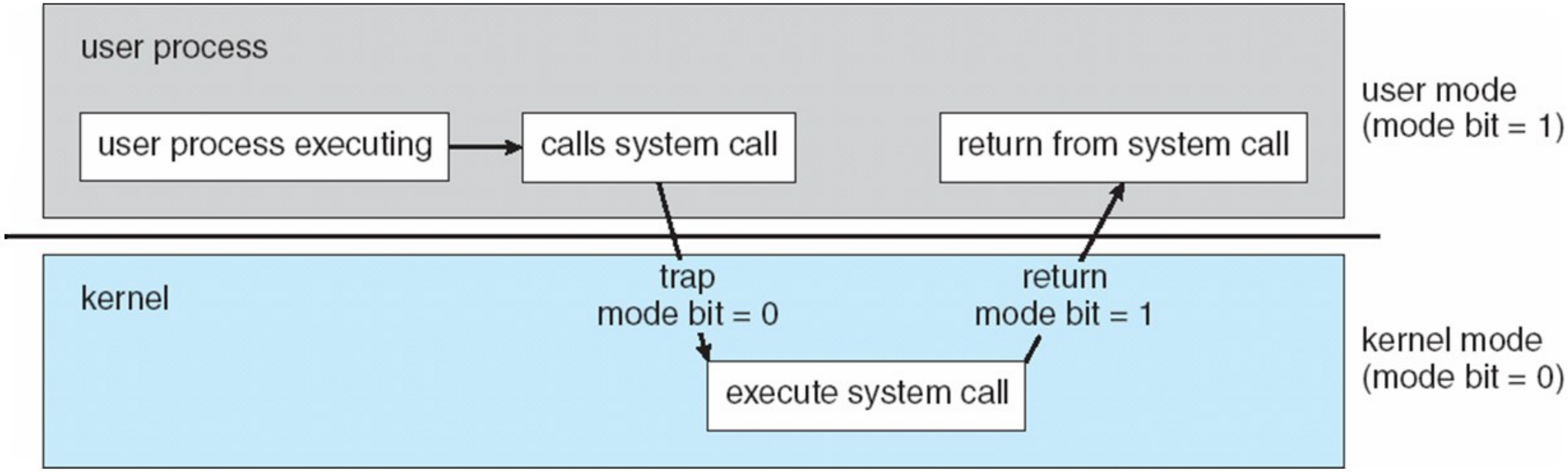
System Calls

- Wait, hardware support for calling the kernel?
  - Why can't I just call it directly (function call)?

MSDOS: No Structure/Protection



System Call w/ Dual-Mode HW



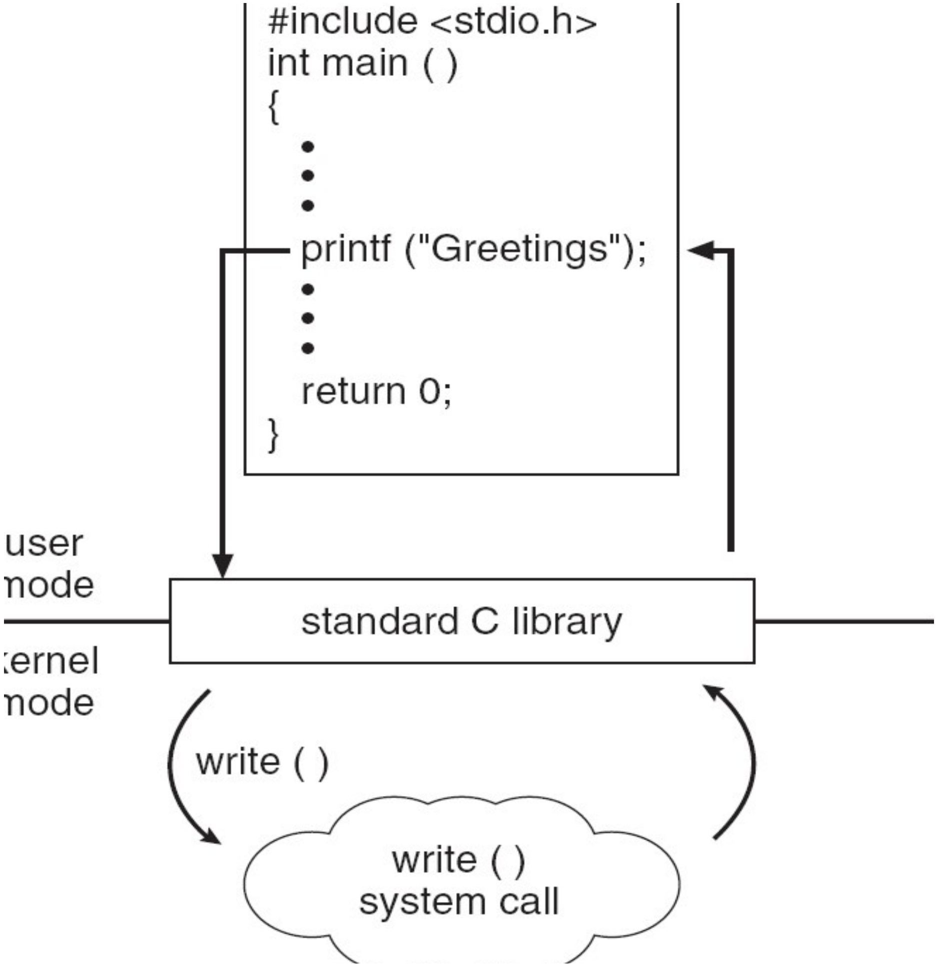
Timesharing systems: 1) protection applications from each other, *and* 2) **kernel from applications** (why the latter?)

- Mode bit == 0
  - Access kernel memory segments
  - Protected instructions
    - Access I/O: instructions to read/write to device control registers (in/out on x86)
    - Sensitive instructions
- *What happens to the registers, and stack?*

Syscall Mechanics

printf(“print me!”)

- write(1, “print me!”)
- put syscall number for write (4), file descriptor (1), and pointer to “print me!” into registers
- **sysenter**: mode bit = 0
  - Change to kernel stack
- Call address in syscall tbl at index 4
- Execute write system call
- **sysexit**: mode bit = 1
  - Restore application registers

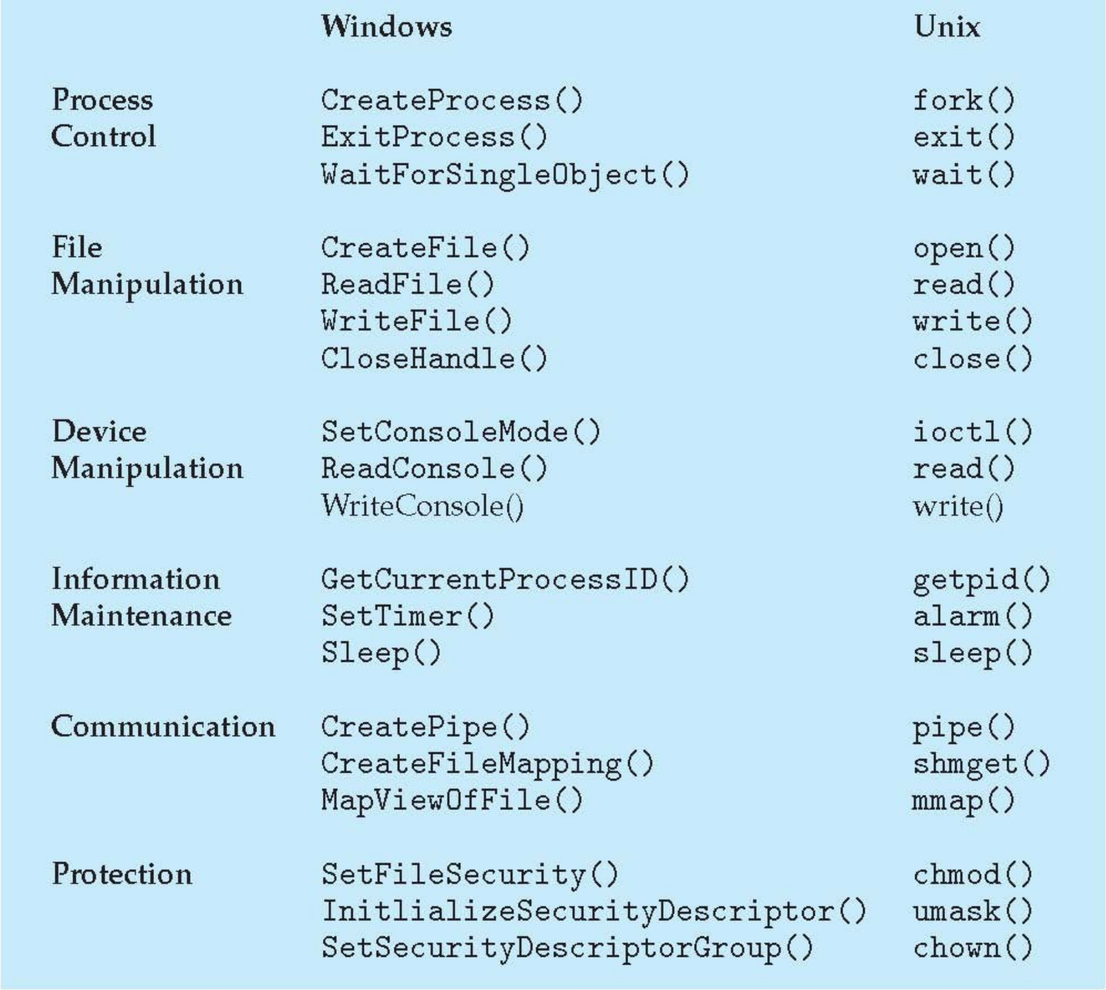


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

Abstraction for syscalls: APIs

- Application Programmer Interfaces (APIs)
  - Hide the details of how a syscall is carried out
  - POSIX (UNIX, Linux)
  - Win32 (Windows)
  - .Net (Windows XP and later)
  - Cocoa (OS X)

APIs (cont)



	Windows	Unix

Process Control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File Manipulation	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device Manipulation	`SetConsoleMode()`
	`ReadConsole()`
	WriteConsole()
		`ioctl()`
		`read()`
		write()

Information Maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communication	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shmget()`
		`mmap()`

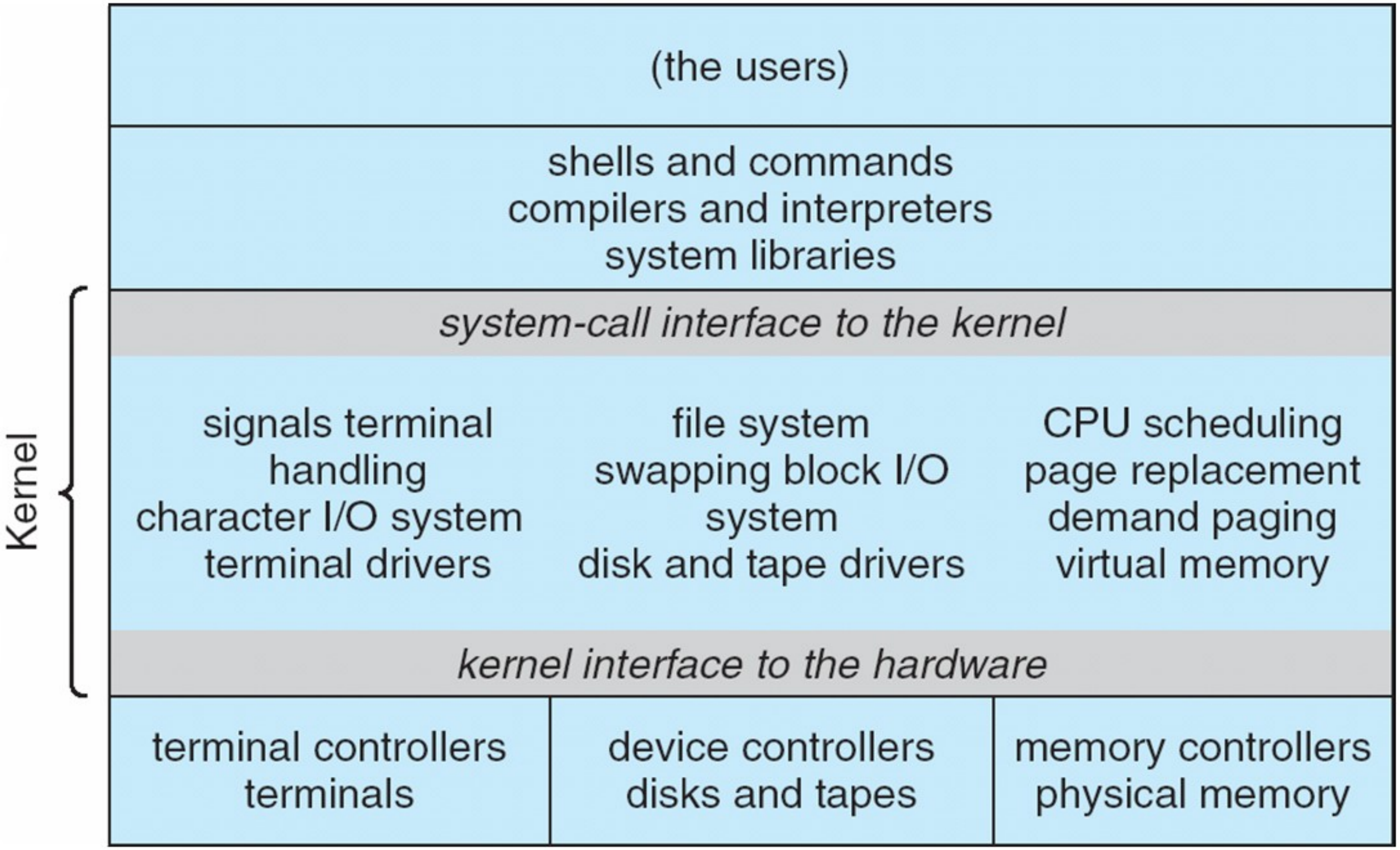
Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

Backup slides

Unix System Design

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel (mode bit = 0)
    - everything below the system-call interface and above the physical hardware
    - file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Unix System Structure



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal
handling
character I/O system
terminal drivers

file system
swapping block I/O
system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

device controllers
disks and tapes

memory controllers
physical memory

Microkernel System Structure

- Moves as much from the kernel into “*user*” space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

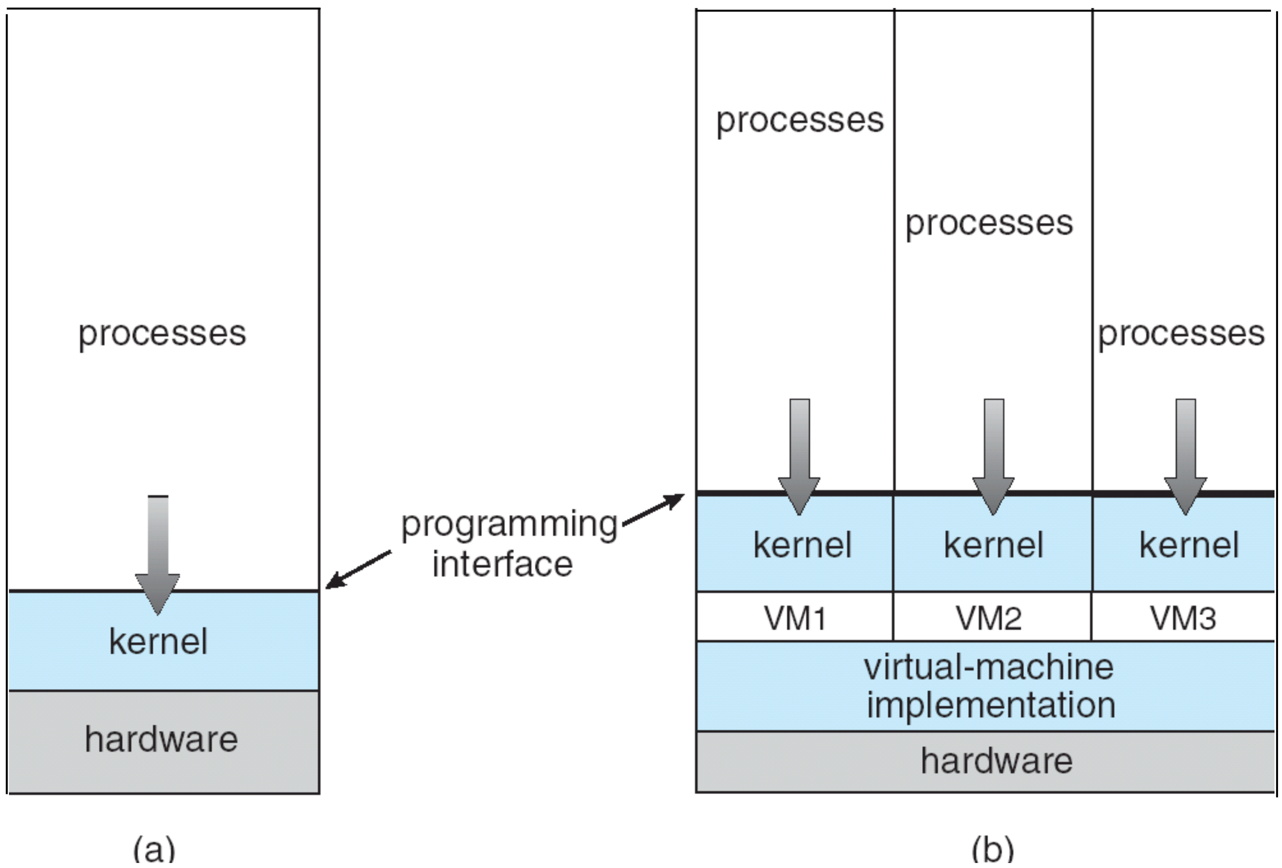
Virtual Machines

- Do you know what these are?
- What is the structure of VMs?

Virtual Machines (cont)

- Virtual machines treat hardware and the operating system kernel as though they were all hardware
- A virtual machine host (the kernel) provides an interface *identical* to the underlying bare hardware
- The operating system *host* creates the illusion that a process has its own processor and memory
- Each *guest* provided with a (virtual) copy of underlying computer
  - The API for virtual machines is a copy of the machine!

Virtual Machines (cont)



(a) Nonvirtual machine (b) virtual machine

Virtual Machine: Benefits

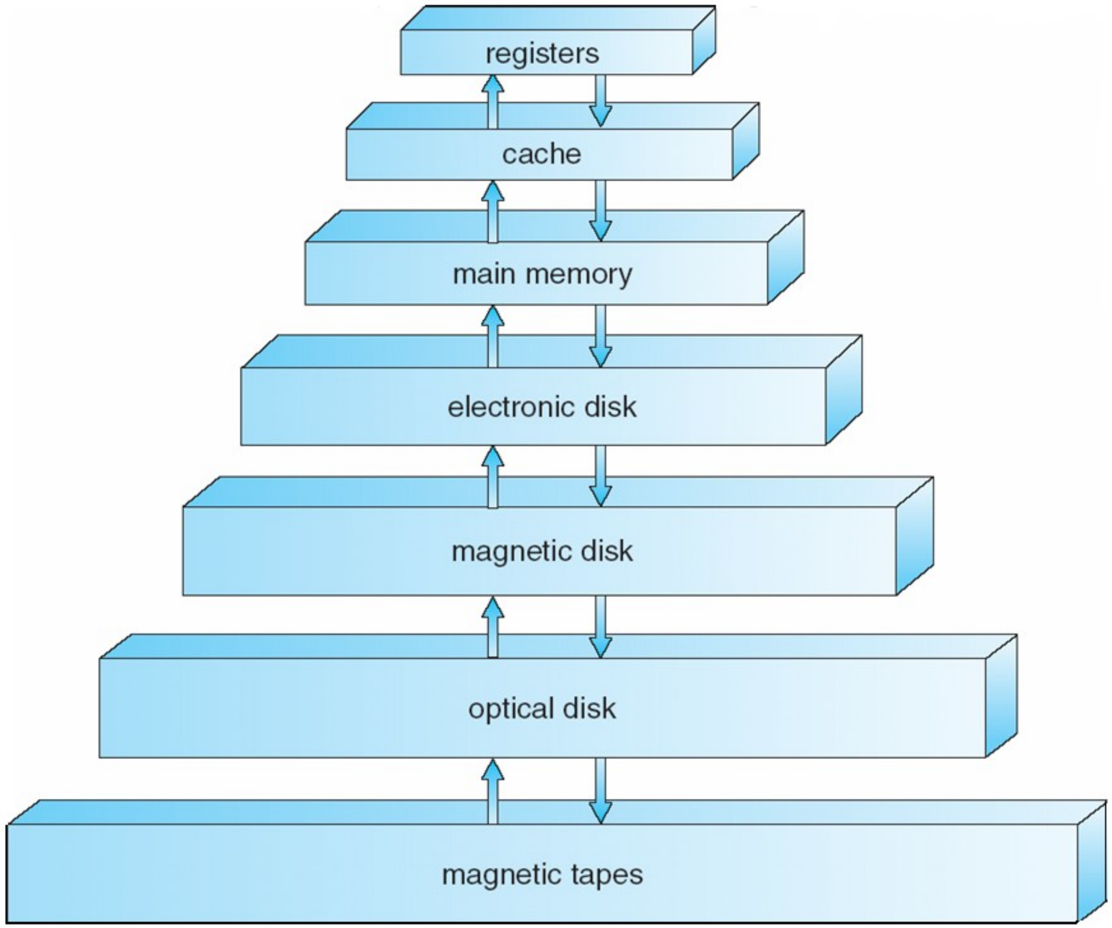
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted, controlled
- Communicate with each other, other physical systems via networking
- Useful for development, testing
- *Consolidation* of many low-resource use systems onto fewer busier systems

Followup

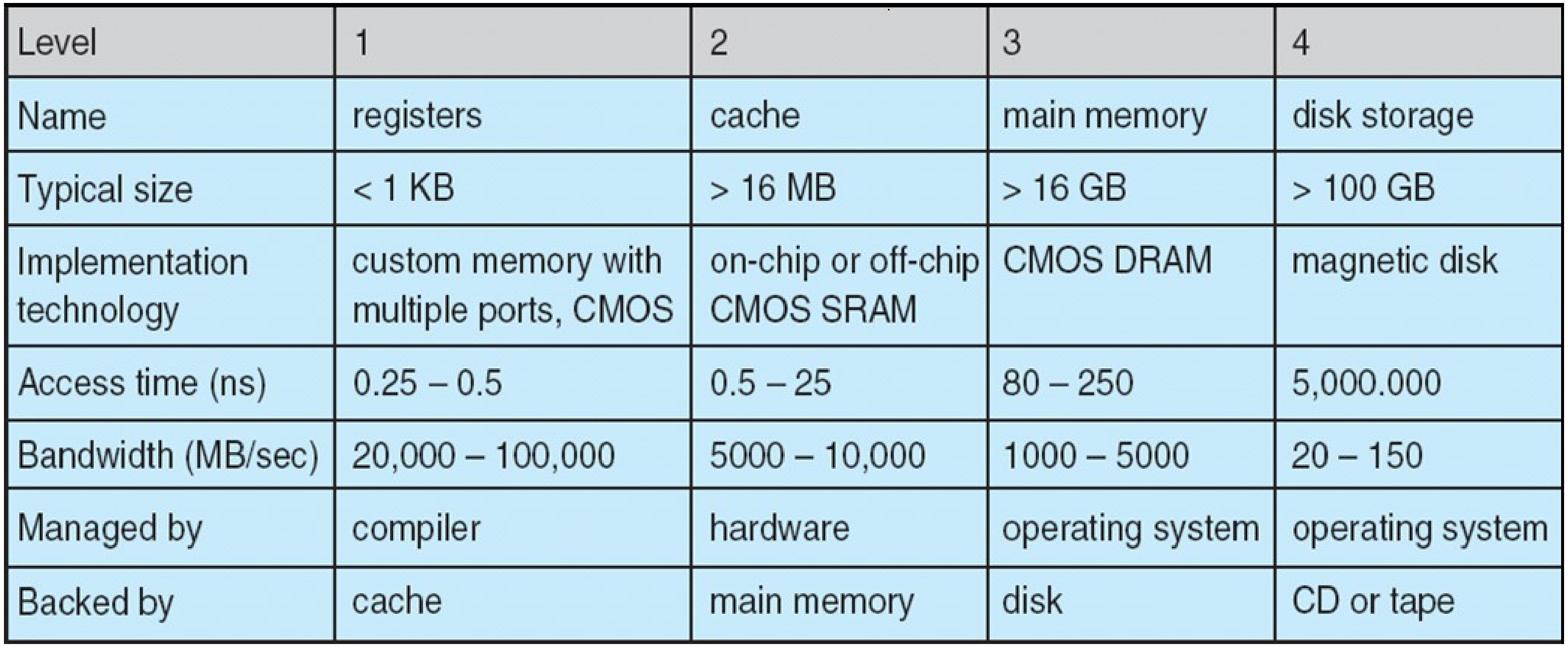
- Read chapters 1 and 2 in the book
  - Follow chapters on webpage
- Course updates, lecture slides (when available) available on course's webpage accessible from

  www.seas.gwu.edu/~gparmer/

Storage Hierarchy



Storage Hierarchy: Attributes



Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Goal:

- We want all accesses to be as fast as registers
- ...and also have the storage size of disk!

Caching

- Important principle, performed at many levels in a computer
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy