

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2025 Lecture 3



Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Today

- Multiprocessors
- OS Operations/Modes
- Storage hierarchy, caches
- OS Services
- Shells/User interfaces

Course Notes

- Follow updates and notes on Teams
- TA Office hour info on Teams
 - HW1 available this Thursday
 - Help Session for next week, perhaps Thursday
- iClicker Cloud
 - iClicker App must be registered and configured properly, otherwise the scores will not be uploaded in Canvas [Please check](#).
 - Exit poll: Identify ^{1 or 2} concepts you found most challenging or significant
 - Purpose of iClicker is to allow interaction, and get feedback, with automatic record keeping

Perspective

Differences among Subroutines/traps/Interrupt service routines

- Subroutines: program specifies transfer of control
- Traps: transfer of control to a system routine
- Interrupt: hardware request transfers control to the interrupt service routine

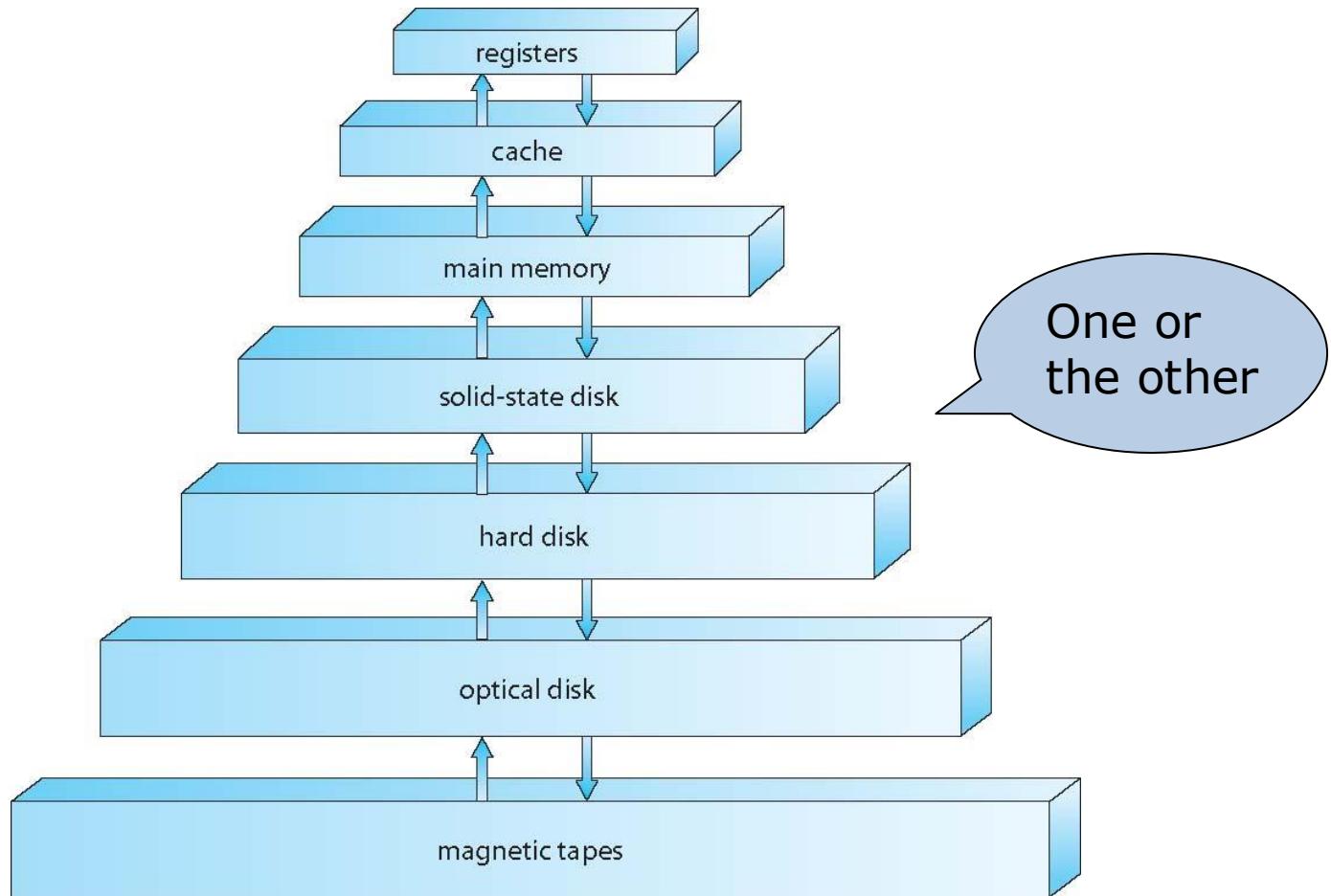
Interrupts: Why? How?

- *Interrupt request line* is hardware
- Interrupt causes transfer of control to *Interrupt Service Routine*
- Hence need to save *context*. Context restored when returning.

Direct Memory Access (DMA)

- for movement of a block of data
 - To/from disk, network etc.
- Requires **DMA controller unit**.
- Bypasses CPU to transfer data directly between I/O device and memory
- OS initiates a DMA transfer.
 - When done, interrupt is sent to the CPU to signal completion

Storage-Device Hierarchy



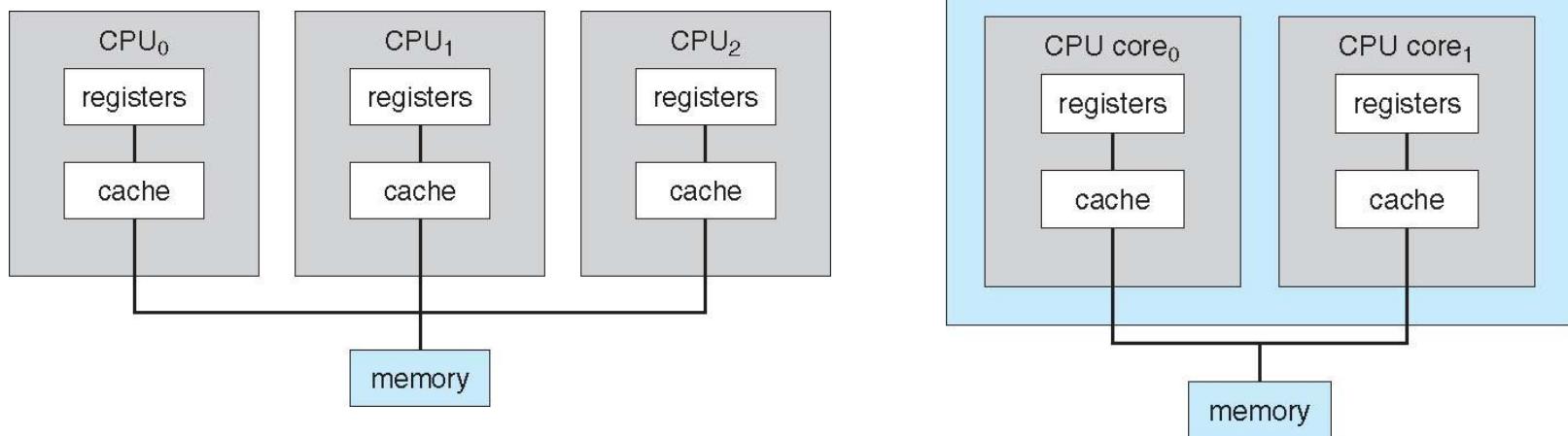
Multiprocessors

- Past systems used a single general-purpose processor
 - Most systems have special-purpose processors as well
- **Multiprocessor** systems were once special, now are common
 - Advantages include:
 1. Increased throughput
 2. Economy of scale
 - Two types:
 1. **Asymmetric Multiprocessing** – each processor is assigned a specific task. (older systems)
 2. **Symmetric Multiprocessing** – each processor performs all tasks

Multiprocessing Architecture

Multi-chip and multicore

- Multi-chip: Systems containing all chips
 - Chassis containing multiple separate systems
- Multi-core

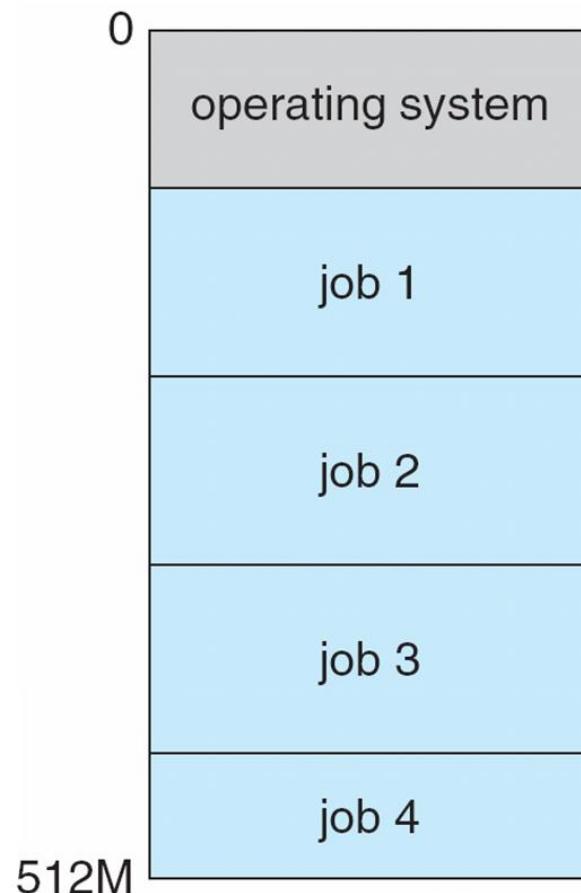


FAQ: How does system decide what information should be in cache?

Multiprogramming and multitasking

- **Multiprogramming** needed for efficiency
 - Single user cannot keep CPU and I/O devices busy at all times
 - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
 - A subset of total jobs in system is kept in memory
 - One job selected and run via **job scheduling**
 - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU **switches jobs so frequently that users can interact with each job while it is running**, creating **interactive** computing
 - **Response time** should be < 1 second
 - Each user has at least one program executing in memory \Rightarrow **process**
 - If several jobs ready to run at the same time \Rightarrow **CPU scheduling**
 - If processes don't fit in memory, **swapping** moves them in and out to run
 - **Virtual memory** allows execution of processes not completely in memory

Memory Layout for Multiprogrammed System



Operating-System Operations

- “**Interrupts**” (hardware and software)
 - Hardware interrupt by one of the devices
 - Software interrupt (**exception** or **trap**):
 - Software error (e.g., division by zero)
 - Request for operating system service
 - Other process problems like processes modifying each other or the operating system

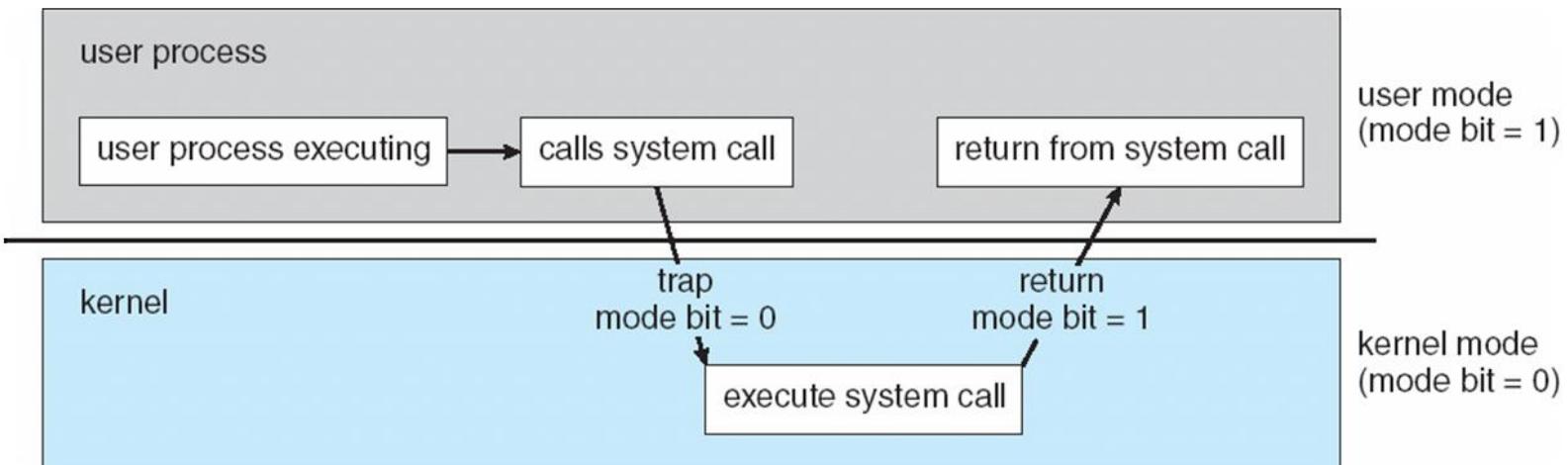
Operating-System Operations (cont.)

- **Dual-mode** operation allows OS to protect itself and other system components
 - **User mode** and **kernel mode**
 - **Mode bit** provided by hardware
 - Provides ability to distinguish when system is running user code or kernel code
 - Some instructions designated as **privileged**, only executable in kernel mode
 - System call changes mode to kernel, return from call resets it to user
- Some CPUs support multi-mode operations
 - i.e. **virtual machine manager (VMM)** mode for guest **VMs**

Also called Supervisor mode

Transition from User to Kernel Mode

- Ex: to prevent a process from hogging resources
 - Timer is set to interrupt the computer after some time period
 - Keep a counter that is decremented by the physical clock.
 - Operating system set the counter (privileged instruction)
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time
- Ex: System calls are executed in the kernel mode



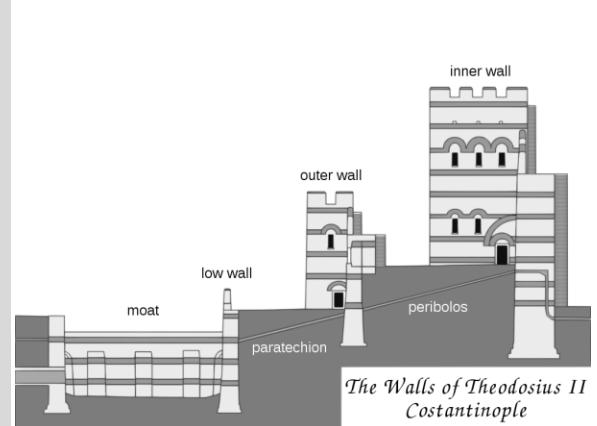
Multiple protection rings

Newer processors may offer multiple modes (“protection rings”)

- Ring -1 hypervisor VT-x, SVM
- **Ring 0 Kernel/supervisor**
- Rings 1,2 Device drivers
- **Ring 3 Applications**

To simplify discussions, we will consider **only two**. Linux and Windows uses only these two.

Note that labels/terminology may vary.



Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity***; process is an ***active entity***.
- Process needs resources to accomplish its task
 - CPU, memory, I/O, files
 - Initialization data
- Process termination requires reclaim of any reusable resources
- **Single-threaded process** has one **program counter** specifying location of next instruction to execute
 - Process executes instructions sequentially, one at a time, until completion
- **Multi-threaded process** has one program counter per thread
- Typically, system has many processes (some user, some operating system), running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes / threads



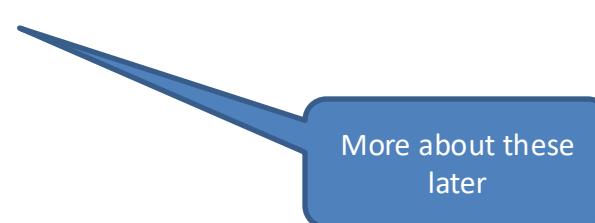
A program may involve multiple processes.

Our text uses terms **job** and **process** interchangeably.

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for
 - process synchronization
 - process communication
 - deadlock handling



More about these
later

Memory & Storage Management

K-scale: Amount of information/storage

Byte (B) = 8 bits (b)

Amount of info:

Kibibyte?

- A **kilobyte**, or **KB**, is 1,024 (or 2^{10}) bytes
- a **megabyte**, or **MB**, is $1,024^2$ (or 2^{20}) bytes
- a **gigabyte**, or **GB**, is $1,024^3$ bytes
- a **terabyte**, or **TB**, is $1,024^4$ bytes
- a **petabyte**, or **PB**, is $1,024^5$ bytes

Measures of time

- **Milliseconds**, **microseconds**, **nanoseconds**,
picoseconds: 10^{-3} , 10^{-6} , 10^{-9} , 10^{-12}

Performance of Various Levels of Storage

| Level | 1 | 2 | 3 | 4 | 5 |
|---------------------------|----------------------------------------|-------------------------------|------------------|------------------|------------------|
| Name | registers | cache | main memory | solid state disk | magnetic disk |
| Typical size | < 1 KB | < 16MB | < 64GB | < 1 TB | < 10 TB |
| Implementation technology | custom memory with multiple ports CMOS | on-chip or off-chip CMOS SRAM | CMOS SRAM | flash memory | magnetic disk |
| Access time (ns) | 0.25 - 0.5 | 0.5 - 25 | 80 - 250 | 25,000 - 50,000 | 5,000,000 |
| Bandwidth (MB/sec) | 20,000 - 100,000 | 5,000 - 10,000 | 1,000 - 5,000 | 500 | 20 - 150 |
| Managed by | compiler | hardware | operating system | operating system | operating system |
| Backed by | cache | main memory | disk | disk | disk or tape |

Movement between levels of storage hierarchy can be explicit or implicit

- Cache managed by hardware. Makes main memory appear much faster.
- Disks are several orders of magnitude slower than Main Memory.

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

ICQ



Terabytes, Megabytes

Q1. How many Megabytes are there in a Terabyte?

- A. $1\text{ TB} = 1000\text{ MB}$
- B. $1\text{ TB} = 10,000\text{ MB}$
- C. $1\text{ TB} = 1000,000\text{ MB}$
- D. Do we really need to know that?

Multiprogramming

- Q2.** You need multiple processors to allow multiprogramming.
- A. True, obviously.
 - B. Only if the processors are not powerful enough.
 - C. Not really.
 - D. Not sure.

Answers

Terabytes, Megabytes

Q1. How many Megabytes are there in a Terabyte?

- A. $1 \text{ TB} = 1000 \text{ MB}$
- B. $1 \text{ TB} = 10,000 \text{ MB}$
- C. $1 \text{ TB} = 1000,000 \text{ MB}$ $1 \text{ TB} = K \text{ GB}$, $1 \text{ GB} = K \text{ MB}$
- D. Do we really need to know that?

Multiprogramming

- Q2.** You need multiple processors to allow multiprogramming.
- A. True, obviously.
 - B. Only if the processors are not powerful enough.
 - C. Not really. Programs can time-share a processor
 - D. Not sure.

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Back from ICQ



General Concept: Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- 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
- Examples: “cache”, browser cache ..



Cache la Poudre?

Multilevel Caches

- **Cache**: between registers and main memory
 - Cache is faster and smaller than main memory
 - Makes main memory appear to be much faster, if the stuff is found in the cache much of the time
 - Hardware managed because of speed requirements
- Multilevel caches
 - L1: smallest and fastest of the three (about 4 cycles, 32 KB)
 - L2: bigger and slower than L1 (about 10 cycles, 256KB)
 - L3: bigger and slower than L2 (about 50 cycles, 8MB)
 - Main memory: bigger and slower than L3 (about 150 cycles, 8GB)
- You can mathematically show that multi-level caches improve performance with usual high hit rates.

means Main
Memory here

Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and when
 - Optimizing CPU utilization and computer response to users
- Memory management activities
 - Keeping track of which parts of memory are currently being used and by whom
 - Deciding which processes (or parts thereof) and data to move into and out of memory
 - Allocating and deallocating memory space as needed

CPU
scheduling

Storage Management

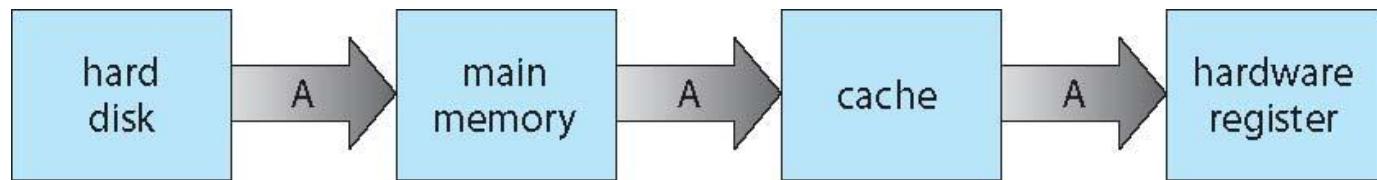
- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - **file**
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - OS activities include
 - Creating and deleting files and directories
 - Primitives to manipulate files and directories
 - Mapping files onto secondary storage
 - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually, disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
 - Free-space management
 - Storage allocation
 - Disk scheduling (for magnetic disks)
- Some storage need not be fast
 - Tertiary storage includes optical storage, magnetic tape
 - Still must be managed – by OS or applications
 - Varies between WORM (write-once, read-many-times) and RW (read-write)

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
 - Several copies of a datum can exist
 - Various solutions covered in Chapter 19 (*will not get to it*)

- Intro Pt 2
- *End of time*

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

OS Structures



Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Chap2: Operating-System Structures

Objectives:

- Services OS provides to users, processes, and other systems
- Structuring an operating system
- How operating systems are designed and customized and how they boot

OS Services for the User 1/3

- Operating systems provide an environment for execution of programs and services to programs and users
 - **User interface** - Almost all operating systems have a user interface (**UI**).
 - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**
 - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device

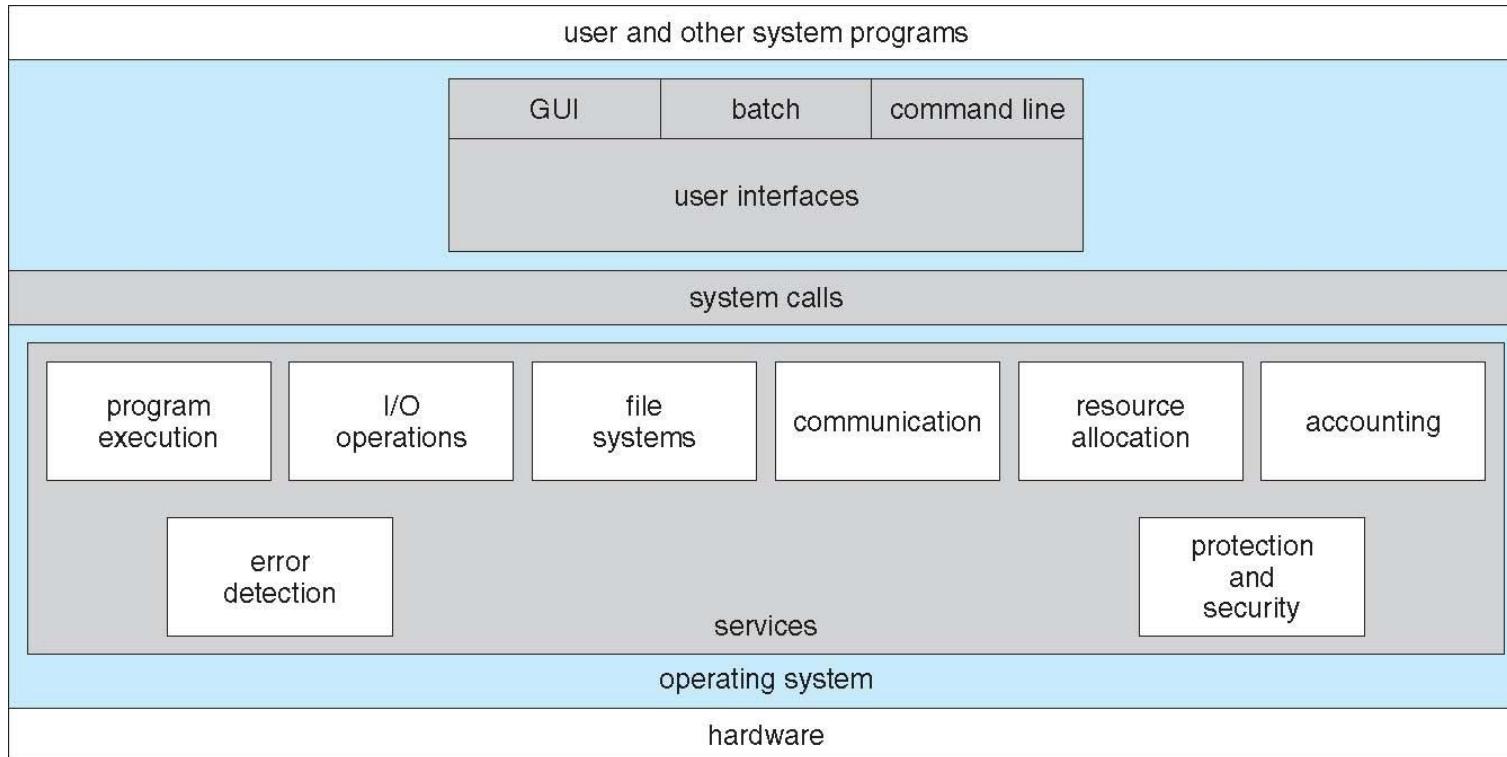
OS services for the User 2/3 (Cont.)

- **File-system operations** - read and write files and directories, create and delete them, search them, list file Information, permission management.
- **Communications** – Processes may exchange information, on the same computer or between computers over a network
 - via shared memory or through message passing (packets moved by the OS)
- **Error detection** – OS needs to be constantly aware of possible errors
 - May occur in the CPU and memory hardware, in I/O devices, in user program
 - For each type of error, OS should take the appropriate action to ensure correct and consistent computing

OS services for system 3/3 (Cont.)

- OS functions for ensuring the efficient resource sharing
 - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
 - **Accounting** - To keep track of which users use how much and what kinds of computer resources
 - **Protection and security** - concurrent processes should not interfere with each other
 - **Protection** involves ensuring that all access to system resources is controlled
 - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

A View of Operating System Services



User interfaces

Let us see

- CLI: command line interface
- GUI: graphical user interface

CLI or **command interpreter** allows direct command entry

- Fetches a command from user and executes it
- Sometimes implemented in kernel, sometimes by systems programs
- Sometimes commands built-in, sometimes just names of programs
 - If the latter, adding new features doesn't require shell modification
- Multiple flavors implemented – **shells**

Ex:

Windows: command prompt

Linux: bash

Shell Command Interpreter

A bash session

```
>Last login: Sat Aug 27 22:09:08 on ttys000
|Ys-MacBook-Air:~ ymalaiya$ echo $0
|  -bash
|Ys-MacBook-Air:~ ymalaiya$ pwd
| /Users/ymalaiya
|Ys-MacBook-Air:~ ymalaiya$ ls
| 270          Desktop        Downloads        Music        android-sdks
| Applications  Dialcom        Library        Pictures
| DLID Books    Documents      Movies        Public
|Ys-MacBook-Air:~ ymalaiya$ w
| 22:14  up  1:12, 2 users, load averages: 1.15 1.25 1.27
| USER   TTY   FROM           LOGIN@  IDLE WHAT
| ymalaiya  console -          21:02   1:11 -
| ymalaiya  s000   -          22:14       - w
|Ys-MacBook-Air:~ ymalaiya$ ps
|   PID  TTY      TIME CMD
|     594 ttys000  0:00.02 -bash
|Ys-MacBook-Air:~ ymalaiya$ iostat 5
|           disk0      cpu      load average
|             KB/t  tps  MB/s  us  sy  id  1m   5m   15m
|             36.76  17   0.60   5   3  92  1.42  1.31  1.28
|^C
|Ys-MacBook-Air:~ ymalaiya$ ping colostate.edu
|PING colostate.edu (129.82.103.93): 56 data bytes
| 64 bytes from 129.82.103.93: icmp_seq=0 ttl=116 time=46.069 ms
| 64 bytes from 129.82.103.93: icmp_seq=1 ttl=116 time=41.327 ms
| 64 bytes from 129.82.103.93: icmp_seq=2 ttl=116 time=58.673 ms
| 64 bytes from 129.82.103.93: icmp_seq=3 ttl=116 time=44.750 ms
| 64 bytes from 129.82.103.93: icmp_seq=4 ttl=116 time=48.336 ms
|^C
|--- colostate.edu ping statistics ---
| 5 packets transmitted, 5 packets received, 0.0% packet loss
| round-trip min/avg/max/stddev = 41.327/47.831/58.673/5.877 ms
|Ys-MacBook-Air:~ ymalaiya$
```

Common bash commands 1/2

| pwd | print Working directory |
|------------------|--------------------------------------------------------|
| ls -l | Files in the working dir –long format |
| cd dirpath | Change to dirpath dir |
| . .. ~username / | This dir , upper , username's home, root |
| cp f1 d1 | Copy f1 to dir d1 |
| mv f1 d1 | Move f1 to d1 |
| rm f1 f2 | Remove f1, f2 |
| mkdir d1 | Create directory d1 |
| which x1 | Path for executable file x1 |
| man cm help cm | Manual entry or help with command cm |
| ls > f.txt | Redirect command std output to f.txt, >> to append |
| sort < list.txt | Std input from file |
| ls -l less | Pipe first command into second |

Common bash commands 2/2

| echo \$((expression)) | Evaluate expression |
|-----------------------------|------------------------------------------------------|
| echo \$PATH | Show PATH |
| echo \$SHELL | Show default shell |
| chmod 755 dir | Change dir permissions to 755 |
| jobs ps | List jobs for current shell, processes in the system |
| kill id | Kill job or process with given id |
| cmd & | Start job in background |
| fg id | Bring job id to foreground |
| ctrl-z followed by bg or fg | Suspend job and put it in background |
| w who | Who is logged on |
| ping ipadd | Get a ping from ipadd |
| ssh user@host | Connect to host as user |
| grep pattern files | Search for pattern in files |
| Ctrl-c | Halt current command |

User Operating System Interface - GUI

- User-friendly **desktop** metaphor interface
 - Usually mouse, keyboard, and monitor
 - **Icons** represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
 - Invented at Xerox PARC in 1973
- Most systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI “command” shell
 - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
 - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Touchscreen Interfaces

- Touchscreen devices require new interfaces
 - Mouse not possible or not desired
 - Actions and selection based on gestures
 - Virtual keyboard for text entry
- Voice commands.



The Mac OS X GUI



System Calls

- What are they?
 - Calls to system routines
- How are they implemented?

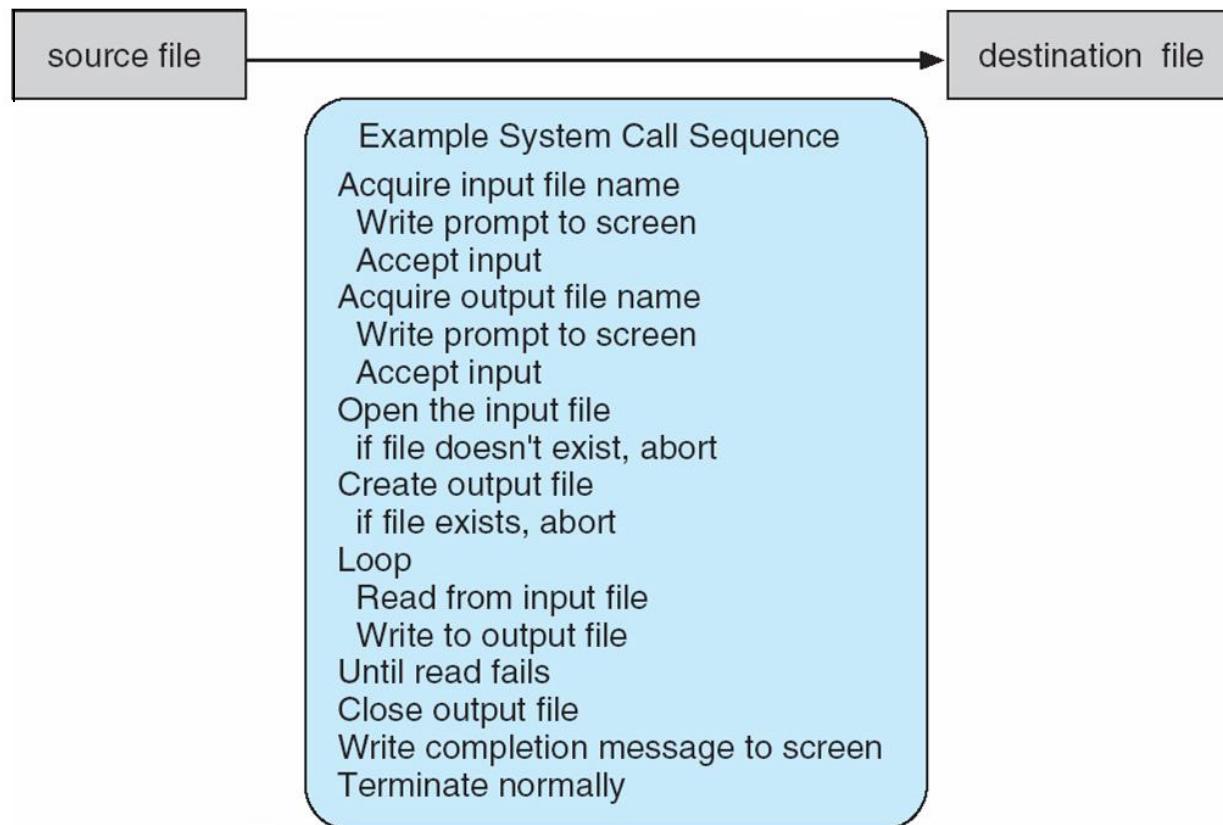
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, **POSIX API for POSIX-based systems** (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout our text are generic.

Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t      read(int fd, void *buf, size_t count)
```

return function parameters
value name

unistd.h header file provides
access to the POSIX API

[read\(2\) — Linux manual page](#)

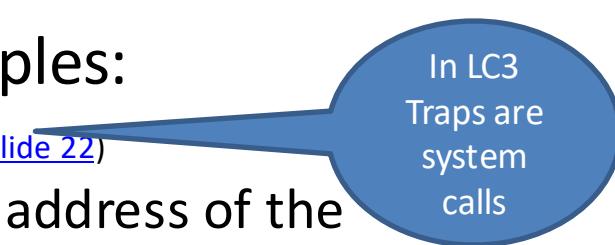
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

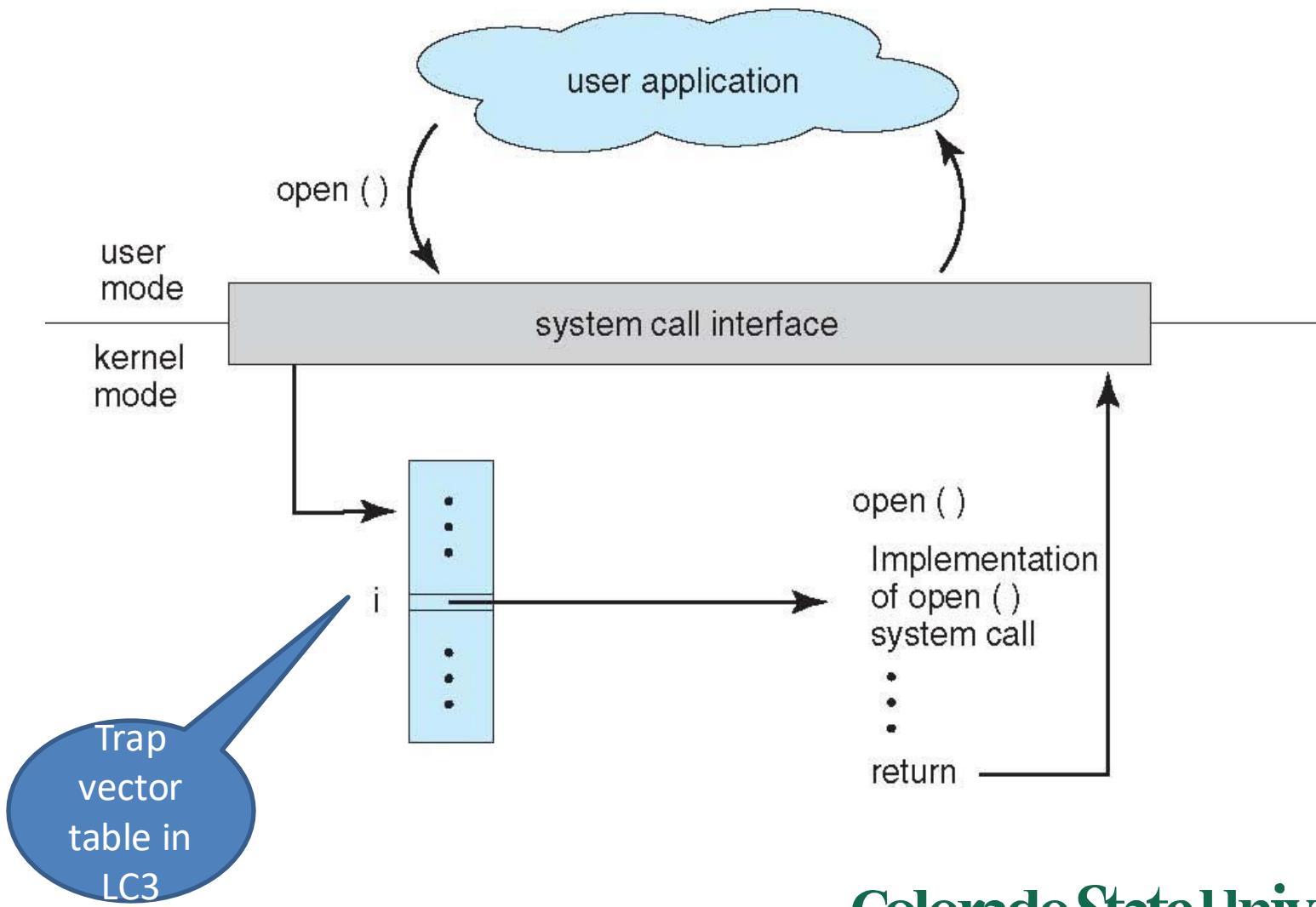
System Call Implementation

- The caller **need know nothing** about how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result call
 - Most details of OS interface hidden from programmer by API
 - Managed by run-time support library (set of functions built into libraries included with compiler)
- System call implementation examples:
 - LC-3 Trap x21 (**OUT**) code in Patt & Patel ([see slide 22](#))
 - Identified by a number that leads to address of the routine
 - Arguments provided in designated registers
 - [Linux x86_64 table](#), [code snippets](#)



In LC3
Traps are
system
calls

API – System Call – OS Relationship

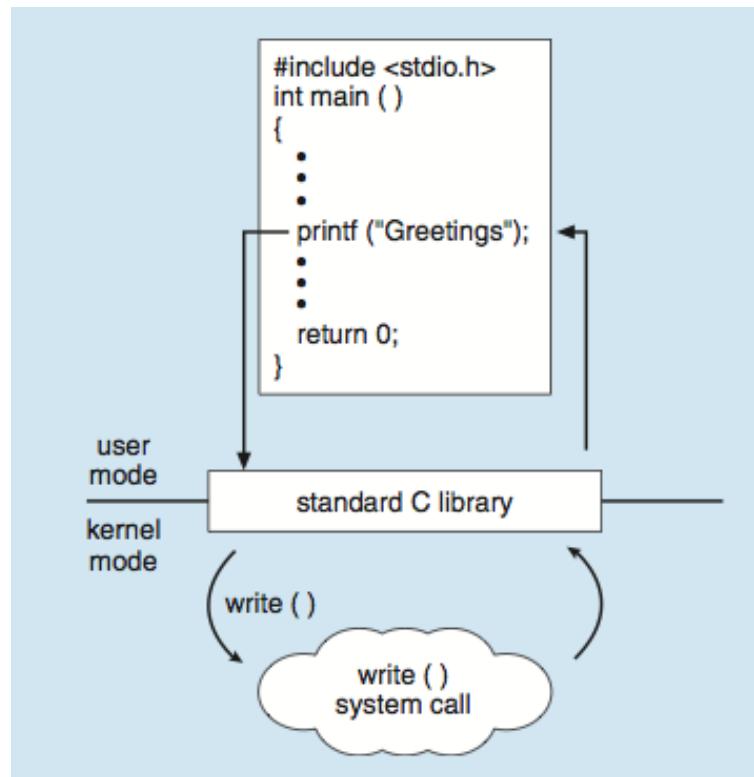


Examples of Windows and Unix System Calls

| | 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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup() | chmod() umask() chown() |

Standard C Library Example

- C program invoking *printf()* *library call*, which calls *write()* *system call*

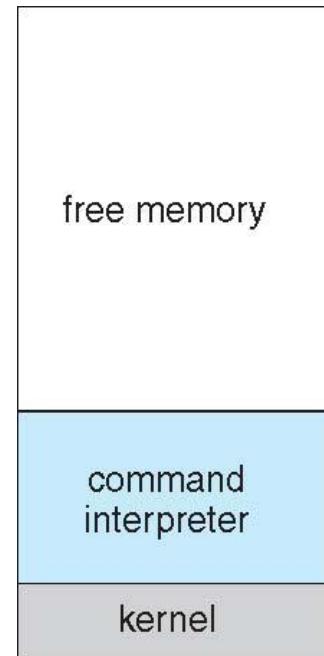


POSIX

- POSIX: Portable Operating Systems Interface for UNIX for system commands *Pronounced pahz-icks*
- **POSIX.1** published in 1988
- Final POSIX standard: Joint document
 - Approved by IEEE & Open Group End of 2001
 - ISO/IEC approved it in November 2002
 - Most recent *IEEE Std 1003.1-2017 Edition*
- Most OSs are *mostly POSIX-compliant*
- We will use a few POSIX-compliant system commands

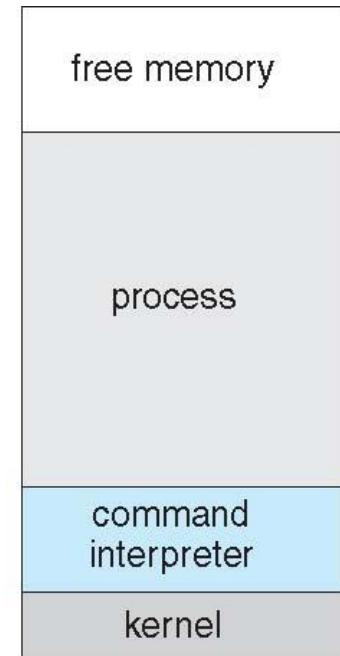
Example OS: MS-DOS '81..

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
 - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



(a)

At system startup



(b)

running a program

Example: xBSD

'93 Berkely

- Unix '73 variant, inherited by several later OSs
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
 - Executes exec() to load program into process
 - Shell waits for process to terminate or continues with user commands
- Process exits with:
 - code = 0 – no error
 - code > 0 – error code

