

# CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 5



## Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

# User Operating System Interface - CLI

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

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

Ex:

Windows: command

Linux: bash

# Shell Command Interpreter

```
ymalaiya — -bash — 81x35
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$
```

# FAQ

- Batch processing: Executing a series of non-interactive jobs
- Why magnetic tapes are still used?
- Shell, Kernel, Terminal
- Buffering for IO vs spooling for printing
- Process vs Thread (more soon)
- Is cache in “Processor” or Memory?
- Cache coherency: how? Coherency when multiple cores share a cache? (advanced, cs470)
- IO subsystem: part of kernel interacts with hardware controller.

# Common bash commands 1/2

<code>pwd</code>	Working directory	
<code>ls -l</code>	Files in the working dir –ling format	
<code>cd dirpath</code>	Change to dirpath dir	
<code>. .. ~username /</code>	This, upper, username's home, root	
<code>cp f1 d1</code>	Copy f1 to dir d1	
<code>mv f1 d1</code>	Move f1 to d1	
<code>rm f1 f2</code>	Remove f1, f2	
<code>mkdir d1</code>	Create directory d1	
<code>which x1</code>	Path for executable file x1	
<code>man cm    help cm</code>	Manual entry or help with command cm	
<code>ls &gt; f.txt</code>	Redirect command std, output to f.txt, >> to append	
<code>sort &lt; list.txt</code>	Std input from file	
<code>ls -l   less</code>	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		
jobs <b>ps</b>	List jobs or processes	
kill id	Kill job or process with 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 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
  - Invented at Xerox PARC
- 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



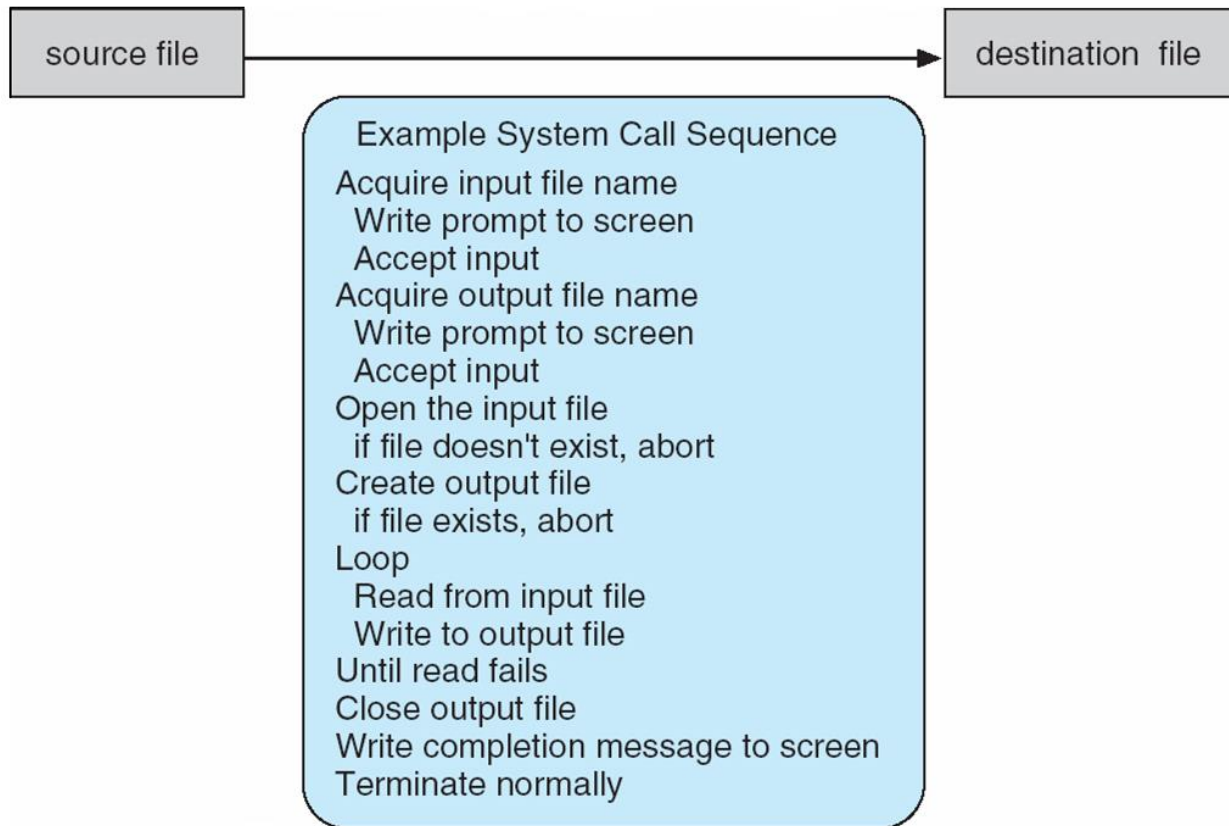
# System Calls

- 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 one CLI command->many system calls



# 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:

<pre>#include &lt;unistd.h&gt;</pre>		
<pre>ssize_t</pre>	<pre>read(int fd, void *buf, size_t count)</pre>	
return	function	parameters
value	name	

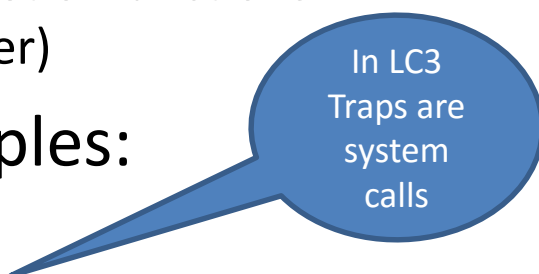
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `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`.

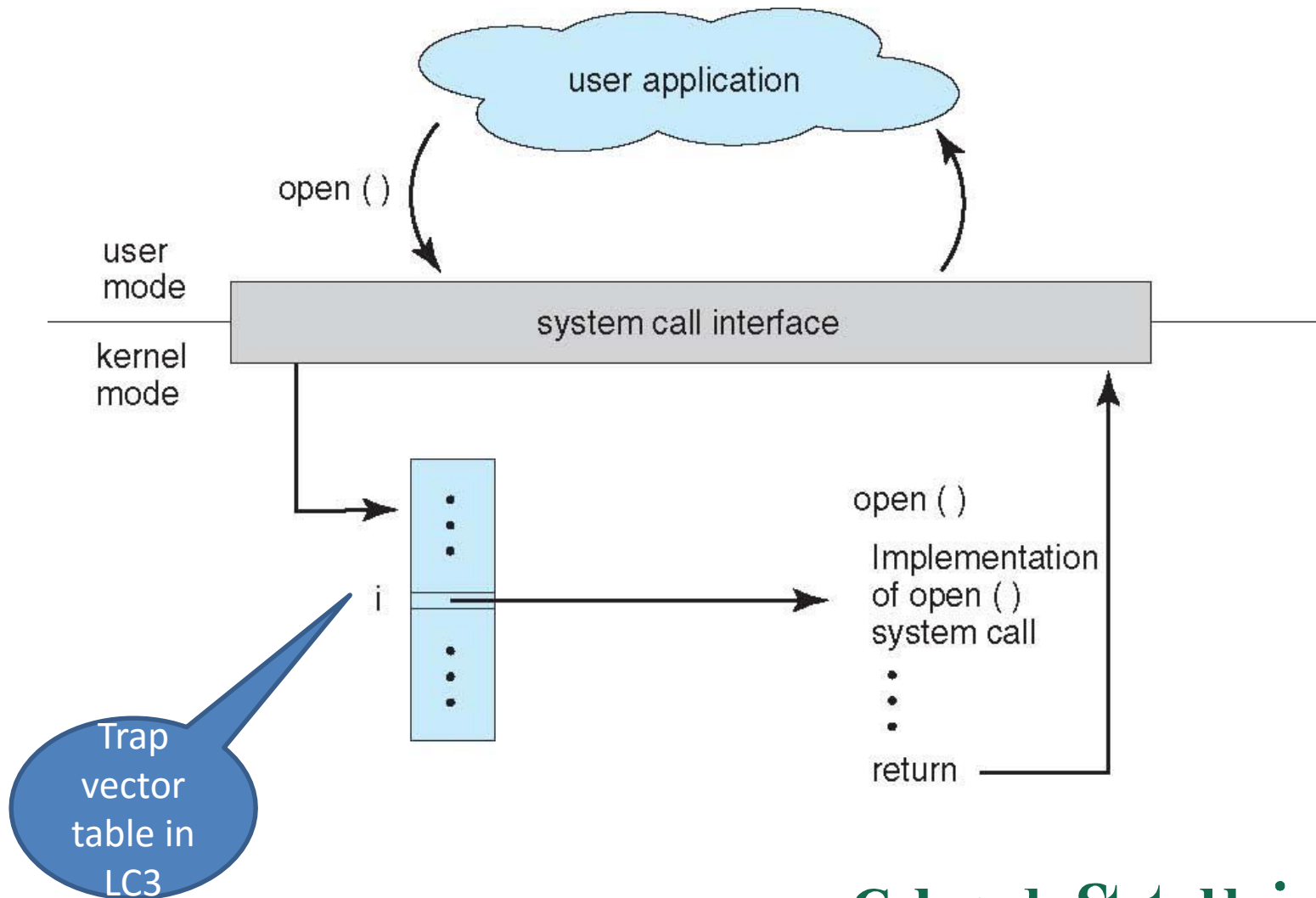
# System Call Implementation

- 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 x23 (IN)** code in p. 208 Patt & Patel



In LC3  
Traps are  
system  
calls

# API – System Call – OS Relationship



# Types of System Calls 1/3

- Process control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- Debugger for determining bugs, single step execution
- Locks for managing access to shared data between processes

# Types of System Calls 2/3

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>• File management<ul style="list-style-type: none"><li>• create file, delete file</li><li>• open, close file</li><li>• read, write, reposition</li><li>• get and set file attributes</li></ul></li></ul> | <ul style="list-style-type: none"><li>• Device management<ul style="list-style-type: none"><li>• request device, release device</li><li>• read, write, reposition</li><li>• get device attributes, set device attributes</li><li>• logically attach or detach devices</li></ul></li></ul> |
|--|---|
- Information maintenance
    - get time or date, set time or date
    - get system data, set system data
    - get and set process, file, or device attributes



# Types of System Calls 3/3

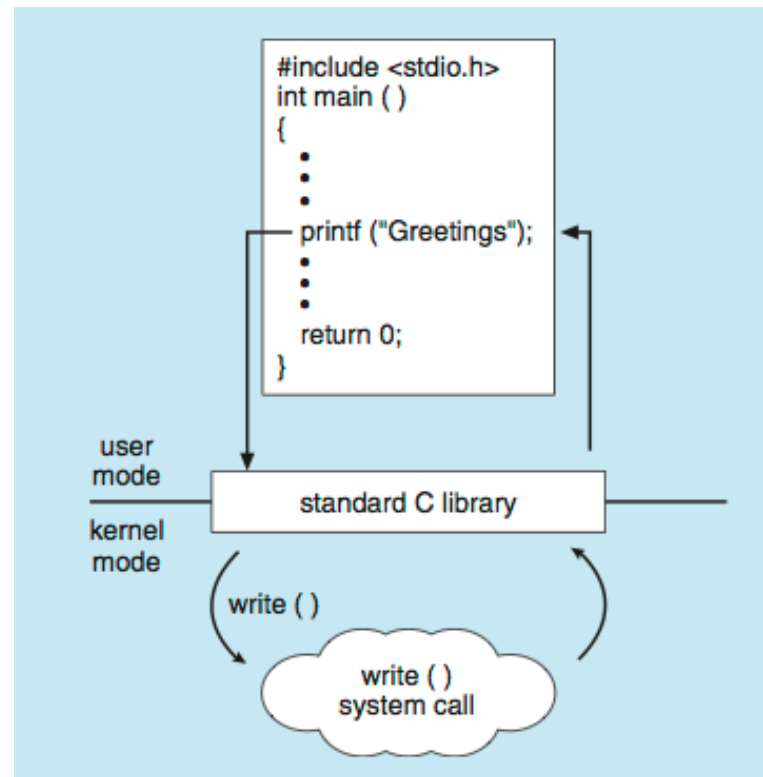
- **Communications**
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices
- **Protection**
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

# 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()

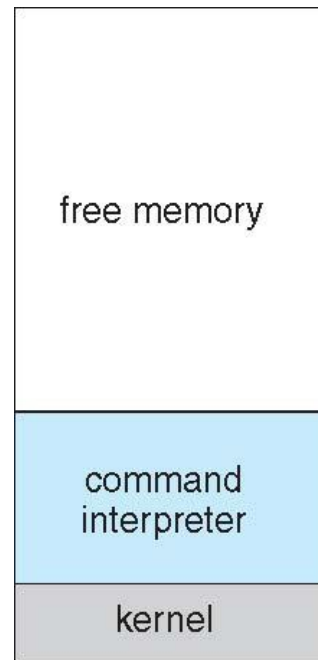
# Standard C Library Example

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



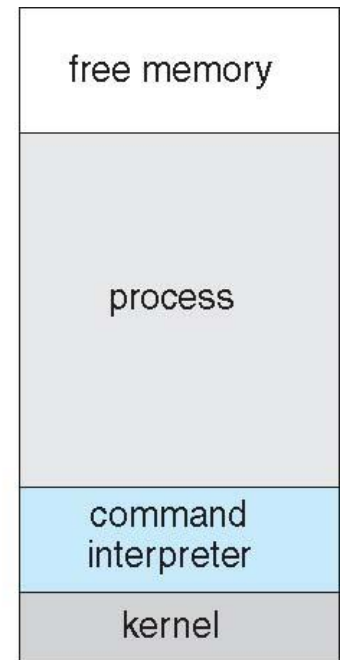
# Example: MS-DOS

- 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

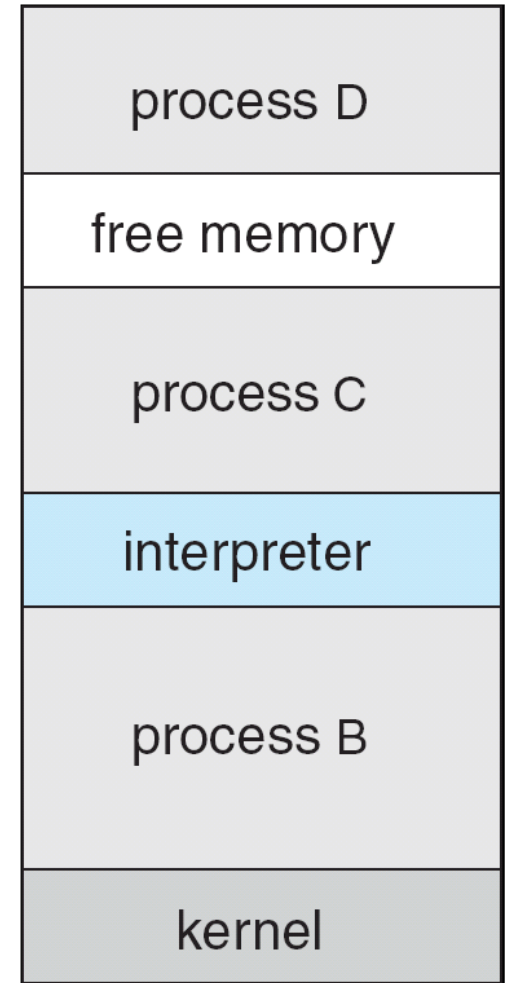


(b)

running a program

# Example: FreeBSD

- Unix variant
- 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



# System Programs 1/4

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

# System Programs 2/4

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry** - used to store and retrieve configuration information

# System Programs 3/4

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another



# System Programs 4/4

- **Background Services**

- Launch at boot time
  - Some for system startup, then terminate
  - Some from system boot to shutdown
- Provide facilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as **services**, **subsystems**, **daemons**

- **Application programs**

- Don't pertain to system
- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke

# Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel -Mach