CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Spring 21 Lecture 4

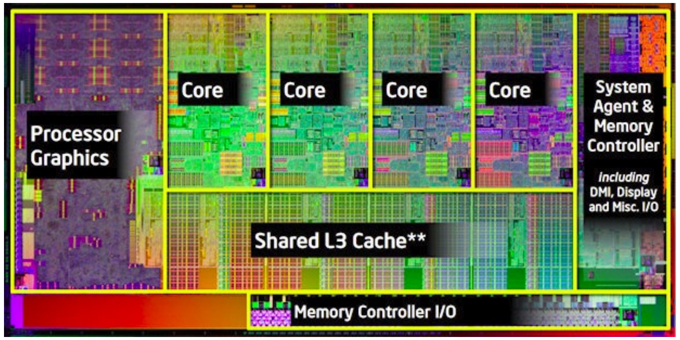
OS Structures/Processes

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

FAQ

- A **chip** can have one or more **processors** (**CPU, core**) and possibly more components.



- **Kernel vs OS**: Kernel: process/memory/file/IO management, OS can include UI, libraries etc.
- **Why User vs kernel mode**? Because users can't be trusted.
- Where are **registers**, **Cache** and **main memory** , physically?
- Trap routines, interrupts
- Good that you are thinking about these.
  - Threads vs processes, Scheduling, Memory management: we will study in detail
  - Multi-level caches: how are they implemented?

Note: TA office hours are available. Help session Slides: Schedule, Video: Teams

Colorado State University

K-scale: Amount of information/storage

Byte (B) = 8 bits (b)

Kibibyte?

Amount of info:

- 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

- **Milli**seconds, **micro**seconds, **nano**seconds, **pico**seconds 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}

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

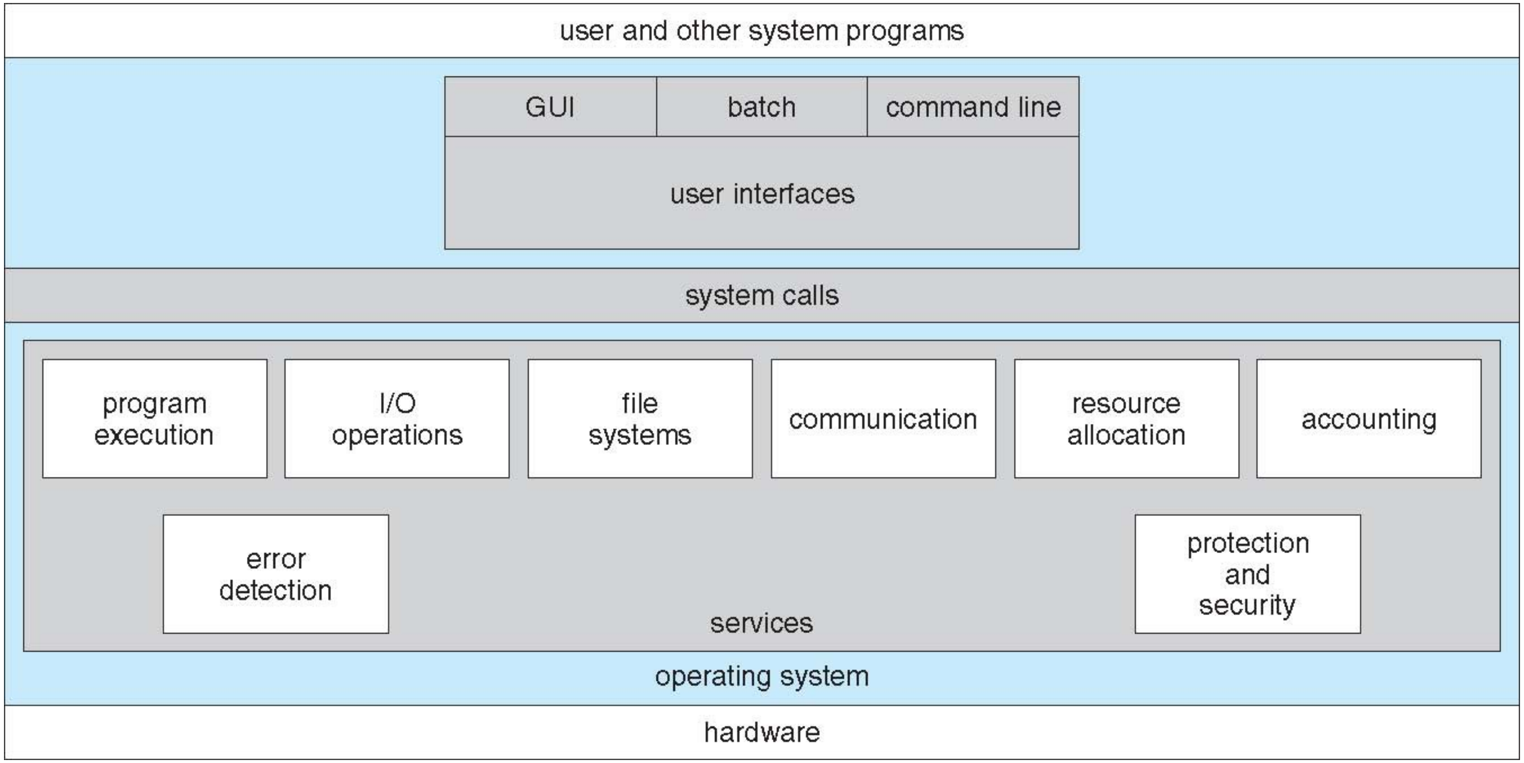
OS Services

- **User interface** -
  - Varies between **Command-Line** **(CLI)**, **Graphics User Interface** **(GUI)**, **Batch**
- **Program execution**
- **I/O operations**
- **File-system operations**
- **Communications**
- **Error detection**
- **Resource allocation**
- **Accounting**
- **Protection and security**

MAC: look at processes
Activity Monitor > CPU
Process, threads, PID etc.
info about a process

Colorado State University

A View of Operating System Services



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 prompt
Linux: bash

Shell Command Interpreter

A bash session

```
ymalaiya — -bash — 81×35
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, usename’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	
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 (shows as ^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

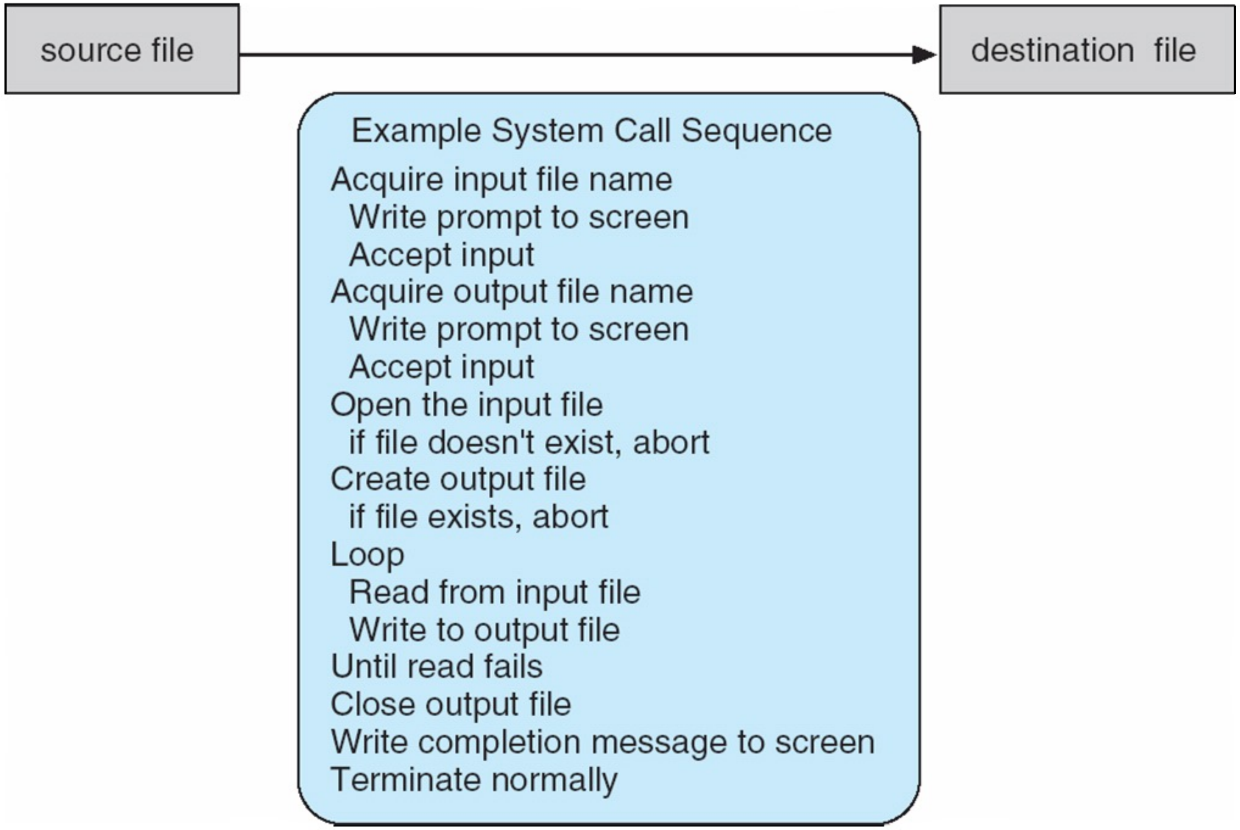
System Calls

- 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 value	function name	parameters
`ssize_t`	`read`	`(int fd, void *buf, size_t count)`

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.

unistd.h header file provides access to the POSIX API

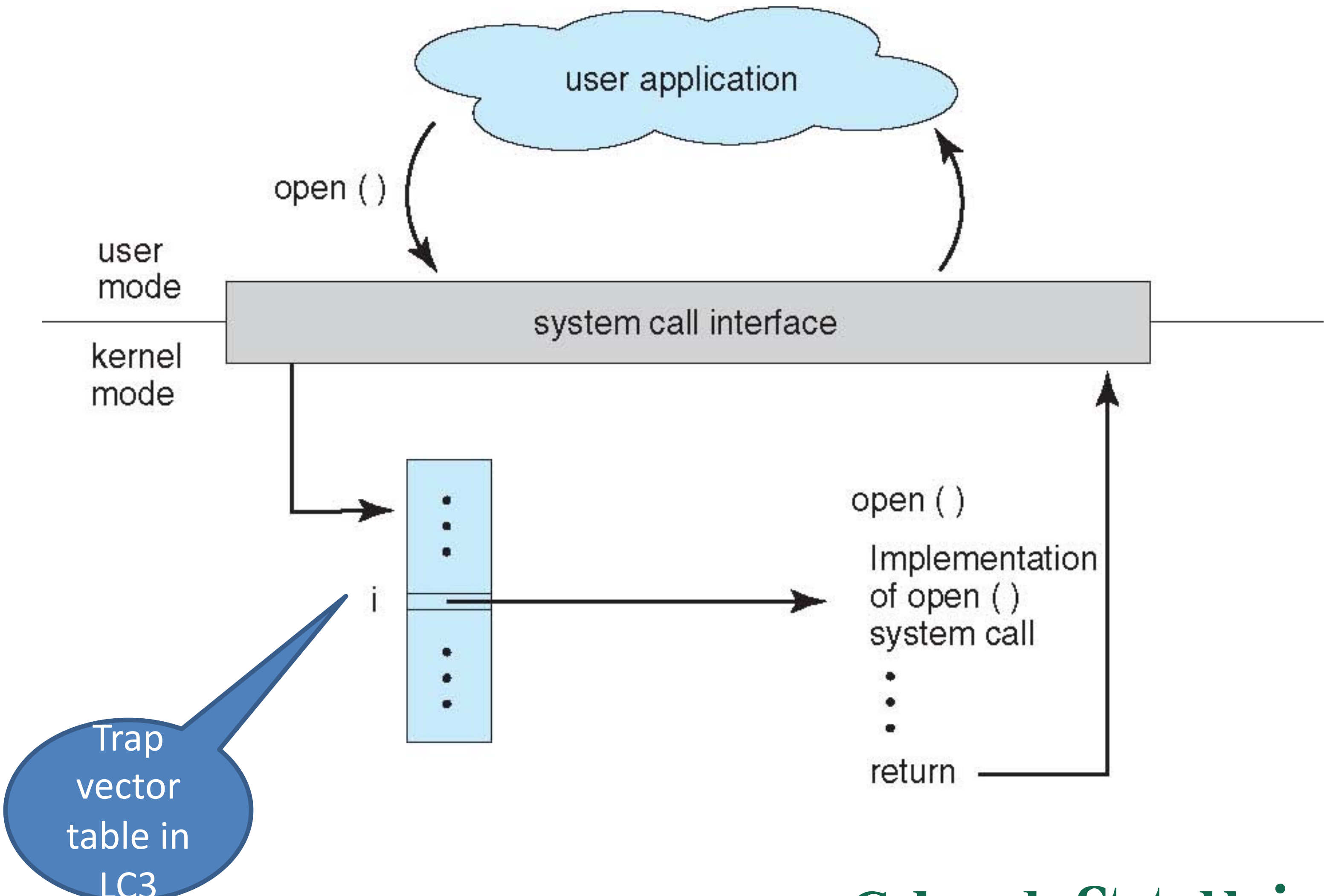
System Call Implementation

- 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

Colorado State University

API – System Call – OS Relationship

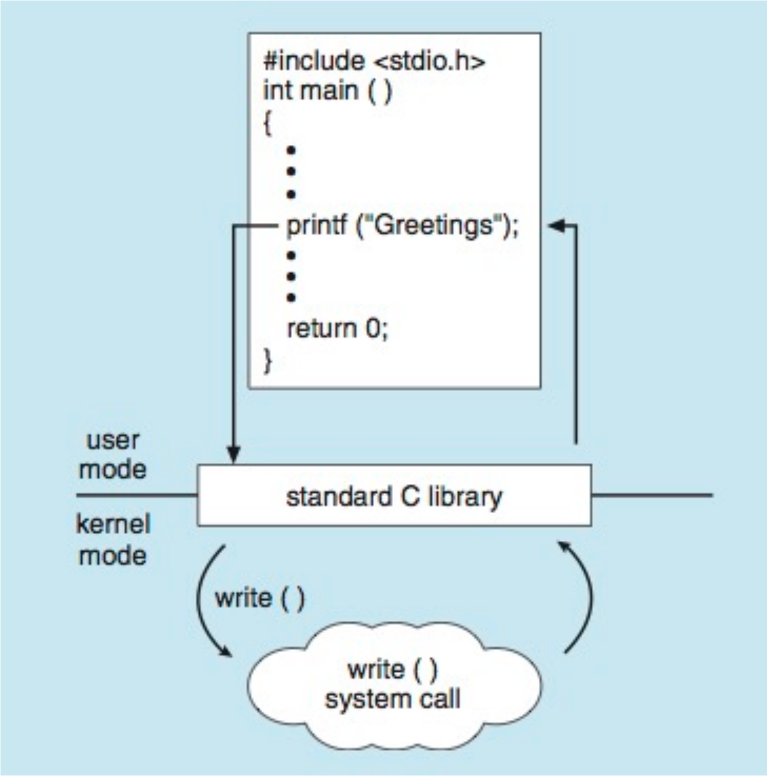


Examples of Windows and Unix System Calls

	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() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Standard C Library Example

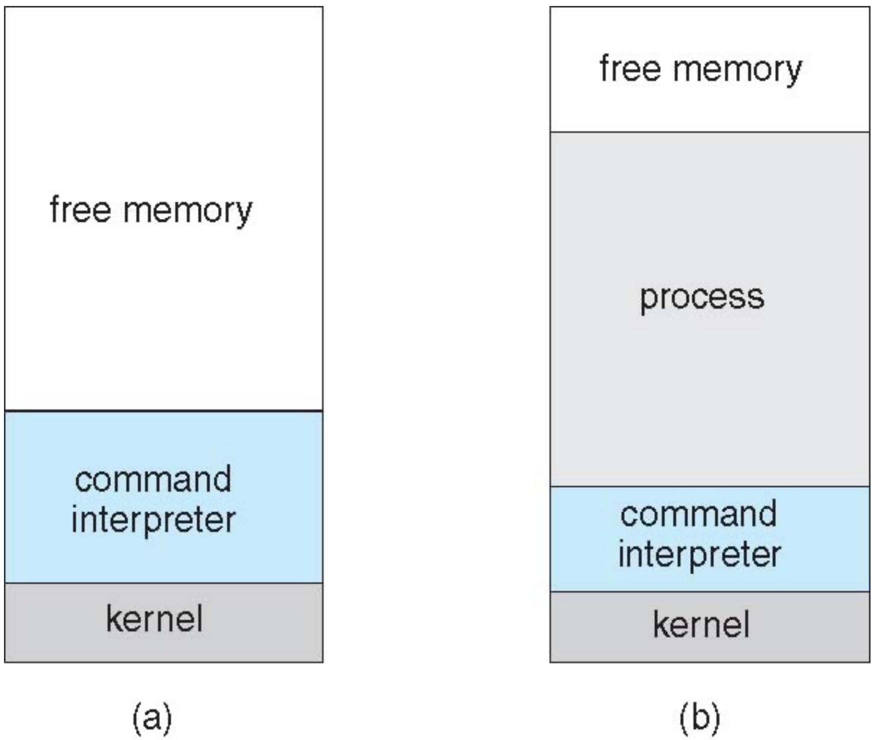
- C program invoking *printf() library call*, which calls *write() system call*



Colorado State University

Example OS: MS-DOS '81..

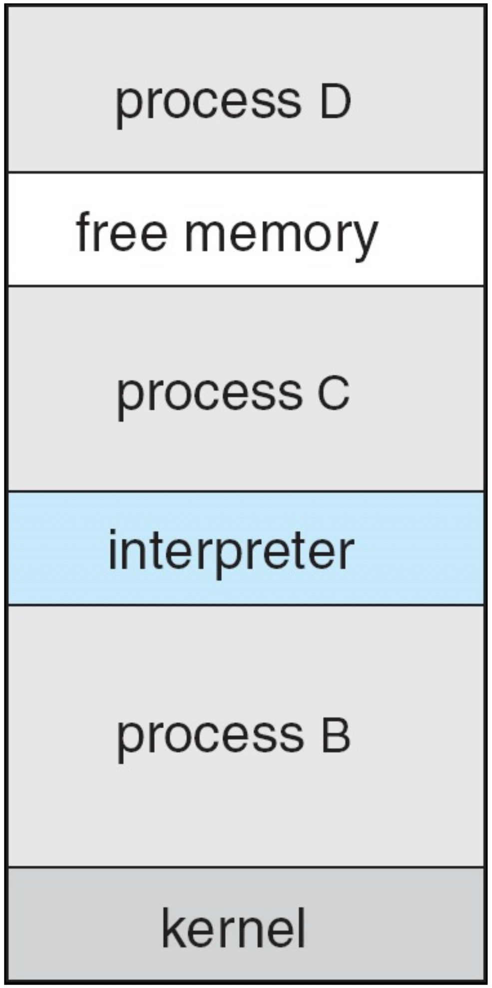
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



At system startup 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



Colorado State University

24

POSIX

- POSIX: Portable Operating Systems Interface for UNIX   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 1003.1-2008, 2016 Edition*
- Most OSs are *mostly POSIX-compliant*

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 Design

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS. not modular
  - More complex – UNIX.
    - Kernel+systems programs
  - Layered – an abstracation
  - Microkernel –Mach: kernel is minimal
  - hybrid

Tanenbaum–Torvalds debate: (January 29, 1992). "LINUX is obsolete".

Colorado State University

CS370 OS Ch3 Processes

- Process Concept: a program in execution
- Process Scheduling
- Processes creation and termination
- Interprocess Communication using shared memory and message passing

Process Concept

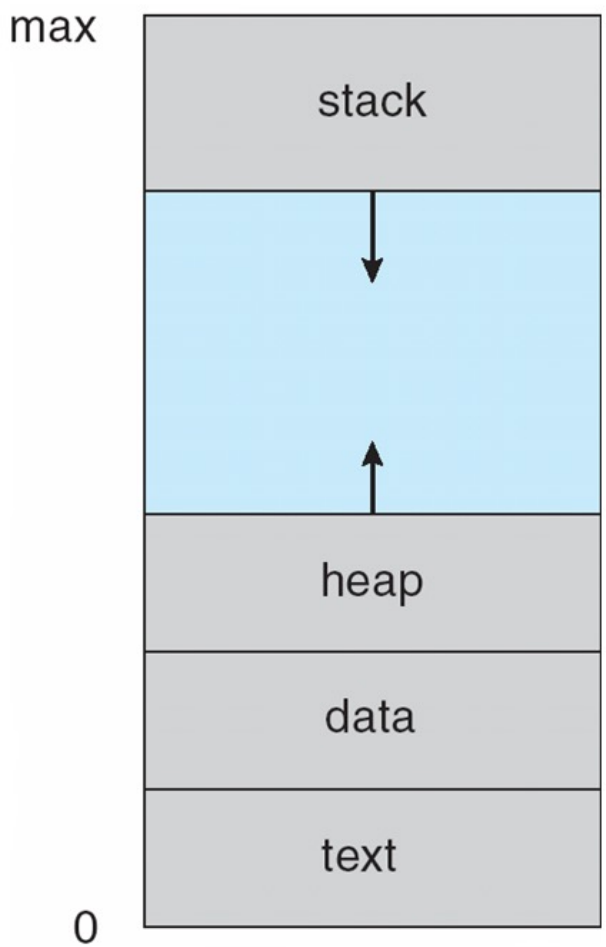
- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion. Includes
  - The program code, also called "**text section**"
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Colorado State University

Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

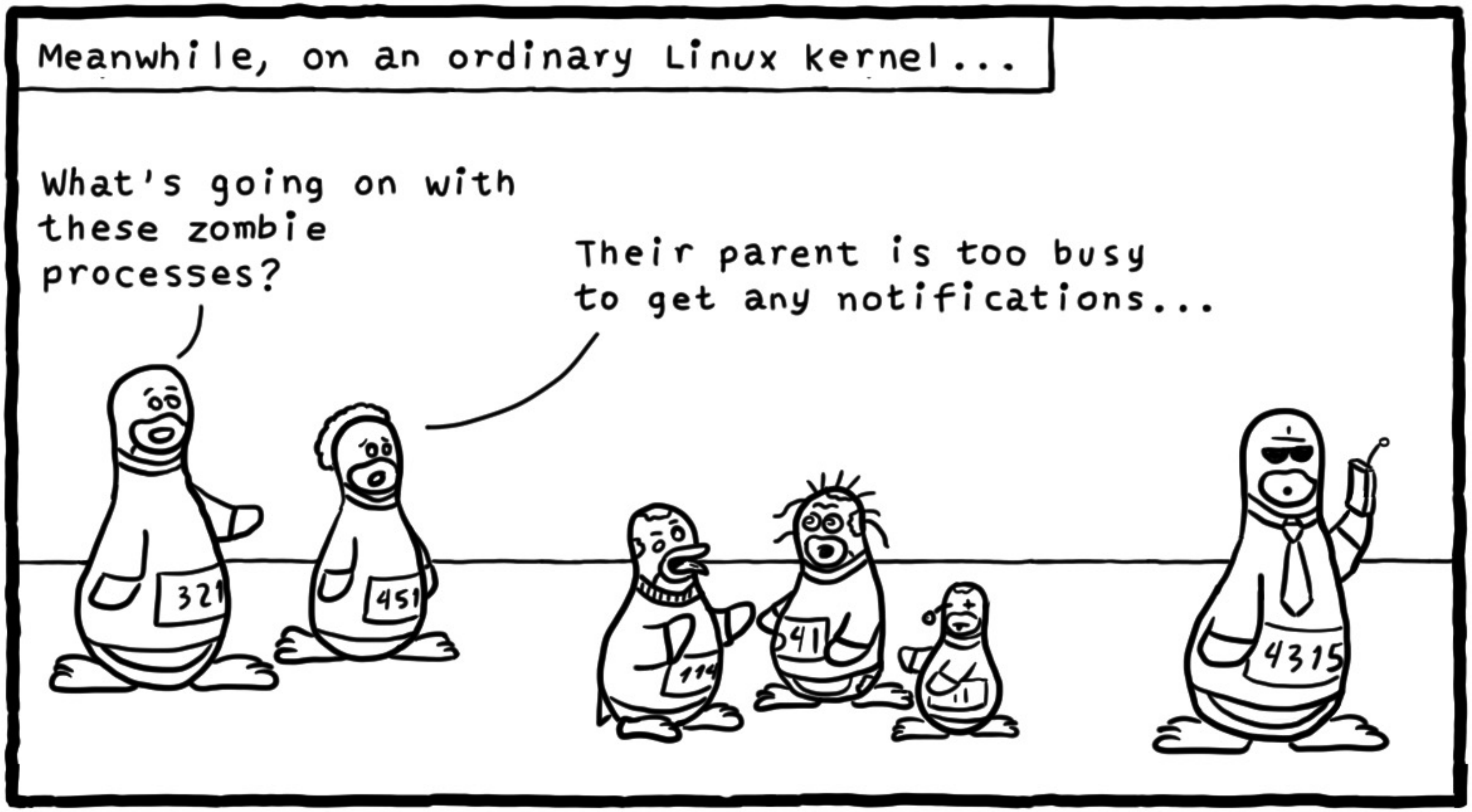
Process in Memory



Colorado State University

Process State

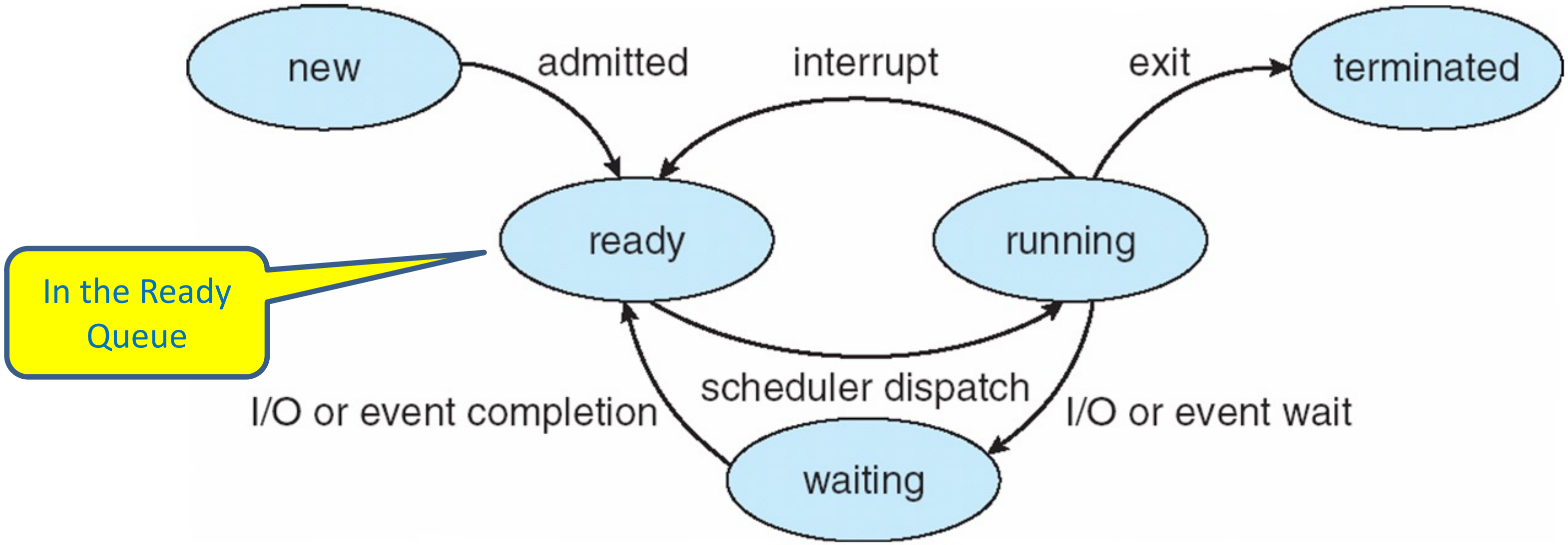
- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution



Daniel Stori {turnoff.us}

Colorado State University

Diagram of Process State



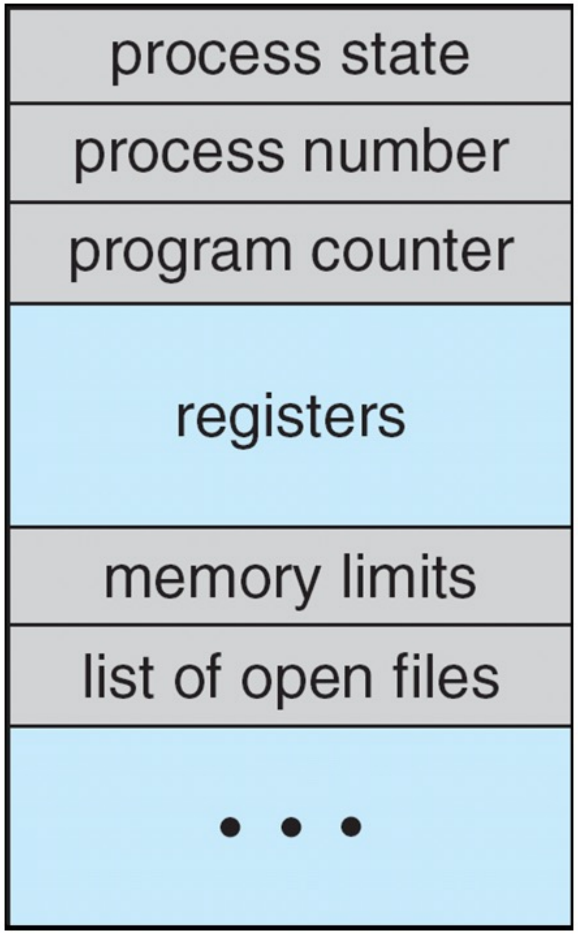
Transitions:
Ready to Running: scheduled by scheduler
Running to Ready: scheduler picks another process, back in ready queue

Running to Waiting (Blocked) : process blocks for input/output
Waiting to Ready: I/O or event done

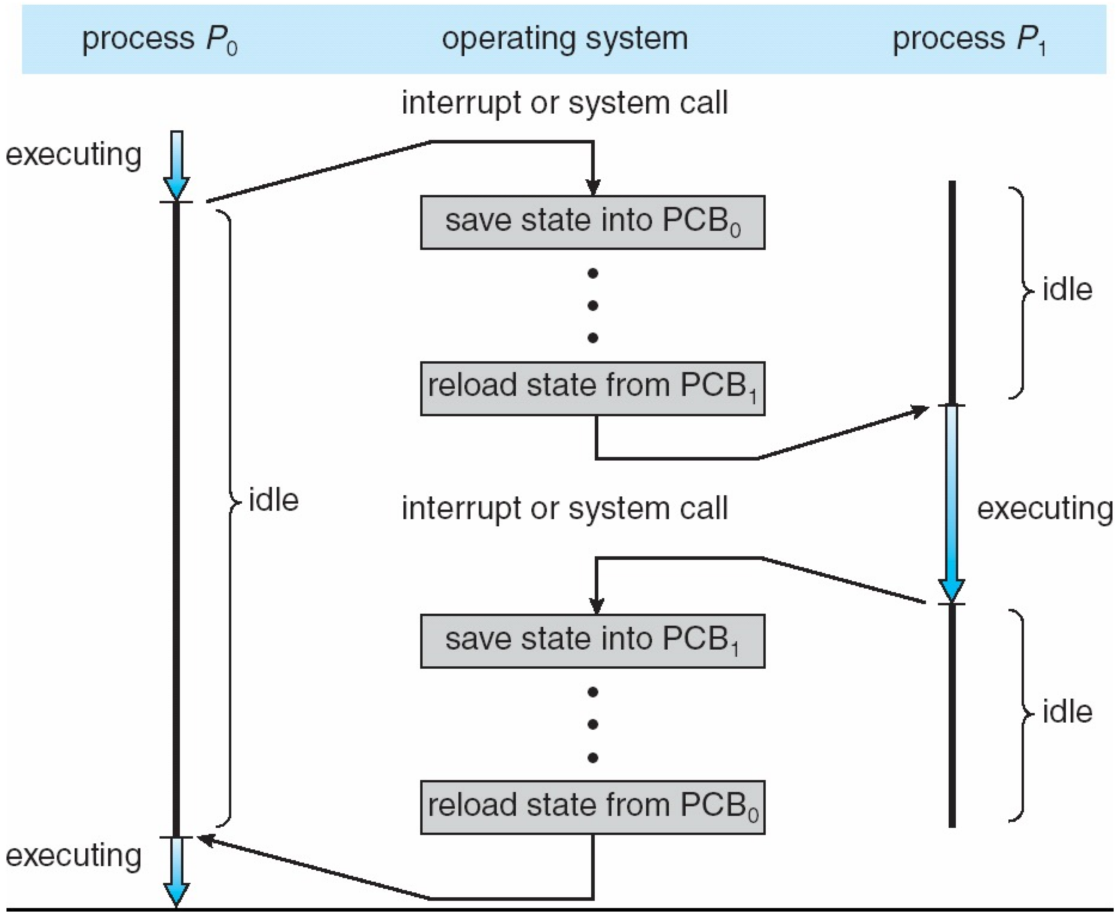
Process Control Block (PCB)

Information associated with each process (also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



CPU Switch From Process to Process



Demonstration: Processes

- Mac: apps> utilities> activity monitor > CPU etc.
- https://support.apple.com/guide/activity-monitor/welcome/mac
  - See information about processes
  - Name, PID, threads, details ..

- Windows 10 Ctrl+Alt+Del
- https://www.howtogeek.com/405806/windows-task-manager-the-complete-guide/ : Task manager