Sistemas Operativos 1

Clases

- Martes y Jueves de 19 a 22

Evaluación

- Trabajo y participación en clase
- Consignas con entregas
- Examen parcial

Bibliografía

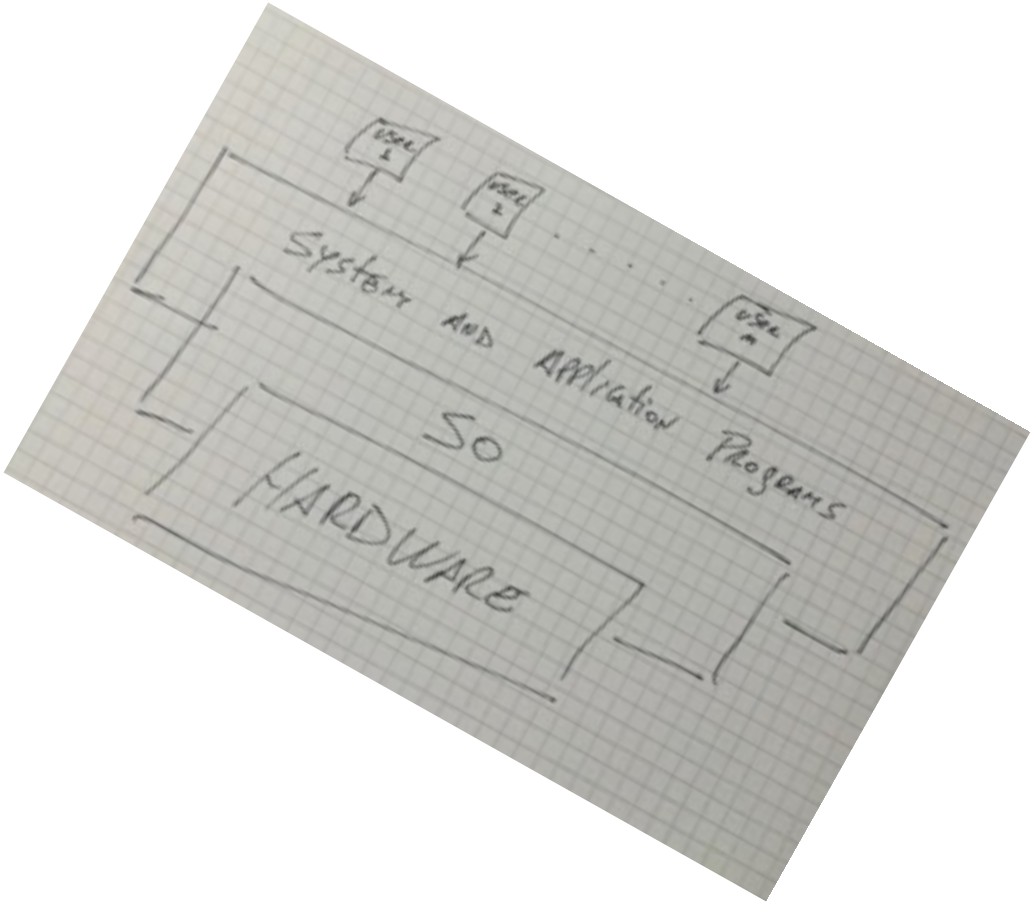
- Sistemas Operativos Modernos — Tanembaum | Pearsons
- Fundamentos de Sistemas Operativos — Silberschatz/ Galvin/ Gagne | McGraw Hill

> TPI-EST-SO@LISTAS.UNQ.EDU.AR
> NANDO - TONDAT

OPERATING-SYSTEM (OS)

QUE ES?

- Un programa (llamado KERNEL)
- Un intermediario entre el usuario y el hardware



QUE HACE?

- Maneja el hardware
- Provee un conjunto de aplicaciones
  system programs and application programs (*middleware*)
- Coordina la ejecución de los programas

ENVIRONMENT para **EJECUTAR PROGRAMAS** en forma **CONVENIENTE** y **EFICIENTE**

WHAT IS AN OPERATING SYSTEM?

- The Operating System as an Extended Machine
- The Operating System as a Resource Manager

DISEÑO?

la clave es pensar en los **OBJETIVOS**

- **USER?**
  - Fácil uso
  - Rapido acceso
  - Perfomance

  - PERSONAL COMPUTER (PC)
    monolitico aplicaciones de negocios, games
  - MAINFRAME
    optimizar uso de hardware
  - MOBILE COMPUTER
    acceso fácil y rápido a las aplicaciones usando la UI

THE OPERATING SYSTEM ZOO
- Mainframe Operating Systems
- Server Operating Systems
- Multiprocessor Operating Systems
- Personal Computer Operating Systems
- Handheld Computer Operating Systems
- Embedded Operating Systems
- Sensor-Node Operating Systems
- Real-Time Operating Systems
- Smart Card Operating Systems

- **SYSTEM?**
  - RESOURCE ALLOCATOR
  - CONTROL PROGRAM

- Administra recursos
- Administrar programas en ejecución
- Evitar ejecuciones

COMPUTER-SYSTEM ORGANIZATION

COMPUTER HARDWARE REVIEW

- Processors
- Memory
- Disks
- I/O Devices
- Buses
- Booting the Computer

RAM

ROM

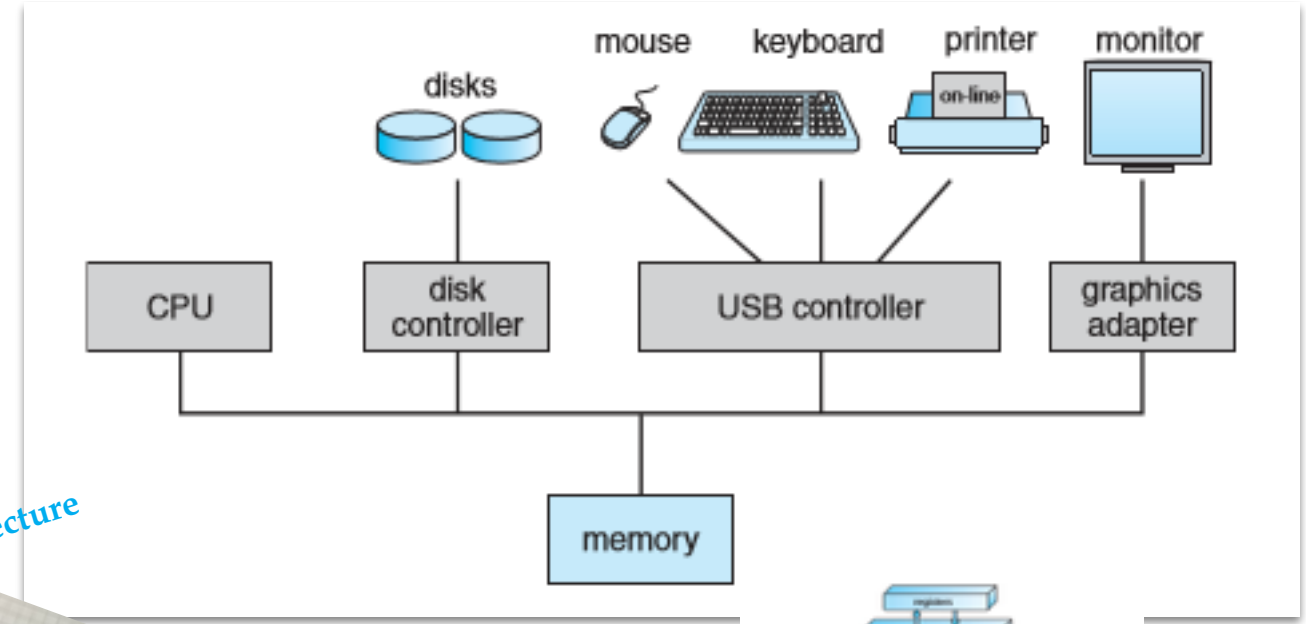
firmware

system daemons

EEPROM

direct memory access

device driver

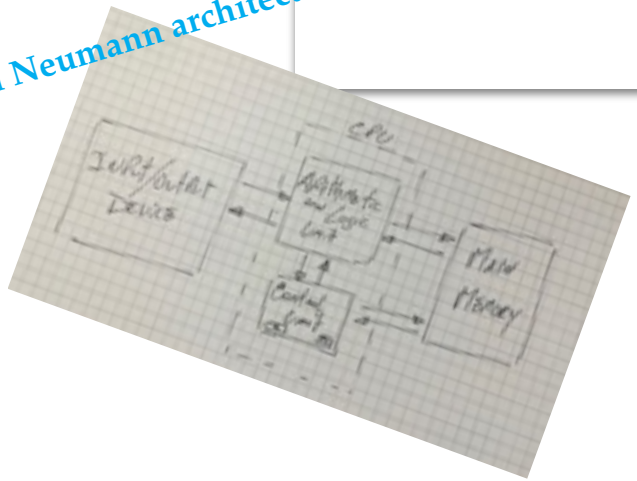


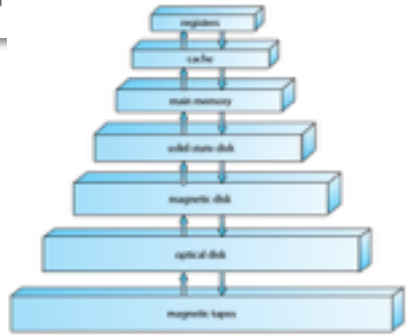
system call

bootstrap program

interrupt

von Neumann architecture





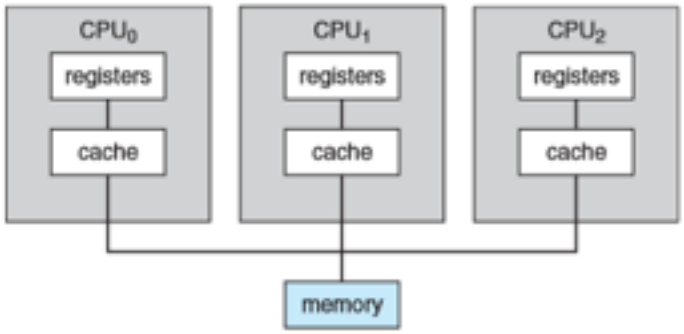
storage

rebooted

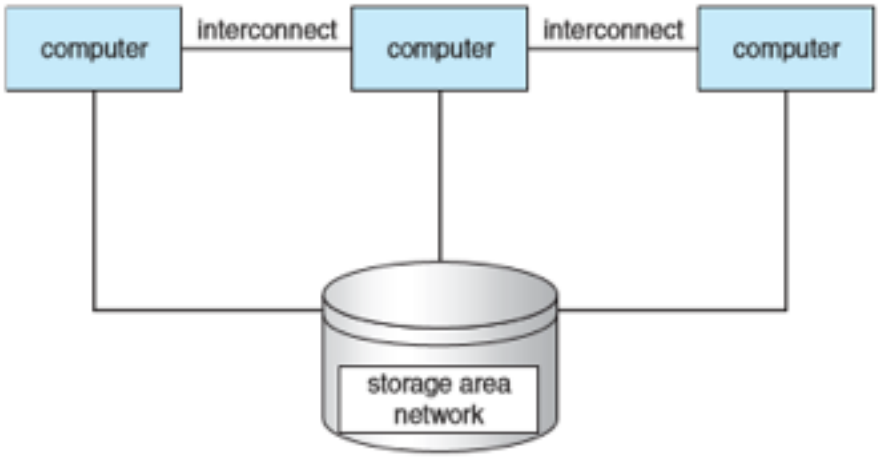
COMPUTER-SYSTEM ARCHITECTURE

1. SINGLE-PROCESSOR SYSTEMS

2. MULTIPROCESSOR SYSTEMS



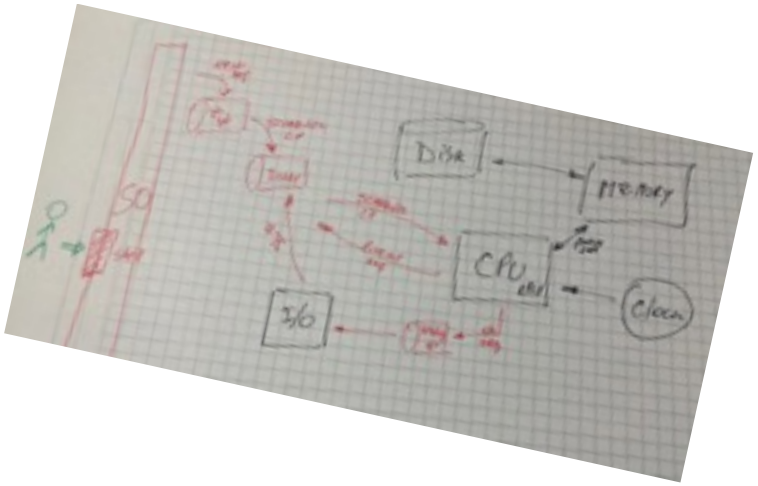
3. CLUSTERED SYSTEMS



Operating-System STRUCTURE

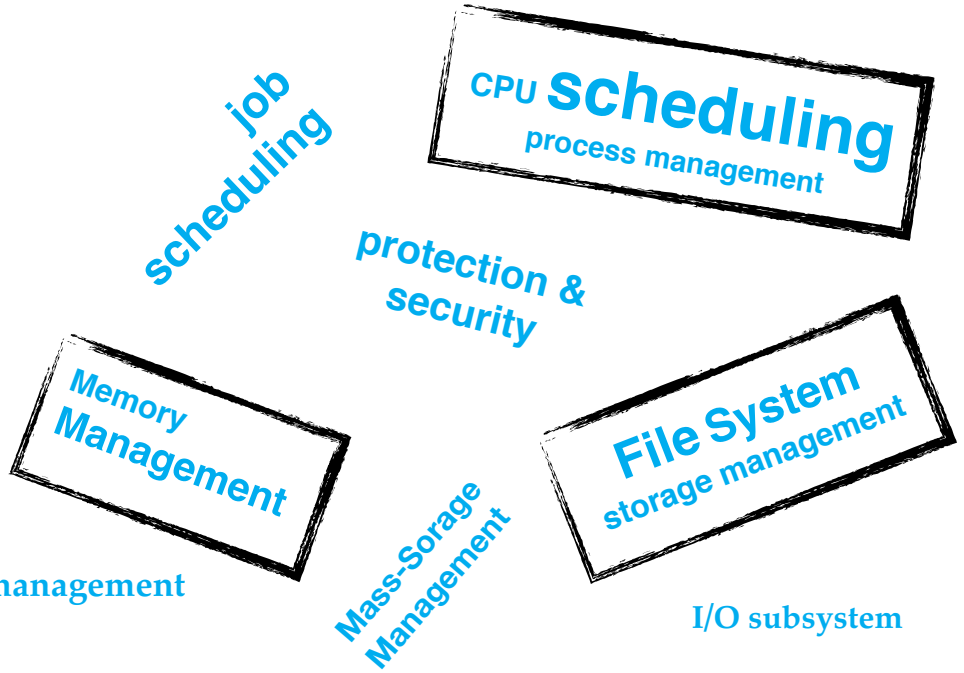
MULTIPROGRAMMING

- INCREMENTA USO de CPU
- JOB POOL en disco
- CPU NO IDLE
- NO INTERACCION con el USUARIO



TIME SHARING (multitasking)

- INTERACCION con el USUARIO - input device
- ACCIONES CORTAS
- POCO TIEMPO de CPU.
- sensación SISTEMA DEDICADO
- asegura tiempo de respuesta
- require PROGRAMAS cargados EN MEMORIA
- ORDEN directa al SO
- **process:** programa en memoria en ejecución



Operating-System OPERATIONS

- operating systems are ***interrupt driven***
- en espera hasta que algo suceda…
- EVENT: ***interrupt*** or ***trap***
- ***interrupt*** ROUTINE

trap (***exception***): software-generated interrupt caused either by an error (division by zero or invalid memory access) or by a specific request from a user program.

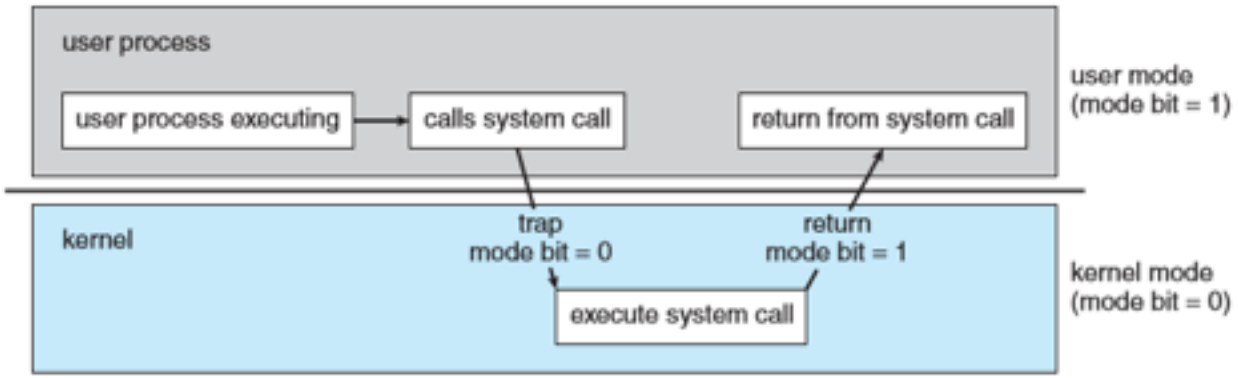
*Un **ERROR** en un programa **SOLO** debe afectar dicho **programa***

HARDWARE **MODE BIT** with ***PRIVILEGES INSTRUCTIONS*** and syscall

USER MODE

virtual

KERNEL MODE



Dual-Mode and Multimode Operation

Timer

Kernel Data Structures

→ the way data are structured in the system

- Lists, Stacks, and Queues

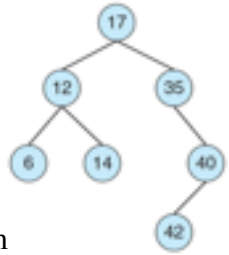
  $O(n)$

  - **stack** (LIFO) push/pop
  - **queue** (FIFO)
  - ejemplo: ready queue

- Trees

  $O(lg\ n)$

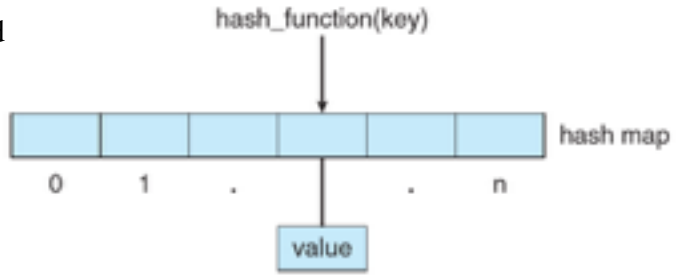
  - **binary search tree**
  - **balanced binary search tree**
  - ejemplo: CPU scheduling algorithm



- Hash Functions and Maps

  $O(1)$

  - **hash map**
  - ejemplo: user password



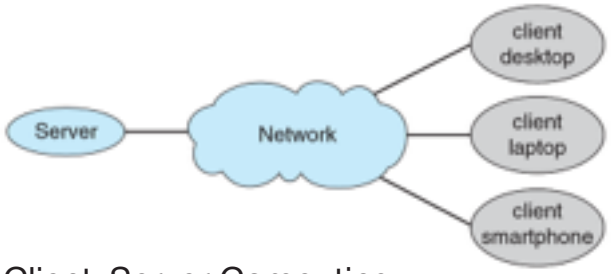
- Bitmaps

  - and/or/xor
  - ejemplo: resources disponibles

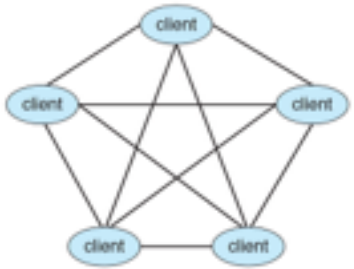
001011101

Computing Environments

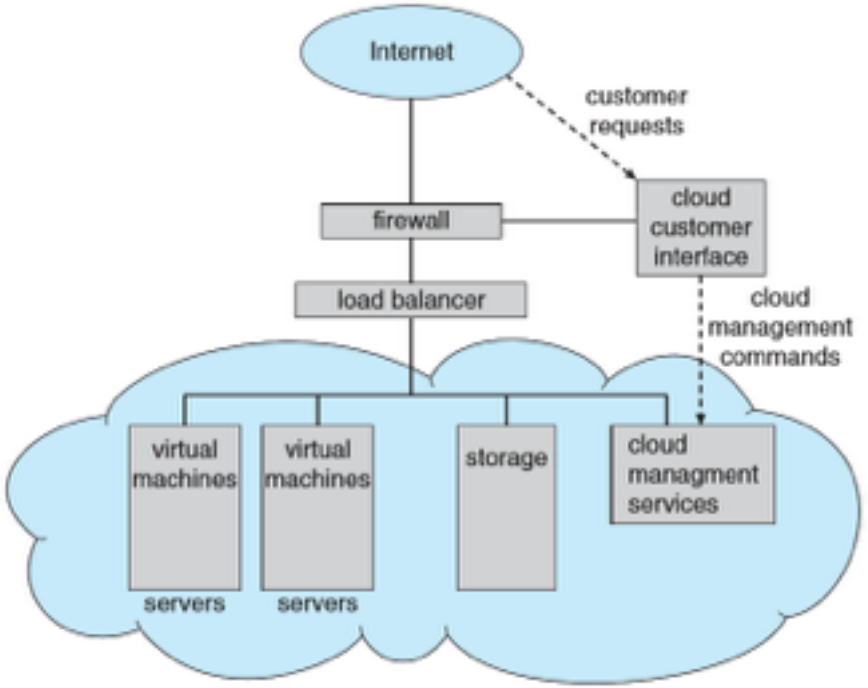
how operating systems are used in a variety of computing environments



Client–Server Computing



Peer-to-Peer Computing



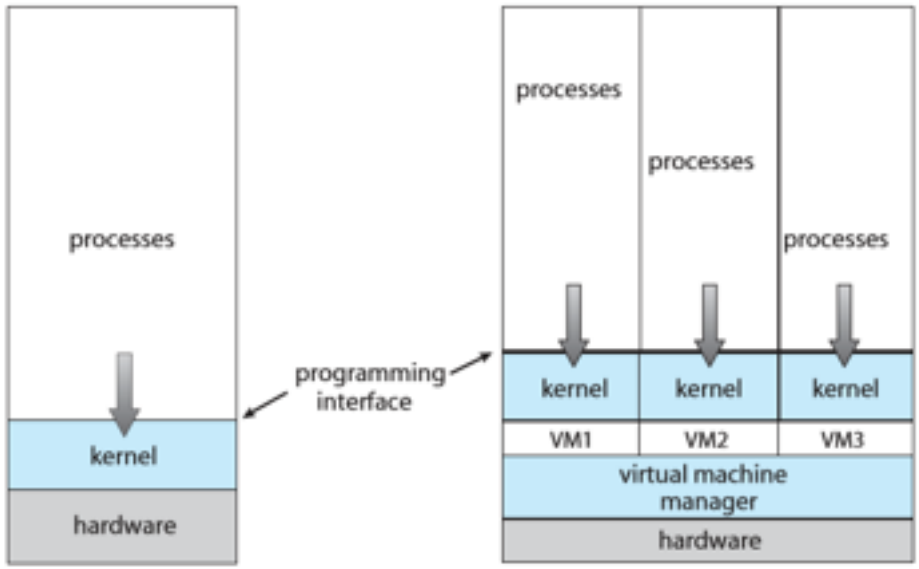
Cloud Computing

Traditional Computing

Mobile Computing

Distributed Systems

Real-Time Embedded Systems

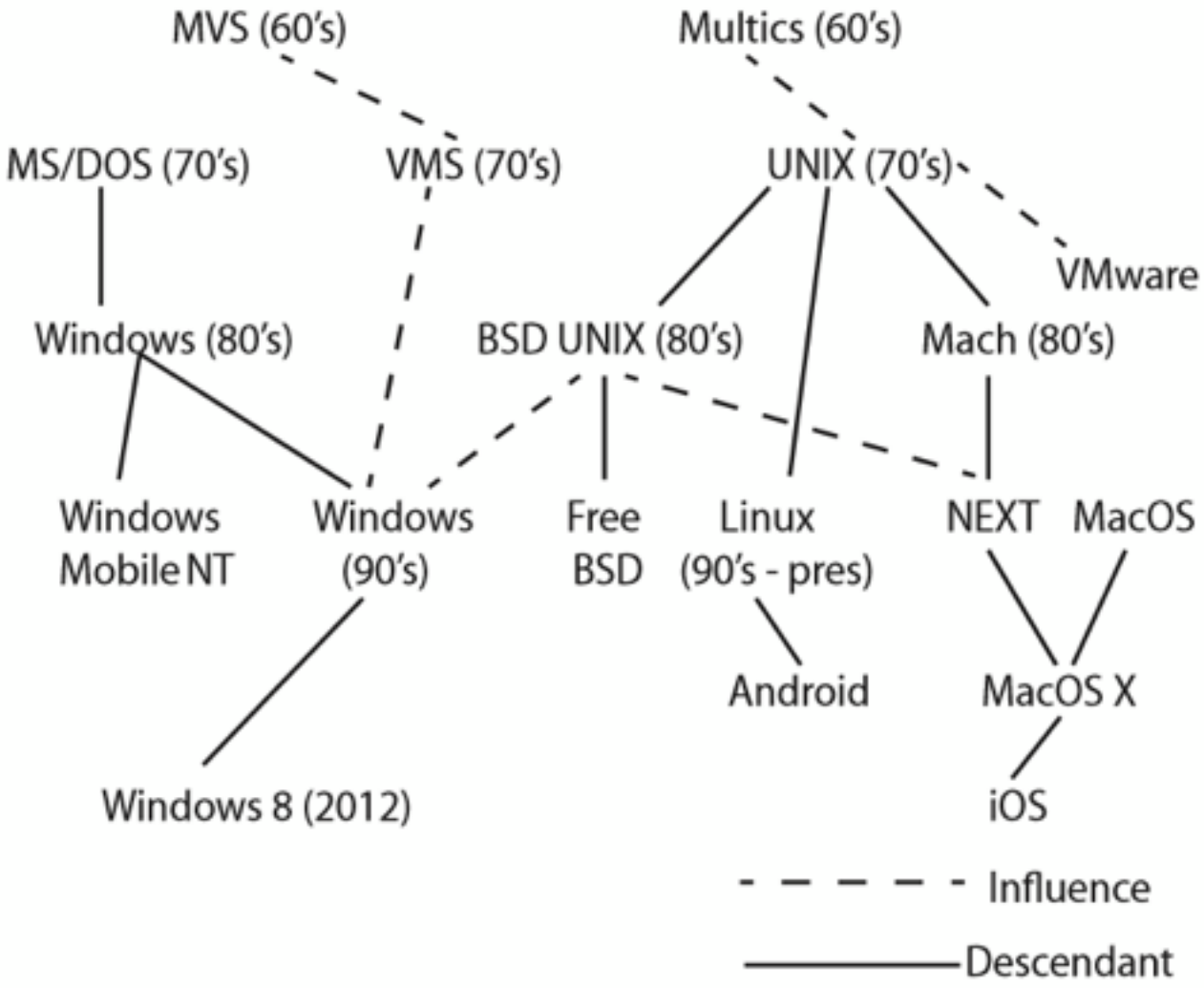


Virtualization

Operating Systems
EVOLUTION

HISTORY OF OPERATING SYSTEMS

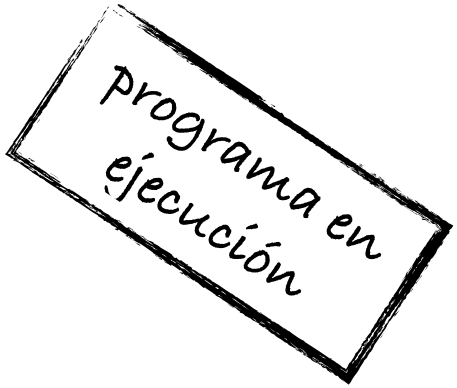
- The First Generation (1945–55): Vacuum Tubes
- The Second Generation (1955–65): Transistors and Batch Systems
- The Third Generation (1965–1980): ICs and Multiprogramming
- The Fourth Generation (1980–Present): Personal Computers
- The Fifth Generation (1990–Present): Mobile Computers



Operating-System

1. provides the environment within which programs are executed
2. services that the system provides
3. interface that it makes available to users and programmers
4. components and their interconnections

PROCESS



- ***process table***
- ***address space****:* area de memoria asociada al proceso
- ***resources***: recursos asociados al proceso
- ***records***: valores de registros de CPU
- ***open files****: file descriptor / special files I/O / pipe*
- ***alarm signal***
- ***child process***: lista de procesos relacionados
- ***pid/uid/gid***

two main functions:

1. providing abstractions to user programs
2. managing the computer's resources

Operating-System SERVICES

- **USER**

1. **User interface (UI)**
2. **Program execution**
3. **I/O operations.**
4. **File-system manipulation**
5. **Communication** shared memory/message passing

- **command-line interface (CLI)**

  bash/MS DOS

- **batch interface**

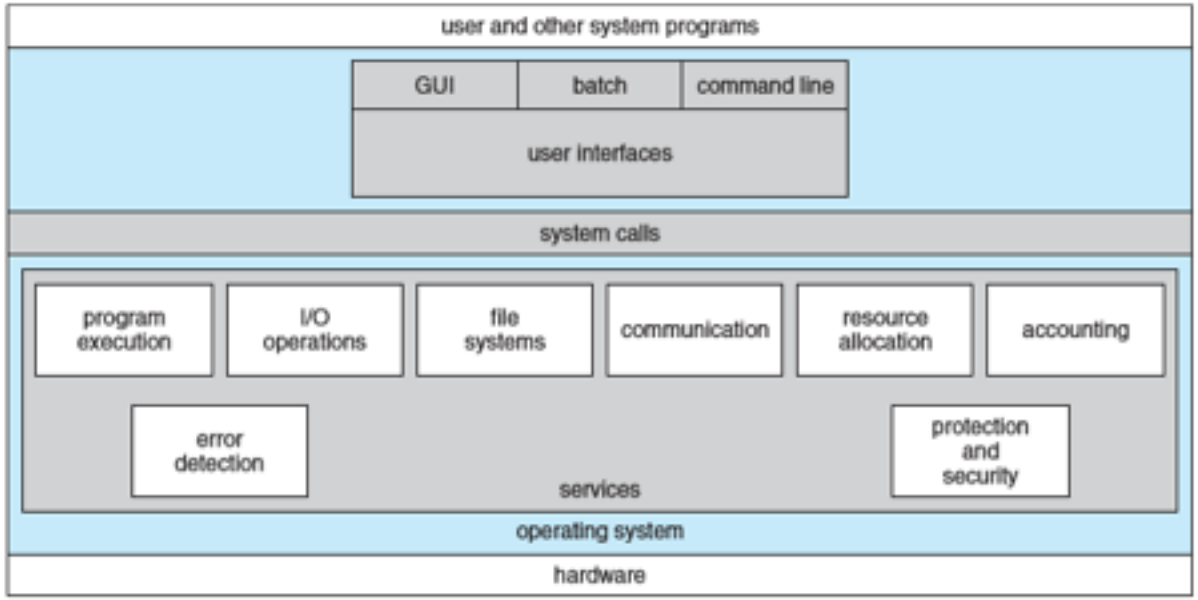
  shell script/.bat

- **graphical user interface (GUI)**

  K Desktop Environment (KDE) /GNOME

- **SYSTEM**

1. **Resource allocation.**
2. **Accounting** usage statistics
3. **Protection and security**



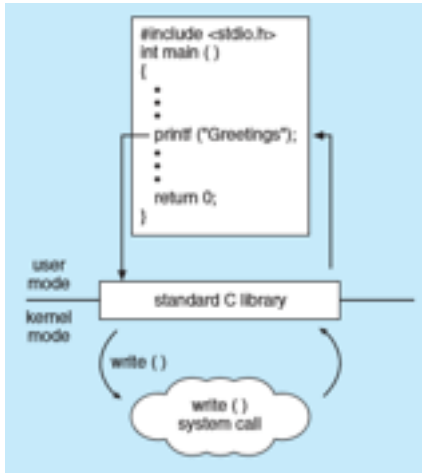
SYSTEM CALLS

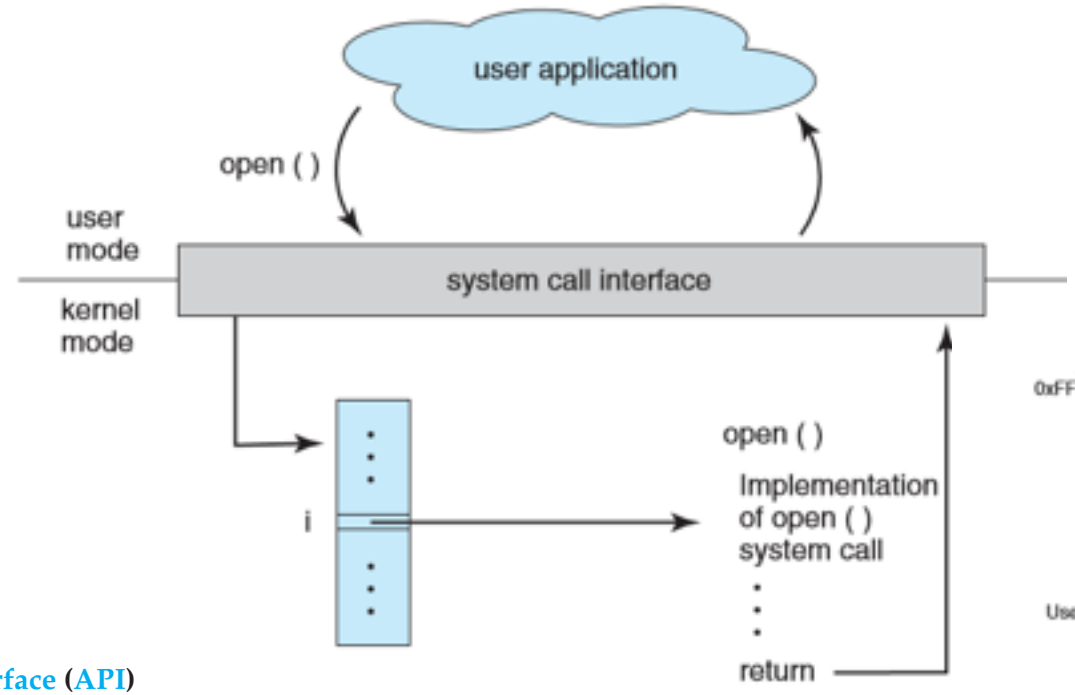
provide an interface to the services made available by an operating system

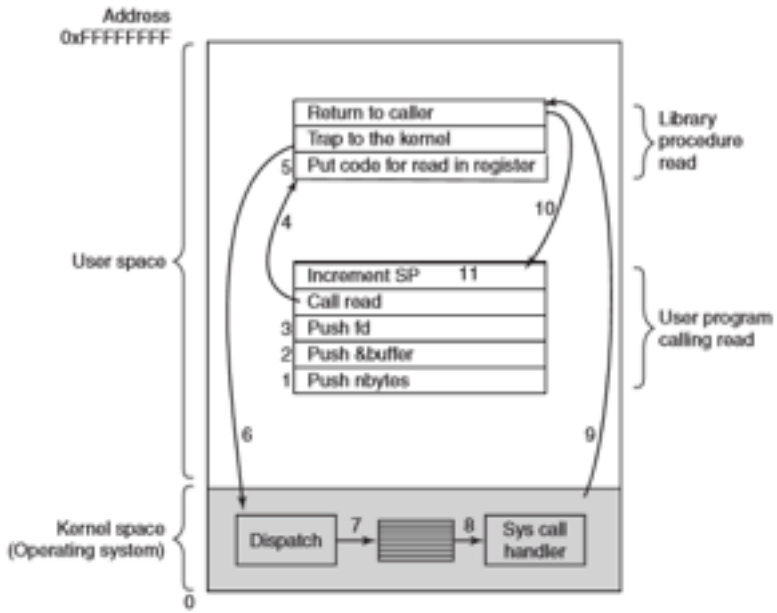
- rutinas en C/C++
- low-level tasks en assembly-language

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

return value — function name — parameters







application programming interface (API)

Windows API / POSIX API / Java API

the goal: PORTABILITY

the run-time support system (libc) provides a **system- call interface**

SYSTEM CALLS TYPES

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

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()`
		`shm_open()`
		`mmap()`

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

DESIGN

- One important principle is the separation of **policy** from **mechanism**.
- Mechanisms determine ***how*** to do something
- Policies determine ***what*** will be done.
- The separation of policy and mechanism is important for ***flexibility***.

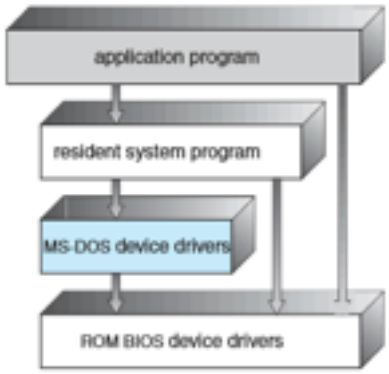
IMPLEMENTATION

MS-DOS
enteramente escrito en assembly-language
x-86 family
no es portable
requiere emulador

Linux
escrito en C
portable
issues de performance se reemplaza por assembly-language

- Lowest levels: Operating systems were written in assembly language (lowest levels)
- Higher-level routines: written in a higher-level language such as C or C++, in interpreted scripting languages like PERL or Python, or in shell scripts

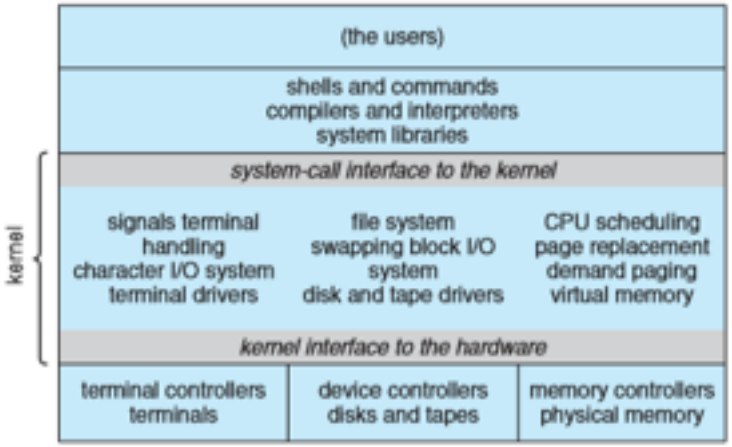
Operating-System Structure



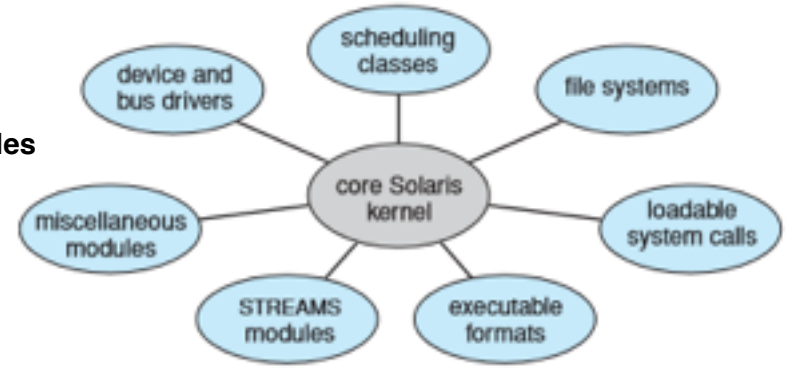
MS-DOS

Linux **monolithic**, because having the operating system in a single address space provides very efficient performance. However, they are also **modular**, so that new functionality can be dynamically added to the kernel.

Windows is largely **monolithic** as well (for performance reasons), but it retains some behaviour typical of **microkernel** systems, including providing support for separate subsystems (known as operating-system ***personalities***) that run as user-mode processes. Windows systems also provide support for dynamically loadable kernel **modules.**

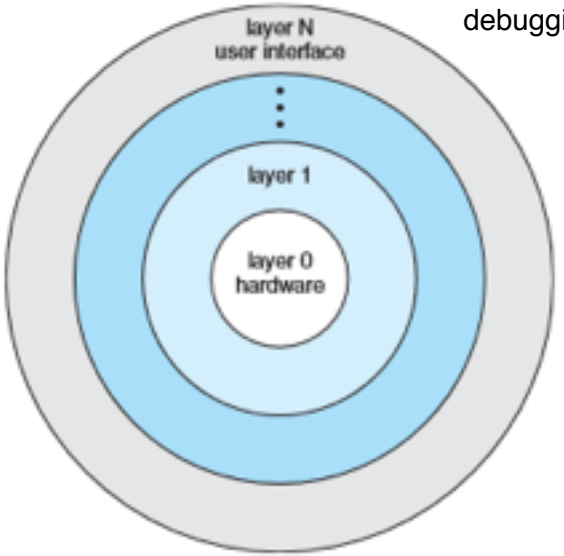


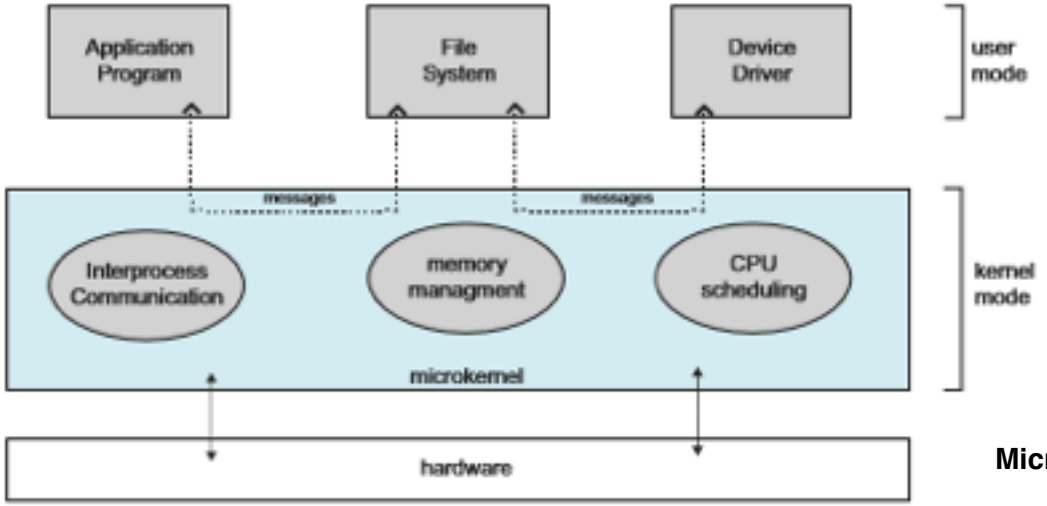
UNIX



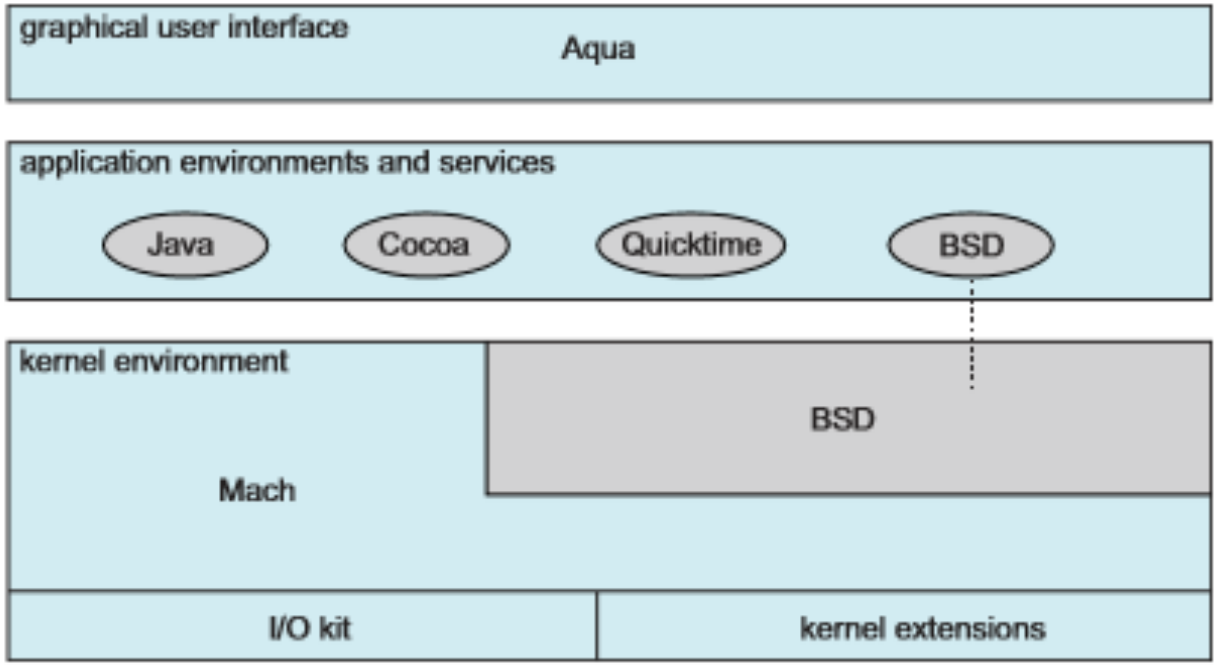
Modules

Layers: simplicity of construction and debugging



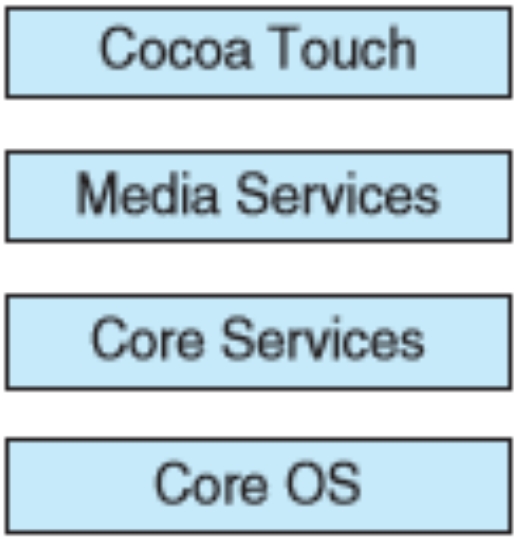


Microkernels

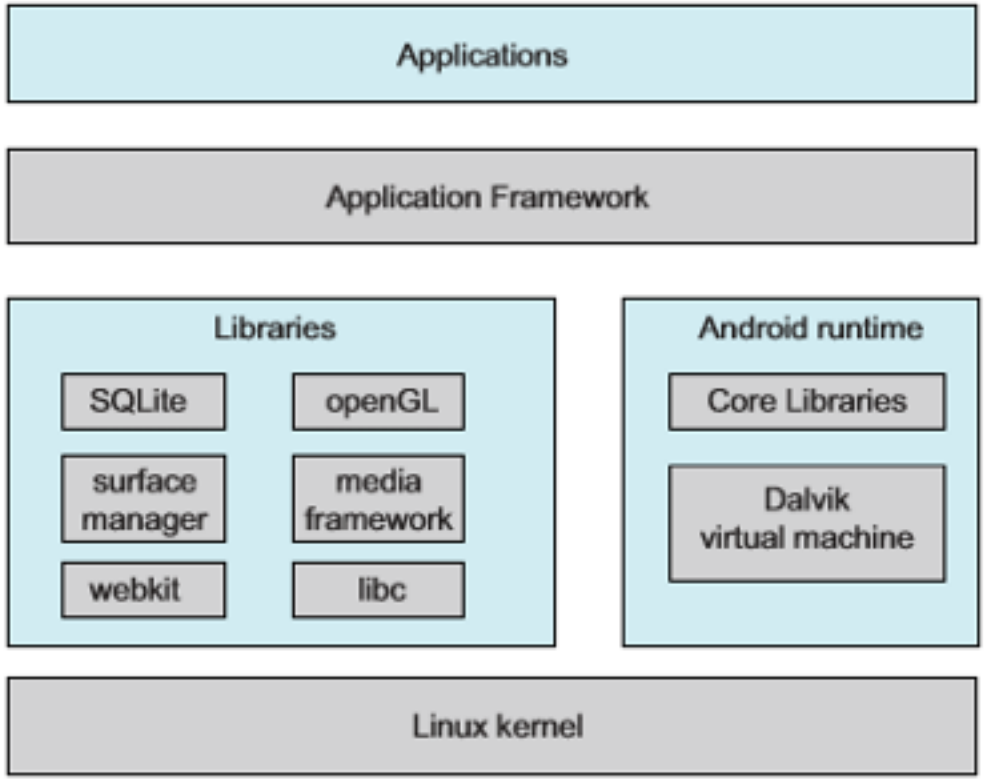


Mac OSX

Android



iOS



PROCESS MANAGEMENT

MEMORY MANAGEMENT

STORAGE MANAGEMENT

PROTECTION AND SECURITY

VIRTUAL MACHINES

DISTRIBUTED SYSTEMS

CASE STUDIES

LINUX
WINDOWS
ANDROID
iOS

PREGUNTAS ?