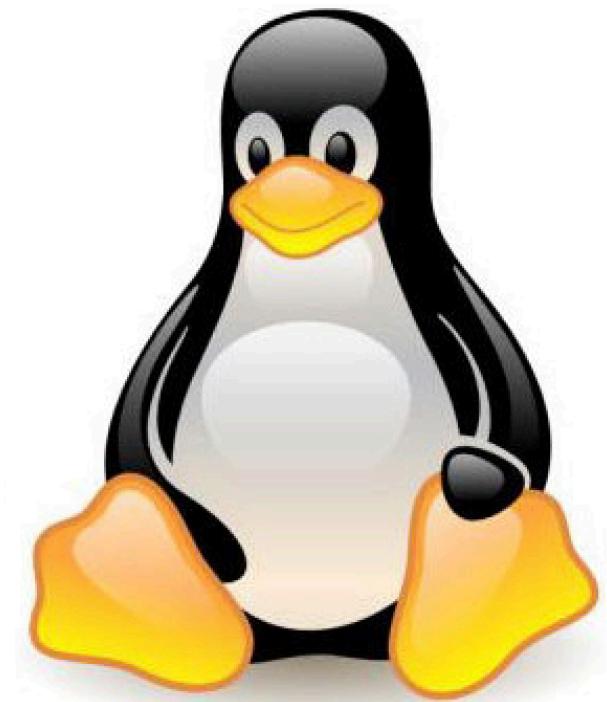
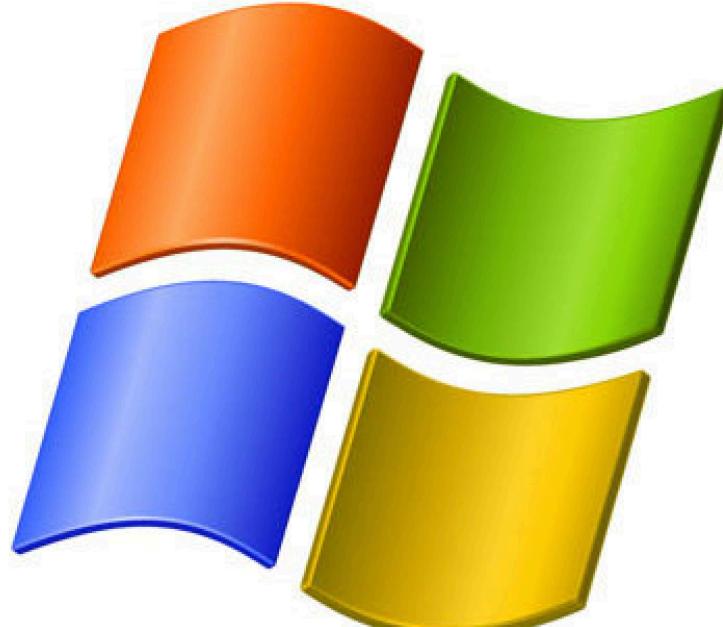


Tema 1: Introducción

Sistemas Operativos

ETSI Informática. Universidad de Málaga



Manuel Ujaldón

Catedrático de Arquitectura de Computadores
Departamento de Arquitectura de Computadores
Universidad de Málaga

Índice [38 diapositivas]

I. Concepto de Sistema Operativo [5]

II. Evolución Histórica [4]

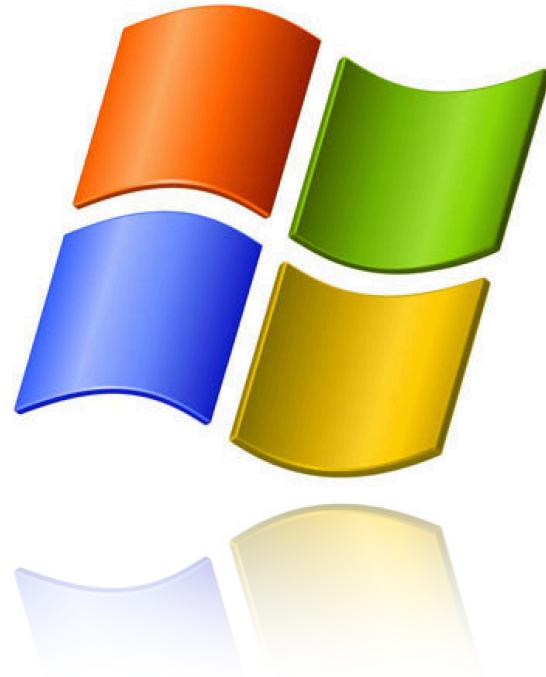
III. Funciones y Servicios [12]

1. Gestión de recursos [1]
2. Interfaz de usuario [3]

IV. Estructura e Implementación [10]

V. Soporte y Protección Hardware [6]

1. Modo dual de operación [2]
2. Protección de la CPU [1]
3. Interrupciones [2]



I. Concepto de Sistema Operativo

Componentes de un sistema informático

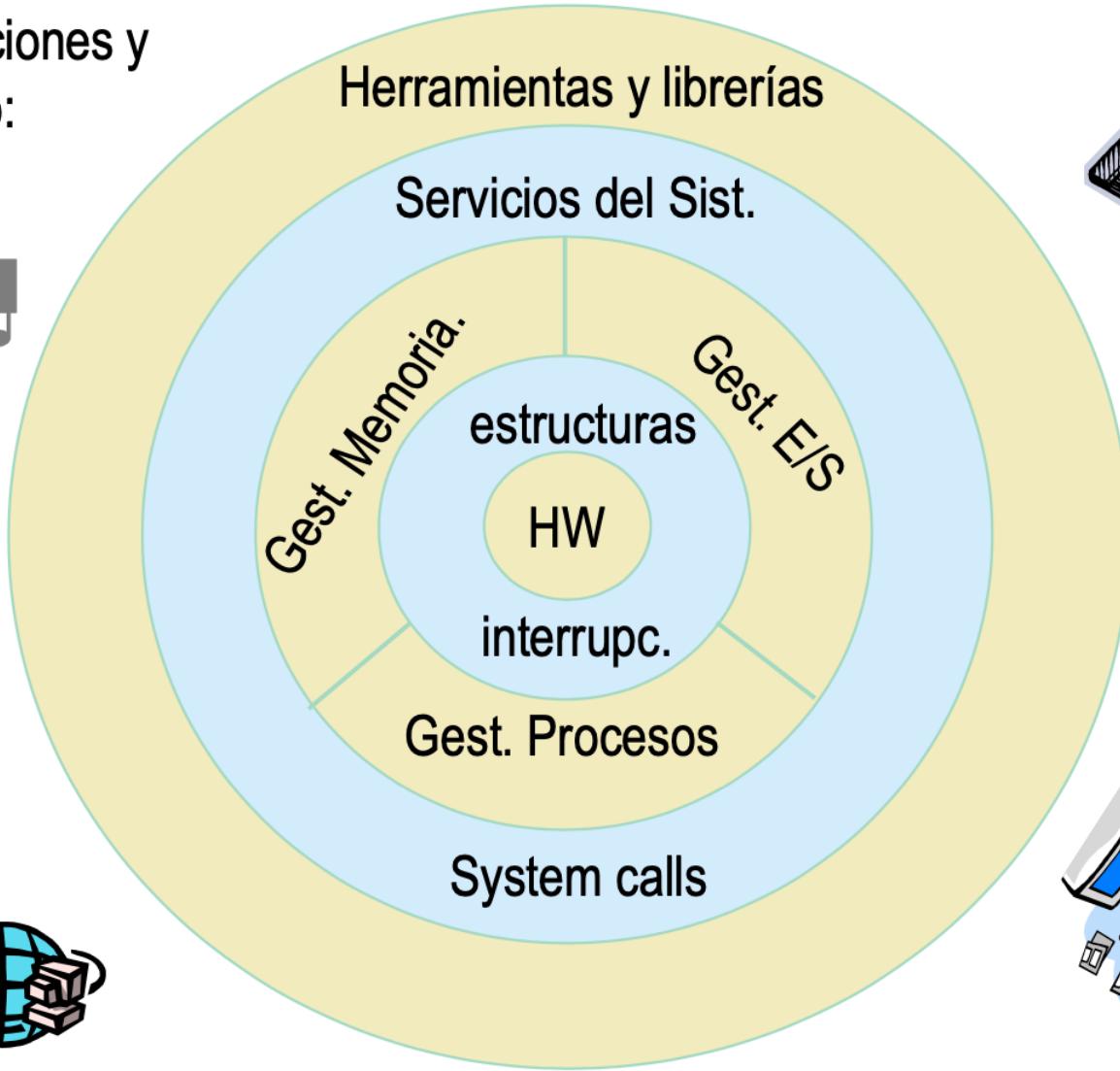
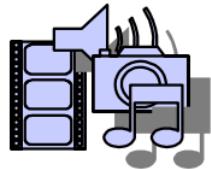
	Caracterización	Ejemplos
La circuitería (hardware)	Proporciona los recursos básicos de computación	CPU, GPU, memoria, dispositivos de entrada/salida
El sistema operativo	Controla y coordina el uso del <i>hardware</i> por parte de las aplicaciones y los usuarios	UNIX/Linux, MacOS, Windows
Las aplicaciones	Establecen la forma de usar los recursos para resolver los problemas computacionales	Compiladores, bases de datos, navegadores Web, lectores de correo, video-juegos, ...

¿Qué es un sistema operativo?

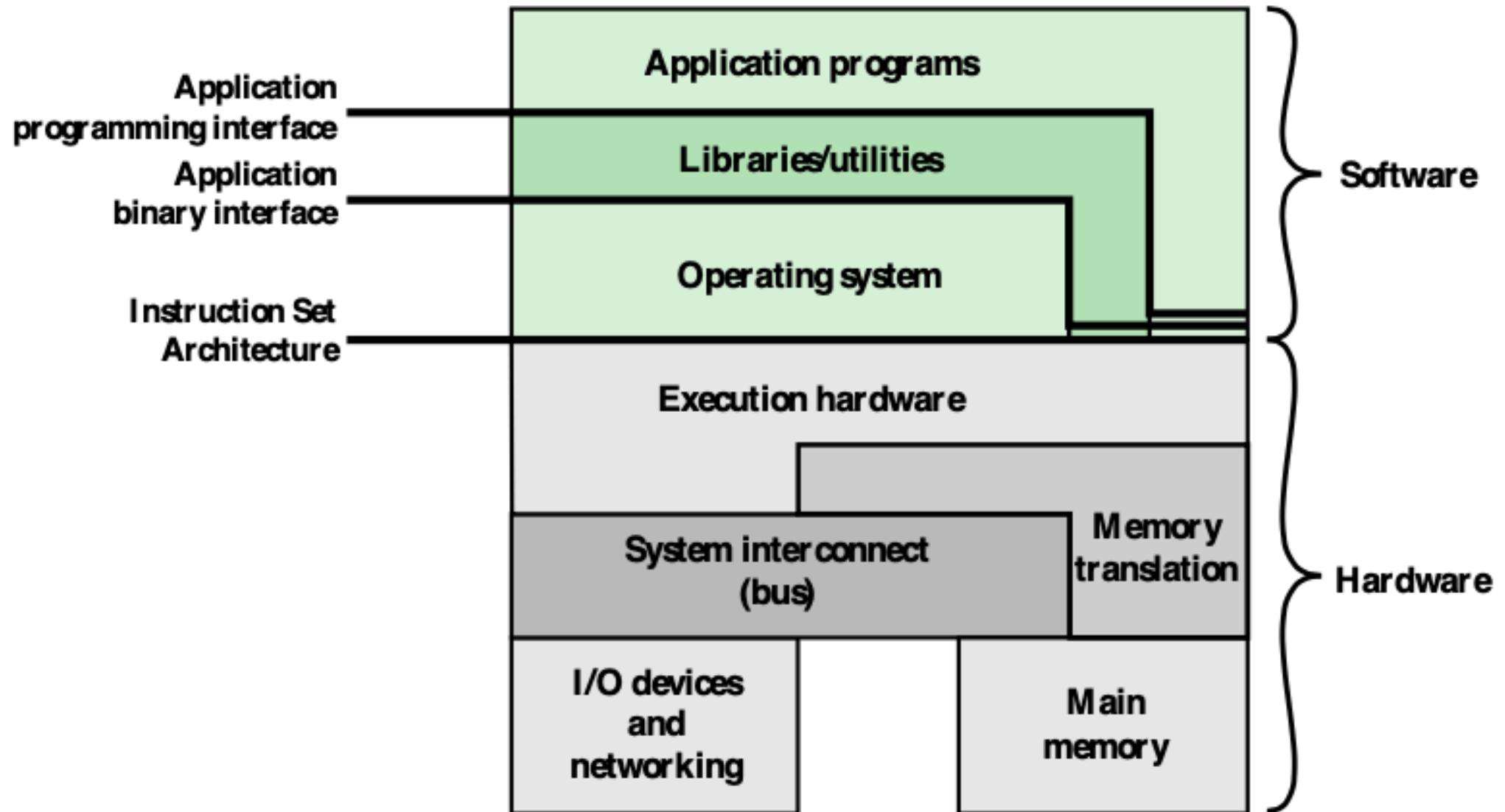
- Un programa que hace de intermediario entre el usuario y la circuitería.
- Un conjunto de programas implementados como software o firmware que facilitan al usuario el uso de la circuitería.
- Un programa con un repertorio de funciones encaminadas a simplificar la gestión y el uso del computador de forma segura y eficiente.
- Un programa que gestiona los recursos de la máquina (CPU, memoria, discos, comunicaciones, ...).

¿Dónde se ubica el sistema operativo dentro del computador?

SW de aplicaciones y
Usuario:



Estructura del sistema de computación



Objetivos del sistema operativo

- Son básicamente tres:

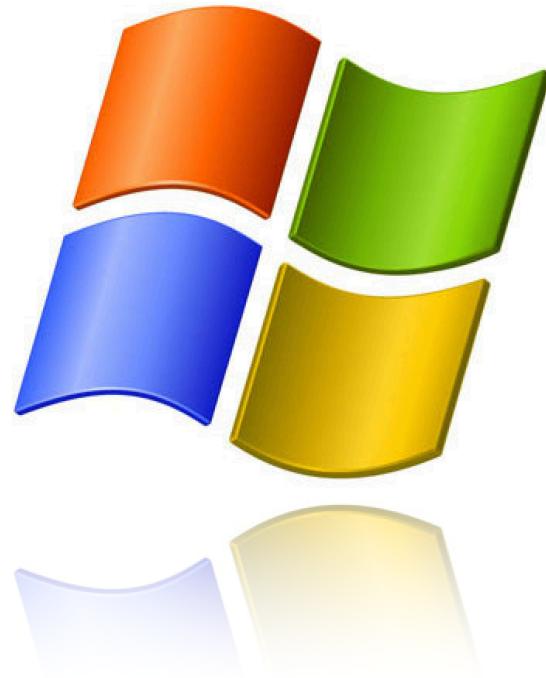
- Ejecutar los programas del usuario.
- Facilitar el uso del equipo.
- Utilizar los recursos de una manera eficiente.

- Desde la perspectiva del usuario:

- El sistema operativo debe ser cómodo de usar, fácil de aprender, fiable, seguro y rápido.

- Desde la perspectiva del sistema:

- El sistema operativo debe ser fácil de diseñar, implementar y mantener, además de flexible, fiable y eficiente.



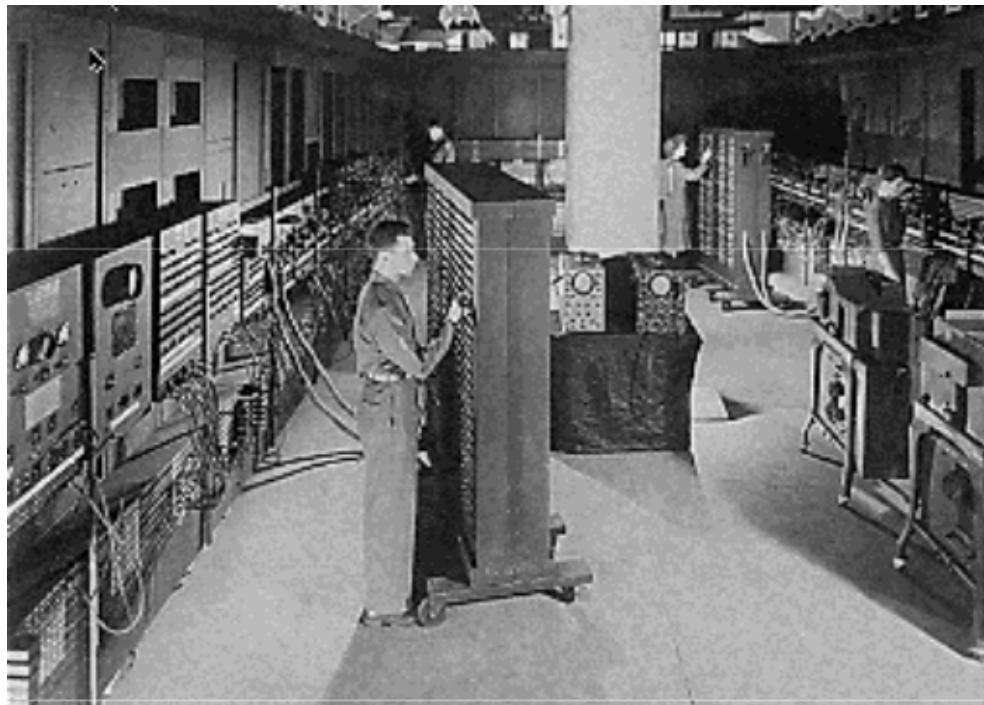
II. Evolución Histórica

Resumen generacional

	Marco temporal	Tecnología predominante	Ejemplos	Personajes ilustres
Pre-electrónica	Antes de 1945	Accionamiento mecánico, bastantes fiascos	<i>Analytical machine</i>	Charles Babbage (1792-1871)
Primera generación	1945-1955	Tubos de vacío: pesados y de gran consumo	ENIAC: 30.000 Kg, 200 kW	von Neumann, Eckert-Mauchly (ENIAC)
Segunda generación	1955-1965	Transistores. Aparecen lenguajes y compiladores	Tarjetas perforadas	John Bardeen, Walter Brattain, William Shockley
Tercera generación	1965-1980	Circuitos integrados. Multiprogramación, tiempo compartido, memoria virtual	Terminales interactivos	Gordon Moore, Robert Noyce (Intel)
Cuarta generación	A partir de 1980	VLSI. Coste reducido, tamaño manejable	PCs domésticos, MS-DOS, Mac	Bill Gates (MS), Steve Jobs (Apple)



Resumen visual: ENIAC (1946), UNIVAC 1 (1951), IBM S/360 (1964), IBM PC (1980)



UNIVAC 1 (1951) First Commercially Available Computer



Hitos de la tercera generación

● Interactividad con el usuario:

- Multitud de terminales compiten por el uso de la CPU, alojando sus programas en memoria y disco.
- La consola del usuario (shell) permite al sistema establecer un diálogo con el usuario a través de comandos y respuestas.

● Productividad del sistema:

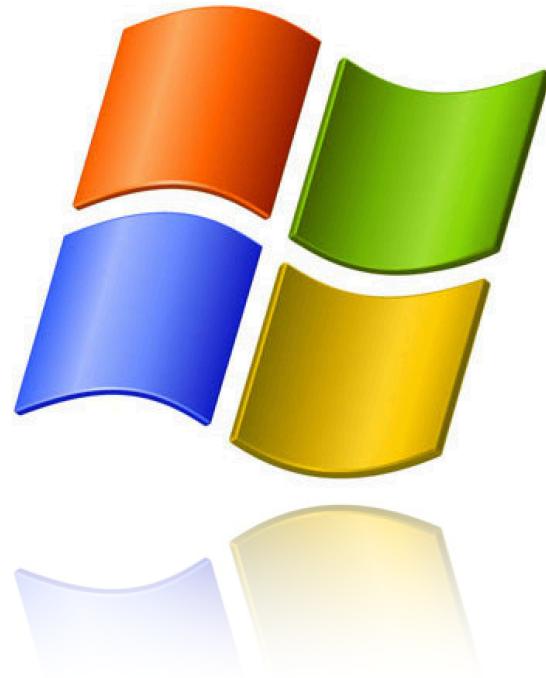
- Compartición de recursos entre usuarios (CPU, memoria, disco, ...).
- Multiprogramación, multitarea, memoria virtual, *swapping* a disco.
- Planificación de la CPU.
- Cada usuario cree disponer de todo el sistema para él.

● Protección:

- Entre aplicaciones, usuarios y dispositivos.

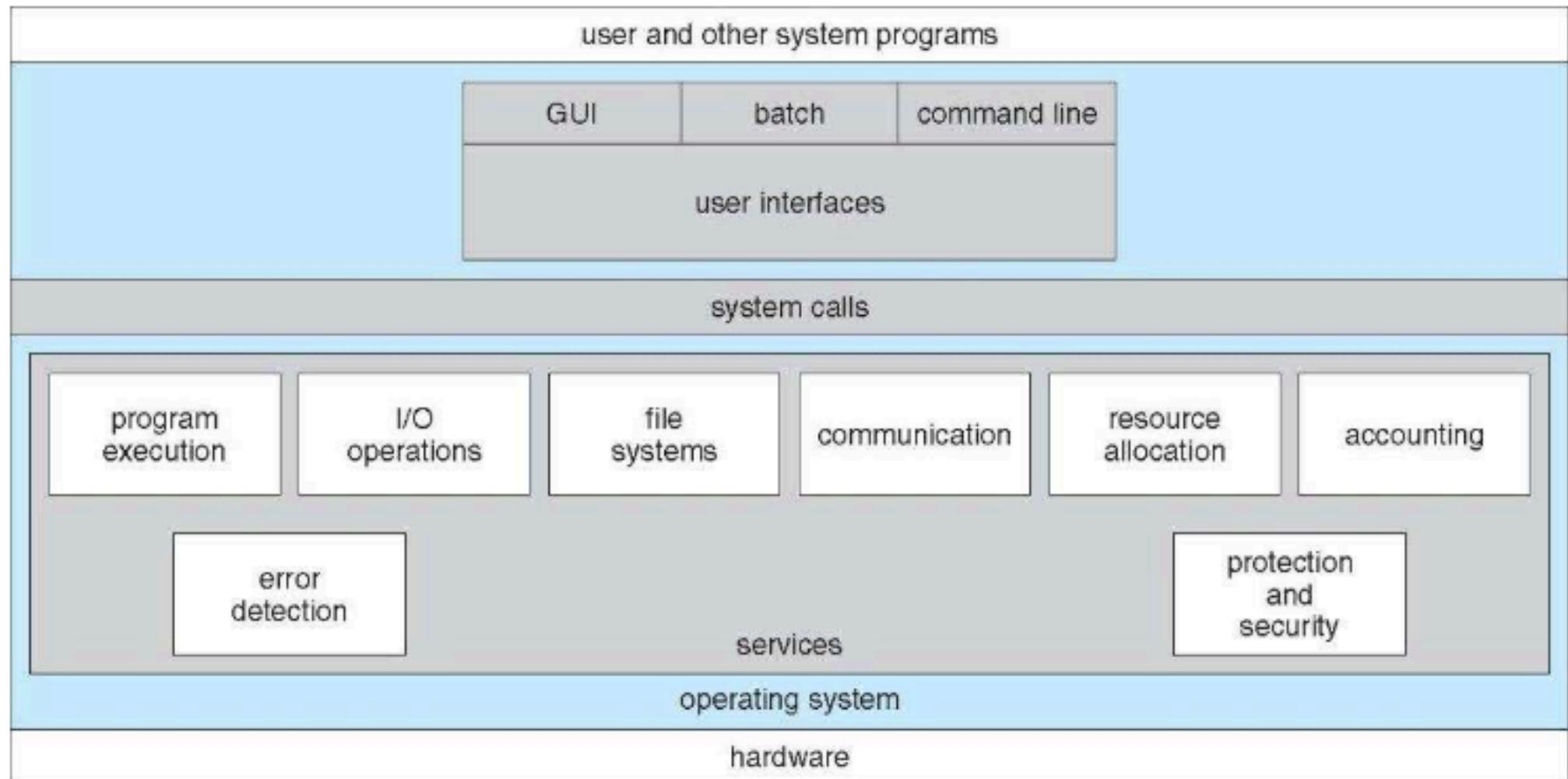
Hitos de la cuarta generación

- Gran reducción de costes, entrada del PC en el hogar.
- Popularidad del sistema operativo. Muchas variantes:
 - Distribuidos (Mach, Amoeba).
 - Middleware (Corba - OMG, DCOM - Microsoft).
 - Multiprocesador (SMP - Symmetric MultiProcessing).
 - Multithread (Linux, Windows).
 - En tiempo real: Sin disco ni memoria virtual. S.O. en ROM.
- El hardware agrupa a legiones de usuarios bipolares:
 - Intel/Motorola (CPUs 80's), Intel/AMD (CPUs 90's).
 - Nvidia/ATI (GPUs 00's), Nvidia/AMD (GPUs 10's).



III. Funciones y Servicios

Vista global de los servicios del S.O.



Tres vertientes de las funciones y servicios del S.O.

● Gestión de recursos:

- CPU (planificación - tema 2).
- Memoria (paginada, segmentada y virtual - tema 3).
- Dispositivos de entrada/salida (sistemas de ficheros - tema 4).

● Máquina abstracta:

- Los servicios del S.O. se articulan a través de llamadas al sistema o de un *API (Application Program Interface)*.

● Interfaz de usuario. Dos alternativas básicas:

- A través de la línea de comandos de un *shell (CLI - Command Line Interface)*.
- Mediante ventanas e iconos (*GUI - Graphics User Interface*).

1. Gestión de recursos

- Controla el acceso a todos los recursos compartidos.

- Físicos: CPU, memoria, E/S.
- Lógicos: Ficheros, puertos de comunicaciones, interrupciones.

- Gestión espacial:

- Alojamiento/liberación en memoria y disco.

- Gestión temporal:

- Uso eficiente y justo de la CPU.
- Planificación de las transferencias a memoria y disco.
- Respuesta en tiempo acotado.

- Monitorización:

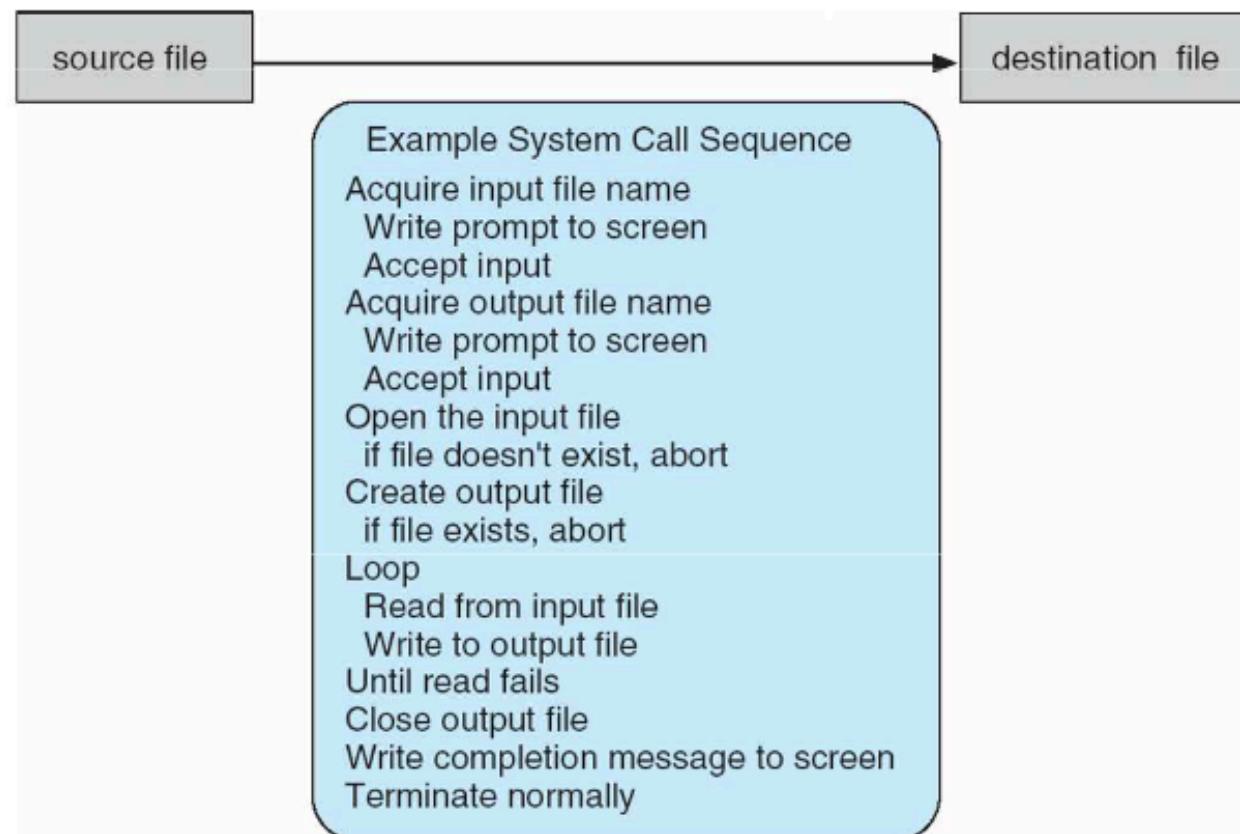
- Lleva un registro de la cantidad de recursos usados y su % de uso.

- Protección/seguridad.

2. Máquina abstracta

- El S.O. impide que el usuario vea el hardware y su complejidad, aportando un API común a aplicaciones y servicios, lo que simplifica la programación.
- Las llamadas al sistema proporcionan el interfaz entre los programas en ejecución y el S.O.
 - Suelen ser rutinas escritas en lenguaje C o ensamblador.
 - Dotan de portabilidad al software, sobre todo si nos apoyamos en el API como interfaz de alto nivel que haga las llamadas al sistema por nosotros.
- APIs más populares:
 - Win32 (Windows).
 - POSIX (UNIX/Linux/MacOS).
 - JVM (Java Virtual Machine).

Ejemplo de secuencia de llamadas al sistema



Ejemplo de 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)
```

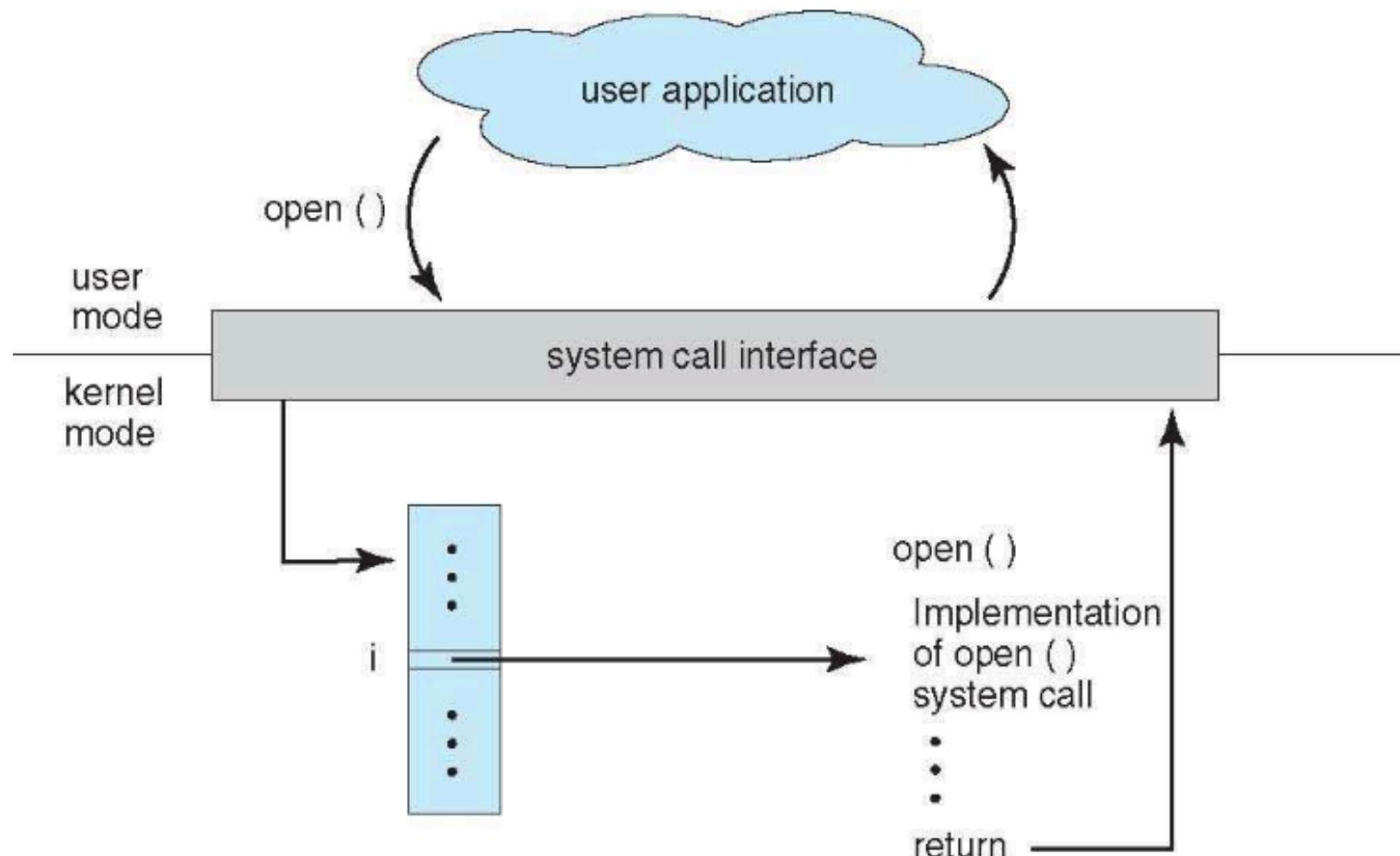
return value	function name	parameters
--------------	---------------	------------

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

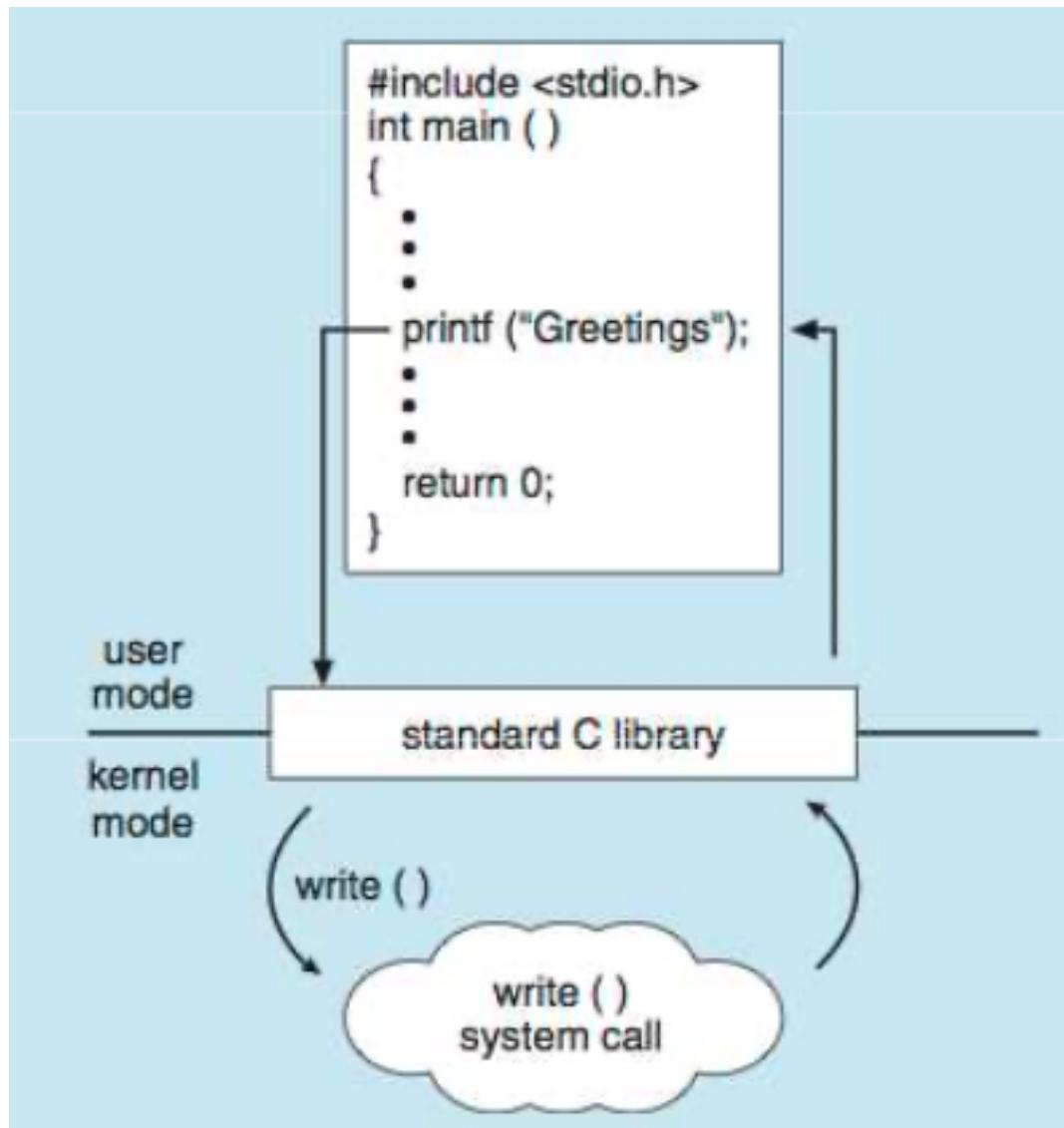
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns -1.

La llamada al sistema y su relación con el S.O.





Ejemplo de uso de una llamada al sistema desde una librería de C



Tipos de llamadas al sistema

• Gestión de ficheros

- Crear/borrar, abrir/cerrar, leer/escribir, establecer/consultar atributos.

• Gestión de dispositivos

- Solicitar/liberar, leer/escribir, establecer/consultar atributos, ligar/desligar dispositivos lógicos.

• Mantenimiento de información

- Establecer/consultar hora y fecha, datos del sistema o atributos de ficheros y procesos.

• Comunicaciones

- Crear/borrar conexiones, enviar/recibir mensajes, ligar/desligar dispositivos remotos.

• Protección

- Controlar accesos a recursos/usuarios, establecer/consultar permisos.

3. Interfaz de usuario

- Establece las vías de comunicación entre usuario y S.O.
Se implementa de dos formas básicas:

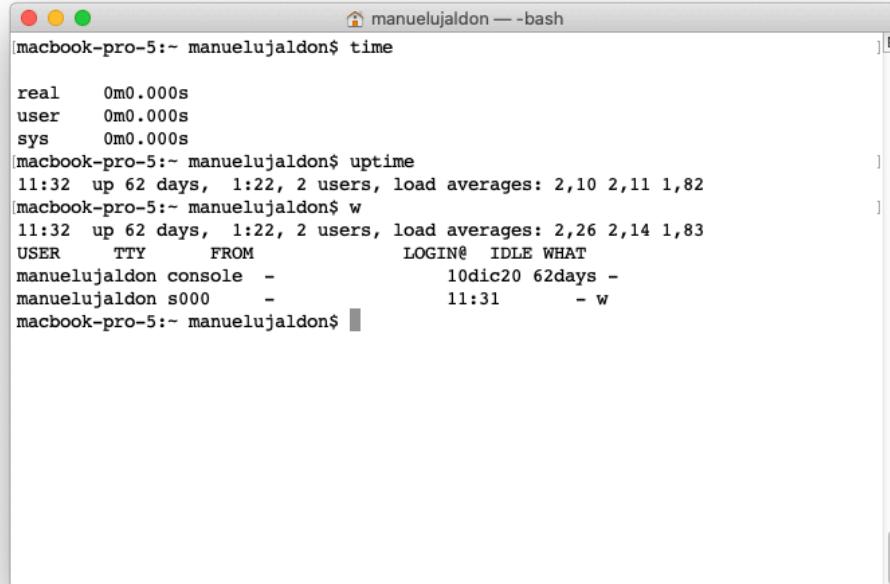
- **CLI (Command Line Interface).** A través de un *shell* o intérprete de comandos, en el que los comandos se teclean en un terminal que se muestra por pantalla.

- En la segunda parte de la asignatura implementaremos nuestro *Shell* propio a partir del básico de Linux, que es también el que usa MacOS.
- La variante de Windows se ha heredado de MS-DOS y es menos versátil.

- **GUI (Graphics User Interface).** Más amigable e intuitivo.

- MacOS fue pionero con su entorno de ventanas e iconos.
- Posteriormente clonado por Windows.
- Linux ha desarrollado variantes propias como GNOME.
- Poco a poco el ratón va dejando paso a las pantallas táctiles a las que nos han acostumbrado los teléfonos móviles y tabletas.

Los 3 típicos ejemplos: CLI, GUI con ratón y GUI con pantalla táctil



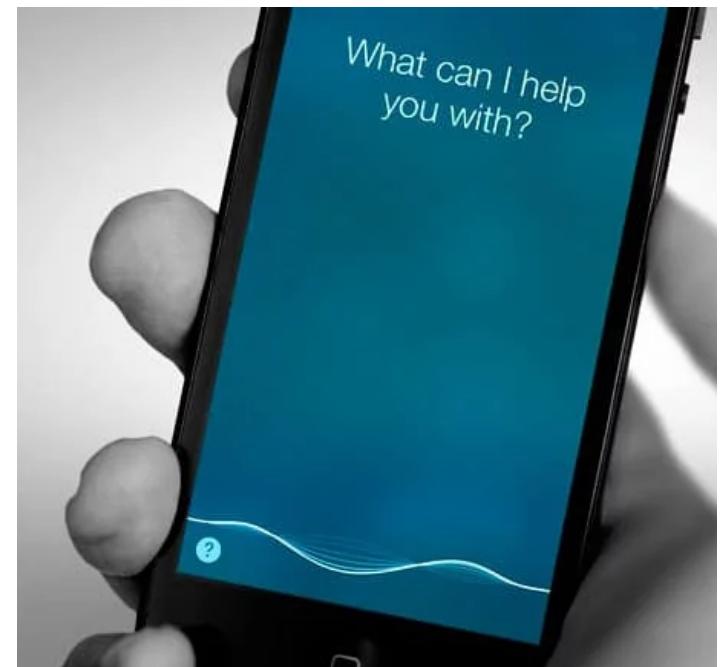
```
manuelujaldon@macbook-pro-5:~ manuelujaldon$ time
real    0m0.000s
user    0m0.000s
sys     0m0.000s
manuelujaldon@macbook-pro-5:~ manuelujaldon$ uptime
11:32 up 62 days,  1:22, 2 users, load averages: 2,10 2,11 1,82
manuelujaldon@macbook-pro-5:~ manuelujaldon$ w
11:32 up 62 days,  1:22, 2 users, load averages: 2,26 2,14 1,83
USER   TTY      FROM             LOGIN@ IDLE WHAT
manuelujaldon console -          10dic20 62days -
manuelujaldon s000   -           11:31      - w
manuelujaldon@macbook-pro-5:~ manuelujaldon$
```

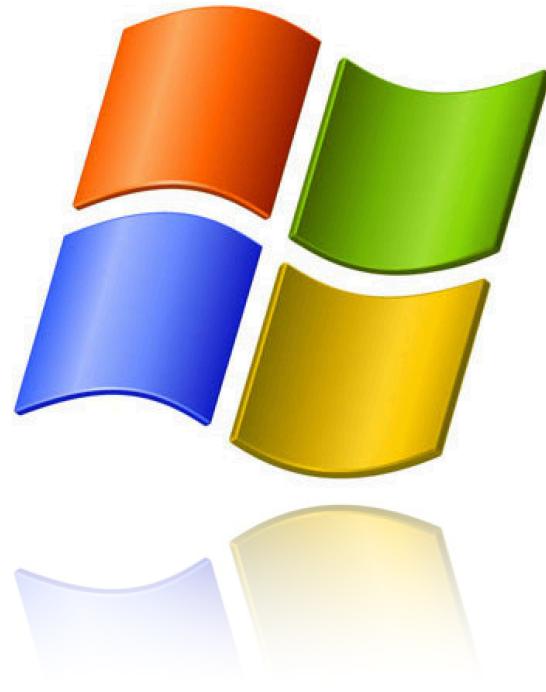




El interfaz más moderno: Asistentes de voz basados en NLP (Natural Language Processing)

- Google Assistant, Amazon Alexa, Apple Siri.





IV. Estructura e Implementación

Cómo ha evolucionado la implementación

- En los primeros años, se hacía en lenguaje ensamblador.
- De ahí pasaron a usarse lenguajes como Algol o PL/1.
- Ahora casi todo se hace en C o C++, aunque la parte de más bajo nivel puede delegarse a ensamblador y la de más alto nivel a Python.
- Se busca un compromiso entre mayor portabilidad y mayor velocidad.

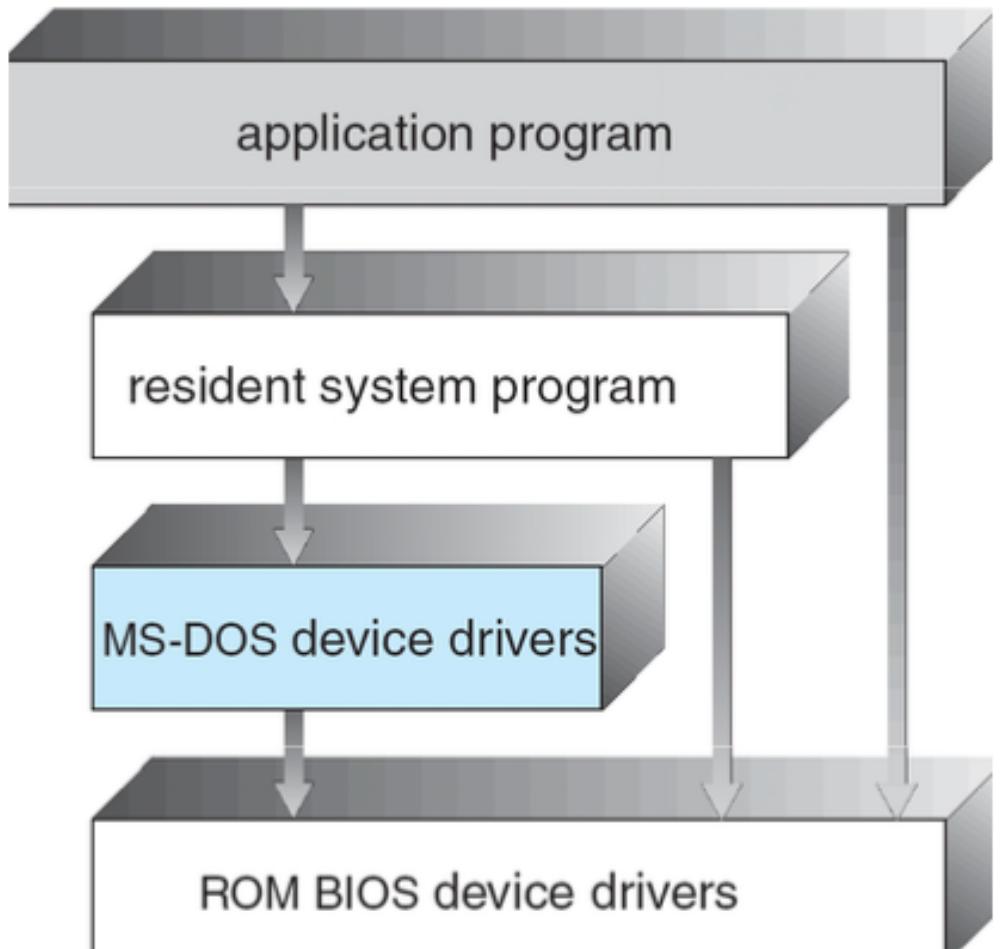
Estructura del S.O.

- El S.O. es un programa muy extenso que requiere una buena estructuración:
 - Aproximación monolítica:
 - Estructura sencilla como MS-DOS.
 - Estructura más compleja como UNIX.
 - Aproximación modular o por capas:
 - Microkernel como Mach.

Ejemplo: MS-DOS

● Pensado para maximizar funcionalidad minimizando el espacio.

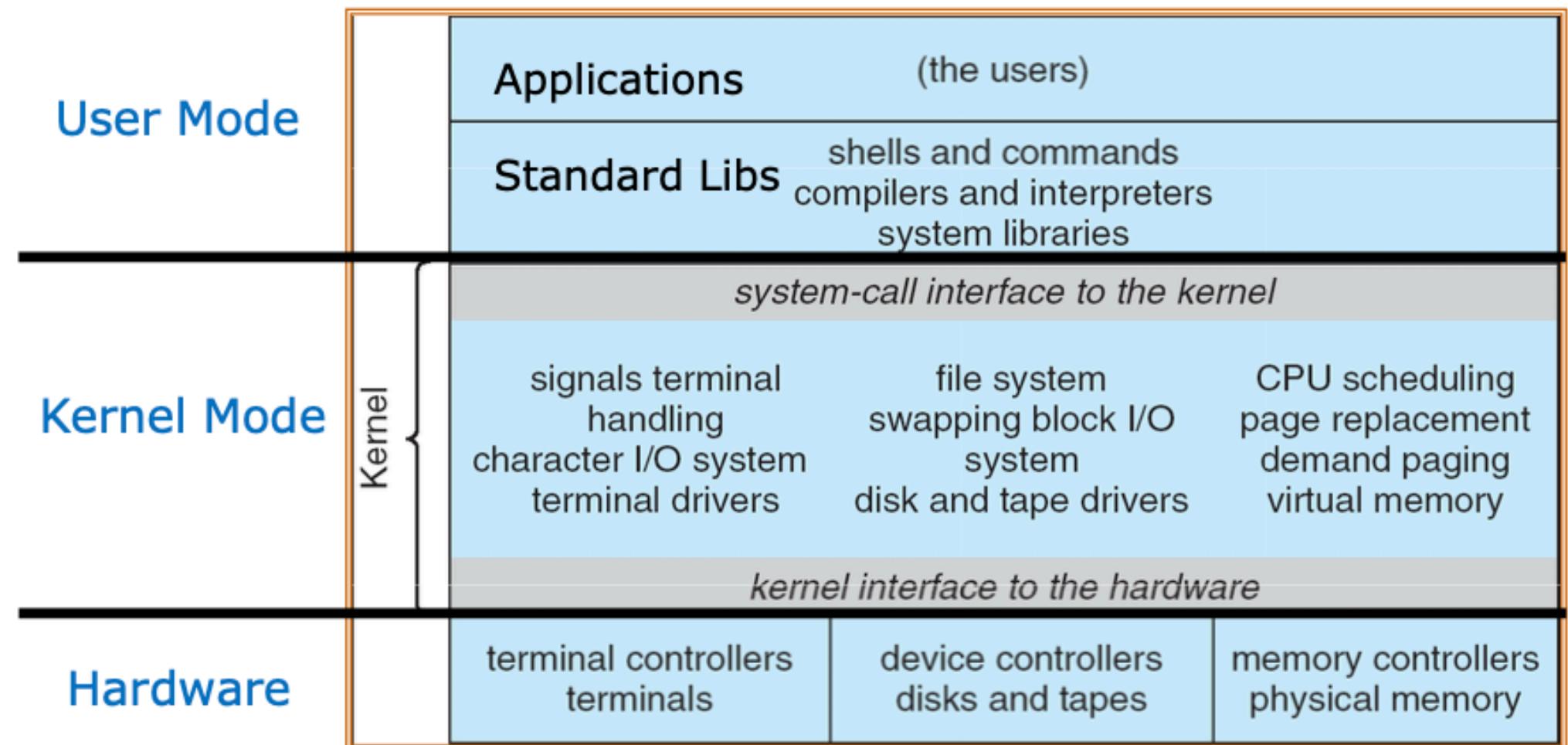
- Modularidad deficiente, interfaces y capas no están bien delimitadas.
- Diseñado sin imaginar el impacto que tendría.
- Inconsistente cuando falla el programa de usuario.
- Escrito para la CPU 8088 de Intel, que carece de modo dual y mecanismos de protección hardware.



Ejemplo: UNIX

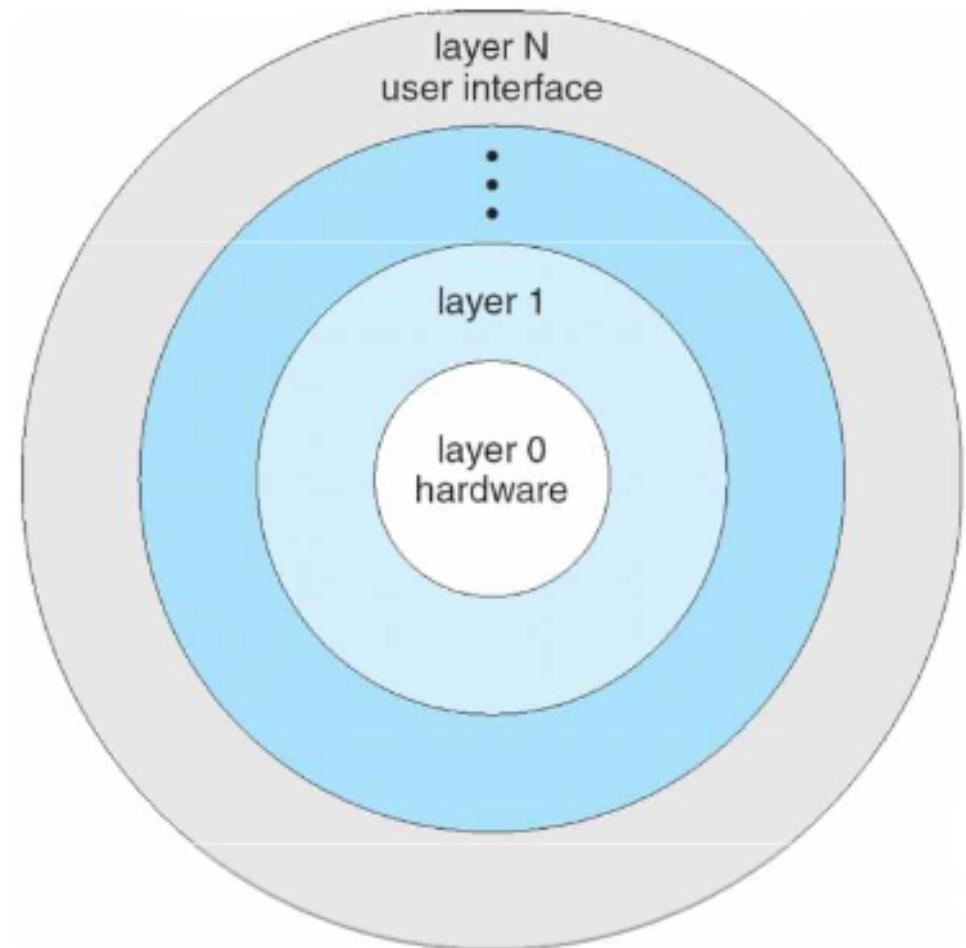
- Su primera versión estaba poco estructurada debido a las limitaciones del hardware. Contempla dos partes:
 - Los programas de usuario.
 - El *kernel* o núcleo central que alberga los *drivers* e interfaces, y que al no estar subestructurado dificulta las acciones de mejora.
- El kernel contiene todo bajo el interfaz que delimitan las llamadas al sistema, y se adentra hasta el límite con el HW:
 - La planificación del uso de la CPU (que estudiaremos en el tema 2).
 - La gestión de la memoria (tema 3).
 - El sistema de ficheros (tema 4).
 - Otras funciones más satélite del S.O.
- Toda esta amalgama de funciones en una sola capa ha lastrado su posterior mejora.

Estructura interna del S.O. UNIX



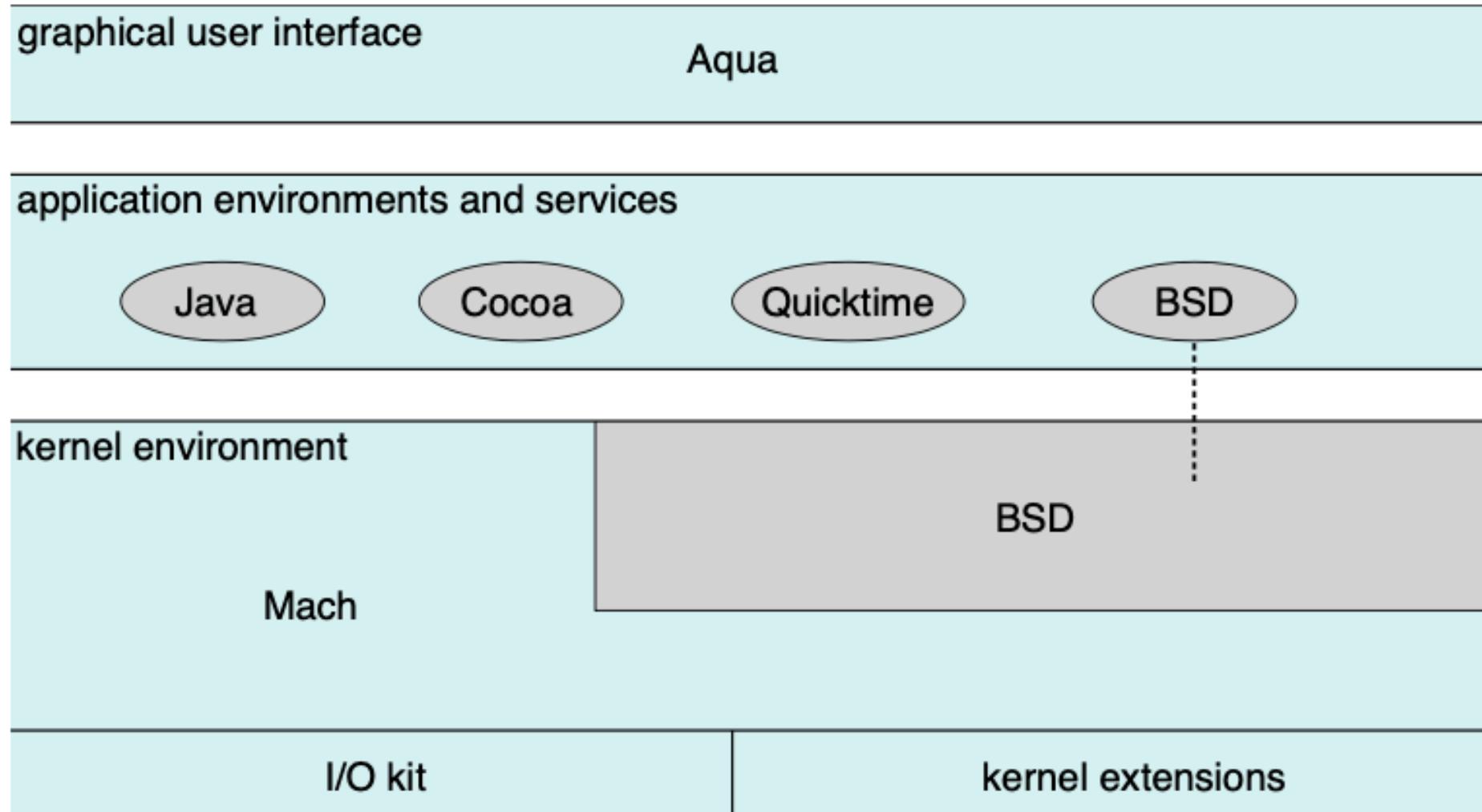
Aproximación modular

- El S.O. se divide en niveles, comenzando con el interfaz de usuario ubicado en el nivel más alto y terminando con el hardware en el nivel más bajo.
- Cada nivel se cimenta a partir del anterior, apoyándose en sus funciones/servicios y mejorando/ampliando éstos hacia el siguiente nivel.



Ejemplo: MacOS

- Apuesta por una estructura híbrida, más modular que UNIX.



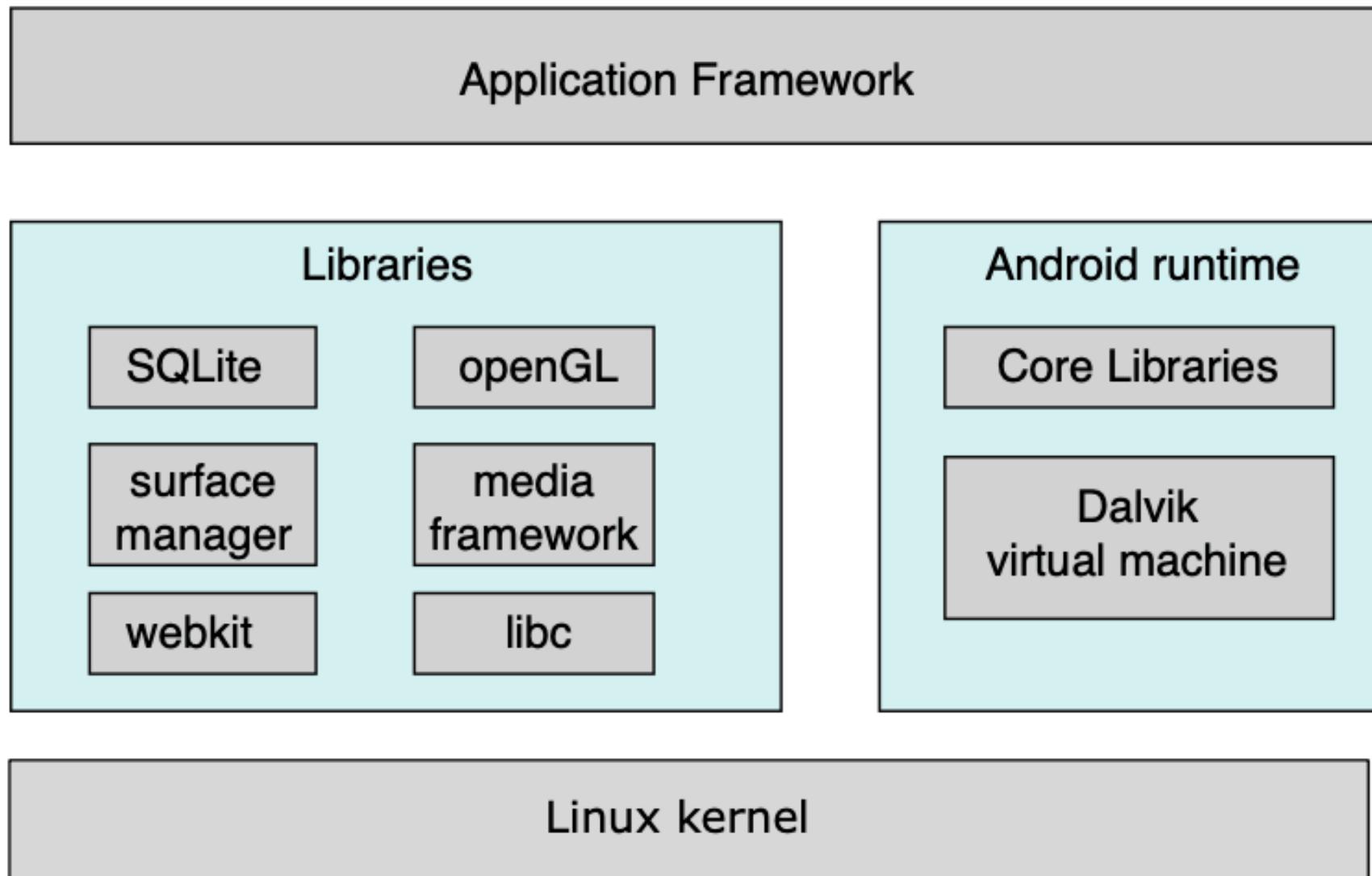
Ejemplo: iOS

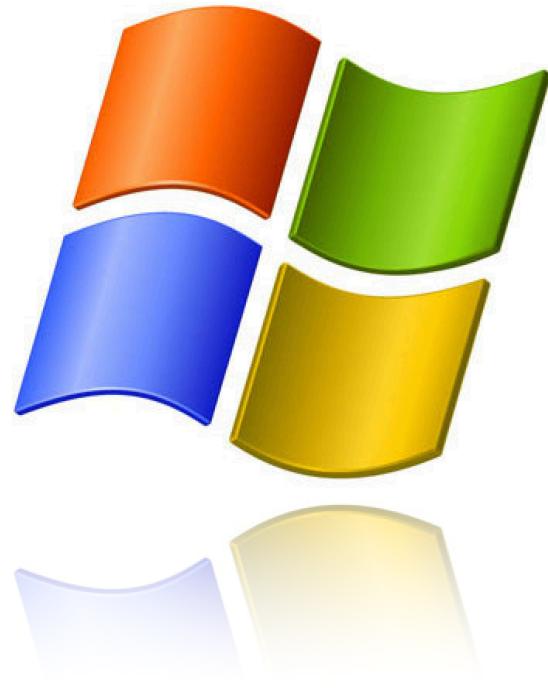
- Es el S.O. para dispositivos móviles de Apple (iPhone iPad).
- Se estructura en torno a MacOS, con funcionalidad añadida.
- No corre las aplicaciones de MacOS nativamente, pero sí se adapta a distintas arquitecturas de CPU (ARM e Intel).
- De abajo a arriba, aporta las siguientes capas, cada una sobre la anterior:
 - **Core OS:** El *kernel* del S.O., basado en el de MacOS.
 - **Core Services:** Proporciona los servicios para computación en la nube y bases de datos.
 - **Media Services:** La capa para gráficos, audio y video.
 - **Cocoa Touch:** Una API para desarrollar apps a través del lenguaje Objective-C (extensión de C para programación orientada a objetos).

Ejemplo: Android

- Desarrollado por Open Handset Alliance (auspiciado por Google).
 - De código abierto.
- La pila de capas es similar a iOS, aunque está basado en el kernel del Linux y modificado a partir de éste.
 - Gestiona los procesos, la memoria y los drivers de los dispositivos, incorporando la gestión eficiente del consumo.
 - El soporte en tiempo de ejecución incluye un conjunto de librerías esenciales y la máquina virtual Dalvik.
 - Las apps se desarrollan en Java junto con el API de Android. Los ficheros de clase de Java se compilan para generar bytecode, que luego se traduce al ejecutable que corre sobre Dalvik.
 - Las librerías incluyen entornos de desarrollo para navegación Web (webkit), bases de datos (SQLite), y multimedia.

Arquitectura de Android





V. Soporte y Protección Hardware

Aportaciones

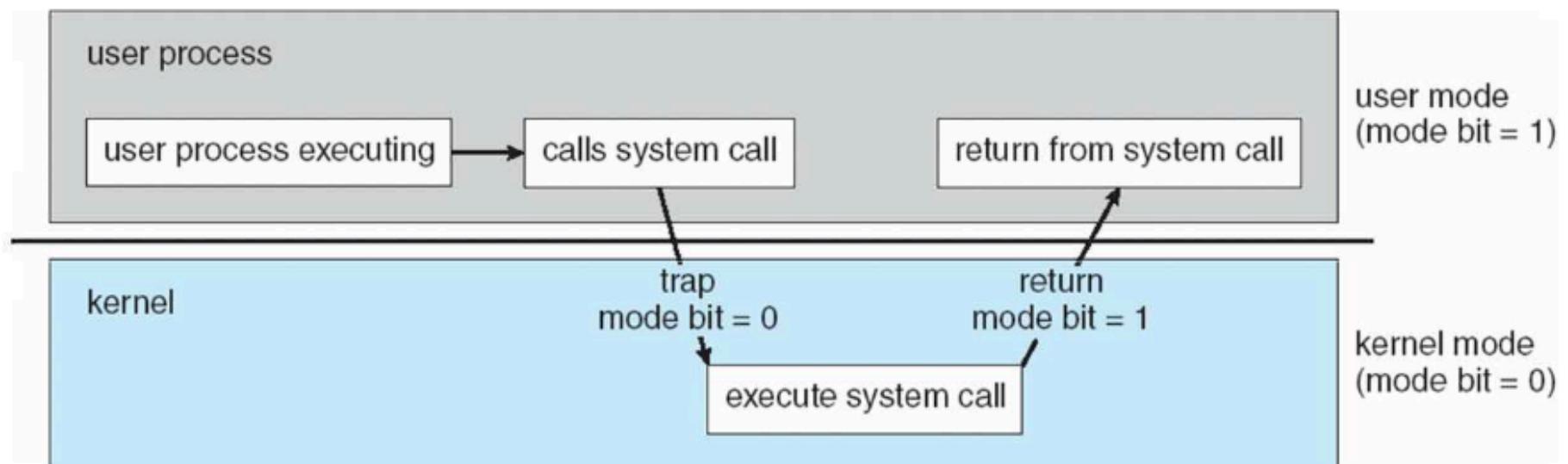
- Para algunas funciones del S.O. es necesario el concurso del hardware, ya sea por velocidad o por protección (contar con sus mecanismos de seguridad).
- Algunos riesgos latentes:
 - Algún proceso cae en un bucle infinito.
 - Procesos que se modifican entre sí.
 - Identificar la congestión en el uso de recursos.
- Mecanismos de protección y soporte hardware:
 - Modo dual de operación.
 - Protección de la CPU.
 - Interrupciones.
- Pasamos a verlos a continuación...

Modo dual de operación

- La compartición de recursos exige monitorizar la actividad para que un programa incorrecto no afecte al resto.
- Una solución consiste en diferenciar entre modos de operación:
 - Modo usuario: La ejecución se personaliza para él.
 - Modo kernel: La ejecución transcurre en nombre del S.O.
- Los primeros S.O. como MS-DOS y las primeras CPUs como el 8088 de Intel no disponían de estos instrumentos, y como consecuencia de ello, los usuarios campaban a sus anchas por el sistema y los programas podían colisionar en el acceso a los dispositivos.

Modo dual de operación (2)

- Se incorpora un bit por hardware para reflejar el modo: kernel (0) o usuario (1).
- Cuando el servicio de una interrupción o llamada al sistema necesita utilizar algún recurso crítico, solicita la comutación al modo kernel.



Protección de la CPU

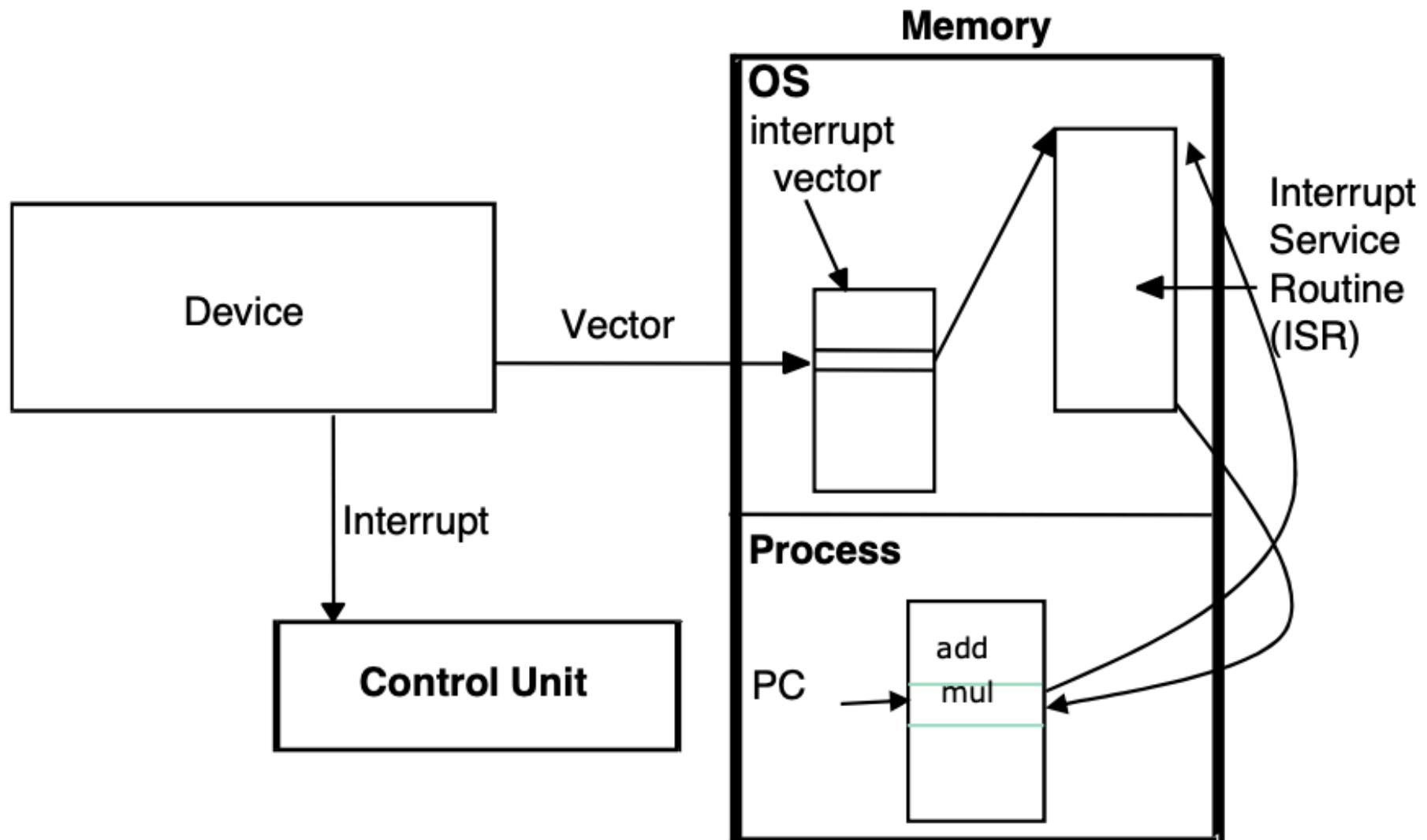
- El S.O. concede un tiempo al proceso de usuario, que al expirar dispara una interrupción que devuelve el control al S.O.
- Es un mecanismo que implementa el tiempo compartido en sistemas multiprogramación, y en los cambios de contexto el S.O. aprovecha para realizar ciertas labores de mantenimiento:
 - Desalojar memoria.
 - Inicializar metadatos.
 - Registrar el uso de los recursos.

Interrupciones

- Hay dos tipos básicos:

- Hardware: Permiten que el software responda ante eventos hardware.
 - Ejemplos: Pulsamos una tecla o movemos el ratón y queremos programar la acción pertinente en su *driver*.
- Software: Se disparan intencionadamente para realizar alguna función de manera asíncrona o por algún error.
 - Ejemplos: Paginación a disco o división por cero.
- Son un mecanismo muy eficiente, al permitir que los dispositivos de entrada/salida puedan trabajar en paralelo con la CPU, avisándole sólo cuando necesiten su concurso.
- Ya desde los primeros PCs, han sido un instrumento fundamental para armonizar software y hardware.

El mecanismo de interrupción



Bibliografía

- Se recomienda la lectura del libro que colocamos en la portada de la asignatura en el Campus Virtual de la UMA:
 - Operating Systems Concepts, 8th Edition, de los autores A. Silberschatz, G. Gagne, P.B. Galvin, publicado por Wiley en 2008.
- Sus contenidos están disponibles íntegramente para los estudiantes de la UMA, a través del enlace Web:
 - <https://www.oreilly.com/library/view/operating-system-concepts/9780470128725/?ar>
- En particular, este primer capítulo del temario de S.O. está cubierto por los dos primeros temas del libro, que componen su primera parte titulada “Overview”:
 - Chapter 1: Introduction
 - Chapter 2: Operating System Structures