Sistemas Operativos: T1: Introducción a los S.O.

1. Concepto de S.O.

-La **circuitería** (Hardware) proporciona los recursos básicos de computación (Ej. CPU, GPU, memoria, etc.)

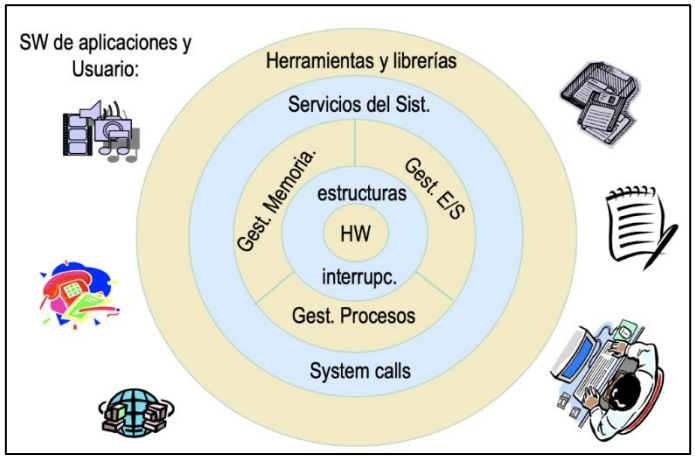
-El **sistema operativo** controla y coordina el uso del hardware por parte de las aplicaciones y los usuarios (Ej. UNIX/Linux, MacOS, Windows, etc.)

-Las **aplicaciones** establecen la forma de usar los recursos para resolver los problemas computacionales (Ej. Compiladores, bases de datos, navegadores, etc.)

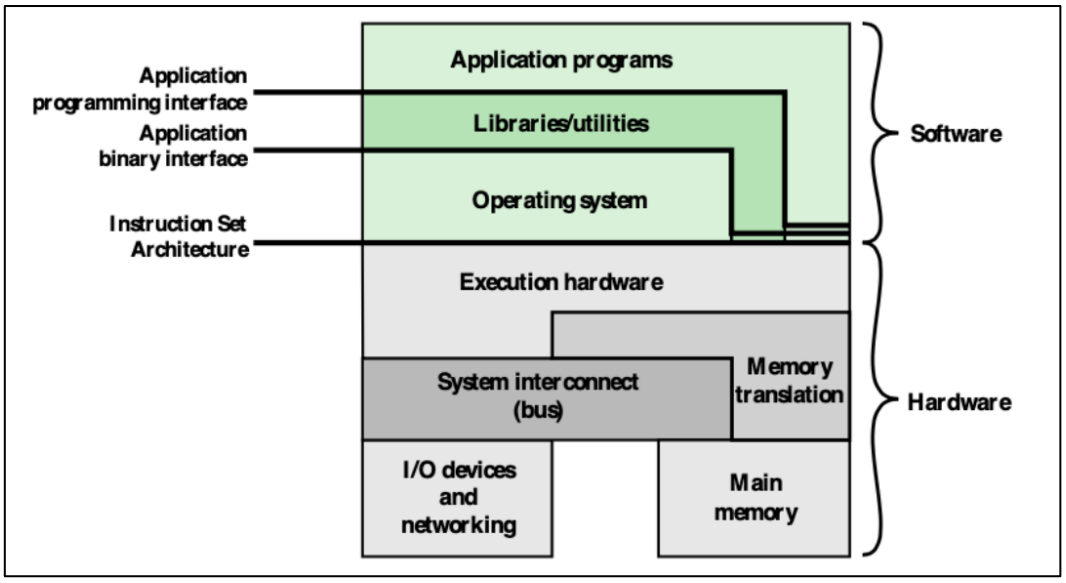
-Es un **programa** que hace de **intermediario** entre el **usuario** y la **circuitería**, es decir, un conjunto de programas implementados como software o firmware que facilitan al usuario el uso de la circuitería.

-Posee un **repertorio** de **funciones** encaminadas a simplificar la gestión y el uso del computador de forma segura y eficiente, además **gestiona** los **recursos** de la máquina (CPU, memoria, discos, comunicaciones, …).

-**Thrashing**: se produce cuando el S.O. dedica más tiempo a gestionar los recursos que a ejecutar lo que le demanda el usuario.



1.1. Estructura del sistema de computación



1.2. Objetivos del Sistema Operativo

Son básicamente tres objetivos:

-**Ejecutar** los **programas** del usuario.

-**Facilitar** el **uso** del **equipo**.

-**Utilizar** los **recursos** de una forma **eficiente**.

Desde el punto de vista del usuario debe ser cómodo de usar, fácil de aprender, fiable, seguro y rápido.

Desde una perspectiva del sistema debe ser fácil de diseñar, implementar y mantener, además de flexible, fiable y eficiente.

2. Evolución histórica

La podemos resumir básicamente en una etapa pre-electrónica y en cuatro generaciones:

	Marco temporal	Tecnología predominante	Ejemplos	Personajes ilustres
Pre-electrónica	Antes de 1945	Accionamiento mecánico, bastantes fiascos	*Analytical machine*	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)

Los **hitos de la tercera generación** fueron:

- **Interactividad** con el **usuario**: multitud de terminales compiten por el uso de la CPU, alojando sus programas en memoria y disco y 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.

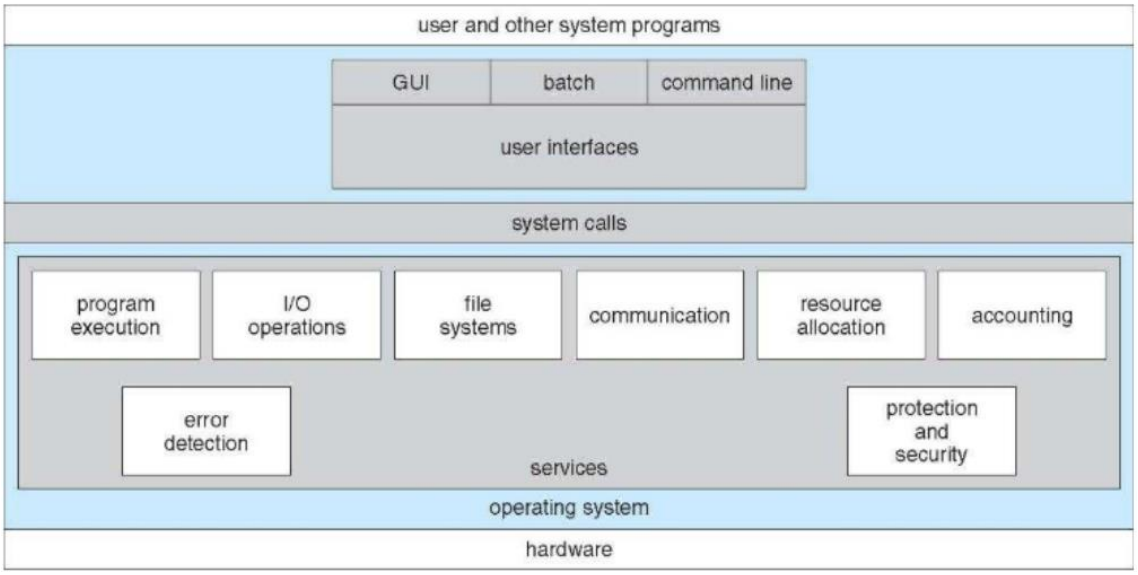
Los **hitos de la cuarta generación** fueron.

- 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.
- **Usuarios bipolares** (competitividad entre dos principales fabricantes):
  - Intel/Motorola (CPUs 80's), Intel/AMD (CPUs 90's), Nvidia/ATI (GPUs 00's) y Nvidia/AMD (GPUs 10's).

3. Funciones y servicios

En primer lugar presentemos una visión global de los servicios del S.O:



3.1. Gestión de recursos

-Controla el **acceso** a todos los **recursos compartidos**:

- **Físicos**: CPU, memoria, etc.
- **Lógicos**: ficheros, puertos de comunicaciones e interrupciones.

-**Gestión espacial**: alojamiento/liberación de memoria en disco.

-**Gestión temporal**: Uso eficiente y justo de la CPU, planificación de las transferencias a memoria y disco y respuesta en tiempo acotado.

-**Monitorización**: registro de la cantidad de recursos usados y su % de uso.

-**Protección** y **seguridad**.

3.2. Máquina abstracta

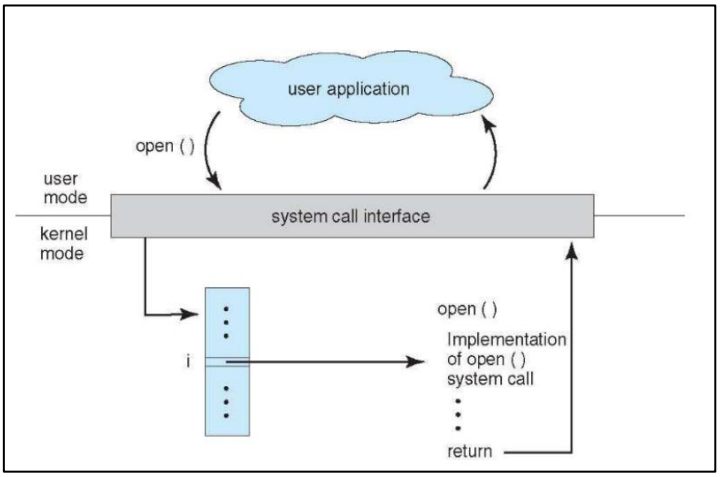
-**API (Application Programming Interface):** repertorio de comandos y procedimientos disponibles que permite integrar sistemas.

-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 en ensamblador.
- Dan portabilidad al software, apoyándonos en la API como interfaz de alto nivel que haga las llamadas al sistema por nosotros.
- API populares: Win32 (Windows), POSIX (Unix, Linux y MacOS) y JVM (Java Virtual Machine).

-**Resumen** de las **llamadas** al **sistema** y su relación con el S.O:



-**Tipos** de **llamada** al **sistema**

- Gestión de ficheros: crear/borrar, abrir/cerrar, leer/escribir, etc.
- Gestión de dispositivos: solicitar/liberar, leer/escribir, establecer/consultar atributos.
- 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.

3.3. 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. Nosotros implementaremos un Shell propio a partir del básico de Linux que es el mismo que en MacOS, el de Windows está basado en MS-DOS y es menos versátil.

- **GUI (Graphics User Interface):** más amigable e intuitivo. MacOS fue pionero, más tarde clonado por Windows y Linux ha sacado sus propias versiones también (GNOME). Poco a poco el ratón va dando paso a las pantallas táctiles. (Ej. CLI: terminal, GUI con ratón: MacOS o GUI con pantalla táctil: iPhone).

-El **interfaz** más moderno son los **asistentes** de **voz** basados en **NLP** (Natural Language Processing).

4. Estructura e implementación

-Inicialmente la implementación se hacía en **ensamblador**, de ahí pasó a hacerse en **Algol** o **PL/1** y actualmente se hace en **C/C++**, aunque la de más bajo nivel puede delegarse a ensamblador y las de más alto nivel en Python.

-Se busca un **compromiso** entre mayor **portabilidad** y **velocidad**.

4.1. Estructura del S.O.

-El **S.O.** es un **programa** muy **extenso** que requiere una **buena estructuración**.

-Hay **dos** implementaciones **posibles**: **monolítica** (estructura sencilla como MS-DOS o más compleja como UNIX) y **modular** o **por capas** (Microkernel como Mach.)

4.2. Aproximación monolítica

4.2.1. MS-DOS

-Pensado para **maximizar funcionalidad minimizando** el **espacio**. Posee una modularidad deficiente, las interfaces y capas no están bien delimitado.

-**Diseñado sin imaginar** el **impacto** que tendría, es inconsistente cuando falla el programa de usuario.

-**Escrito** para la **CPU 8088 de Intel**, que **carece** de **modo dual** y **mecanismos** de **protección** del hardware.

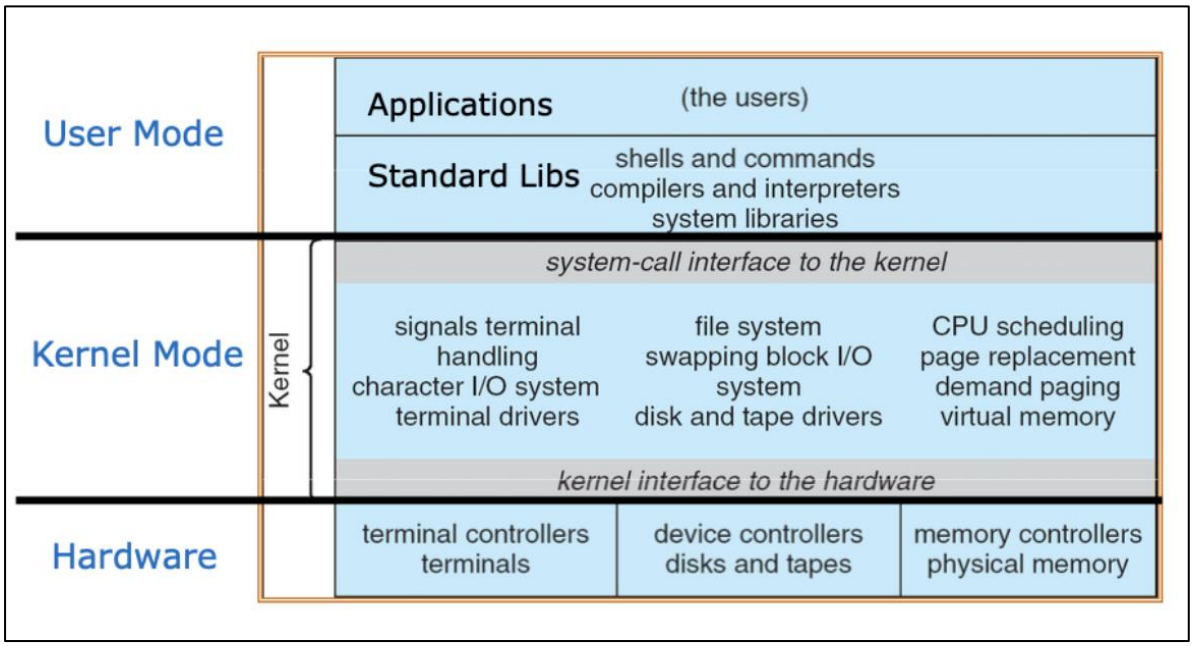
4.2.2. UNIX

-Su **primera versión** estaba **poco estructurada** debido a las **limitaciones** del **hardware**. Contempla **dos partes**:

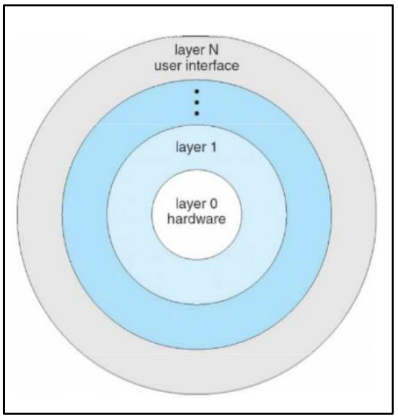
- Los **programas de usuario**.
- El **kernel o núcleo central** que alberga los drivers e interfaces: al no estar subestructurado dificulta las acciones de mejora.

-El **kernel** contiene **todo bajo** el **interfaz** que **delimitan** las **llamadas** a **sistema** y se adentra hasta el **límite** con el **hardware**.

-Toda esta **mezcla** de **funciones** en una **sola capa** ha **lastrado** su posterior **mejora**.



4.3. Aproximación por capas o modular



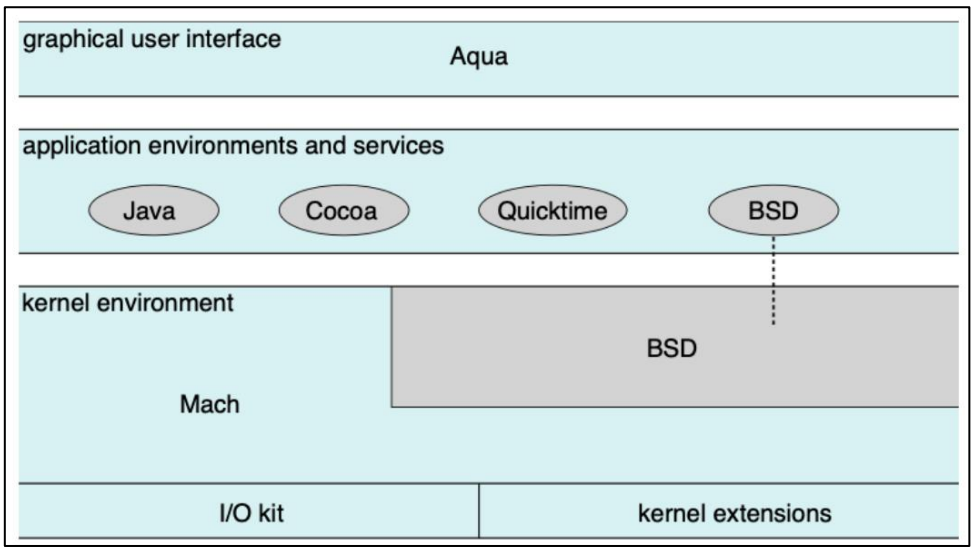
-El **S.O.** se divide en **niveles**, comenzando con el **interfaz** de **usuario** ubicado en el **nivel más alto** y **terminando** con el **hardware** en el **nivel más bajo**.

-**Cada nivel** se **cimenta** a partir del **anterior**, apoyándose en sus funciones/servicios y mejorando/ampliando éstos hacia el siguiente nivel.

-Veamos tres ejemplos: MacOS, iOS y Android, que utilizan este enfoque.

4.3.1. MacOS

-Apuesta por una **estructura híbrida**, más **modular** que UNIX:



4.3.2. iOS

-Es el **S.O.** para dispositivos **móviles** de **Apple** (iPhone iPad). Se **estructura** en torno a **MacOS**, con **funcionalidad añadida**, aunque no corre las aplicaciones de MacOS nativamente, sí se **adapta** a **distintas arquitecturas** de **CPU** (ARM e Intel).

-De abajo a arriba, aporta las **siguientes capas**, cada una sobre la anterior:

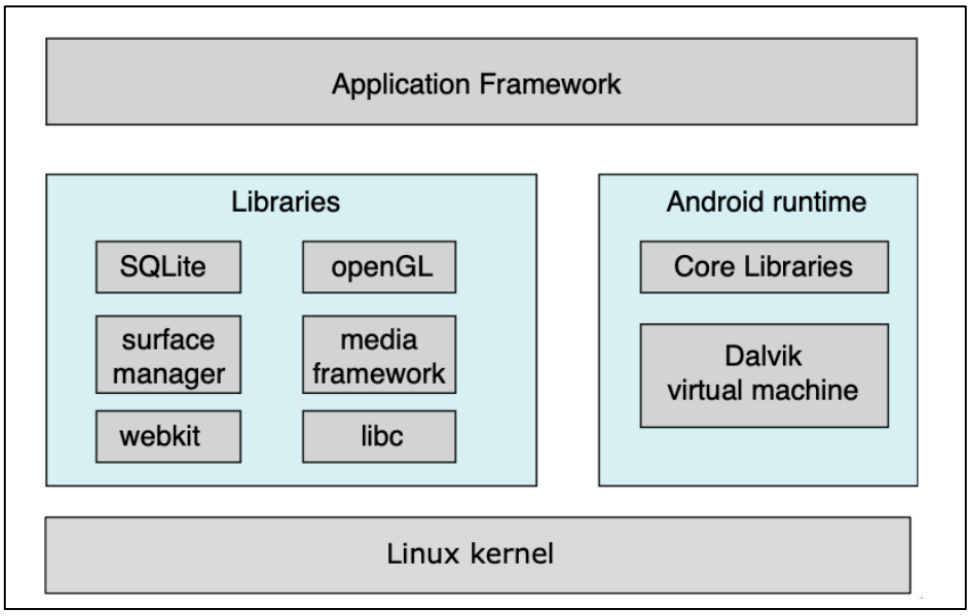
- **Core OS**: El kernel del S.O., basado en el de MacOS.
- **Core Services**: Proporciona los servicios para computación en la nube y bases de datos.
- **Media Services**: La capa para gráficos, audio y video.
- **Cocoa Touch**: Una API para desarrollar apps a través del lenguaje Objective-C (extensión de C para programación orientada a objetos).

4.3.3. Android

-**Desarrollado** por **Open Handset Alliance** (auspiciado por Google): **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 .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**.



5. Soporte y Protección Hardware

5.1. 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).

-**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.

5.2. Modo dual de operación

-La **compartición** de **recursos** exige **monitorizar** la **actividad** para que un programa incorrecto no afecte al resto.

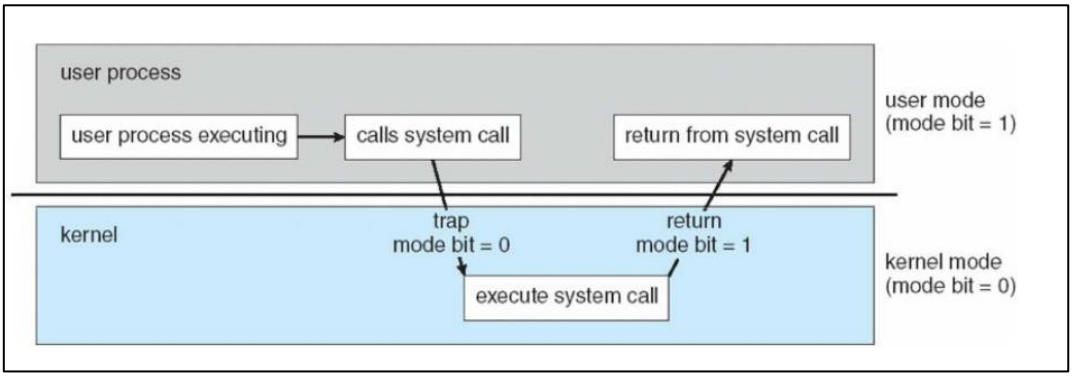
-Una **solución** consiste en diferenciar entre **modos** de **operación**:

- **Modo usuario**: La ejecución se personaliza para él.
- **Modo kernel**: La ejecución transcurre en nombre del S.O.

-Los primeros S.O. y las primeras CPUs como el 8088 de Intel no poseían estos mecanismos y los usuarios campaban a sus anchas por el sistema.

-Se **incorpora** 1 **bit** por **hardware** para reflejar el **modo**: kernel (0) / usuario (1).

-Cuando el **servicio** de una **interrupción** o **llamada** al **sistema** necesita utilizar algún recurso crítico, **solicita** la **conmutación** al **modo kernel**.



5.3. 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.

5.4. Interrupciones

-Existen dos tipos básicos:

- **Hardware**: Permiten que el software responda ante eventos hardware. (Ej. pulsar una tecla o mover 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. (Ej. paginación a disco o división por cero).

-Son un **mecanismo** muy **eficiente** ya que permite que los **dispositivos** de **E/S puedan trabajar** en **paralelo** con la **CPU**, avisándole solamente cuando necesiten su concurso.

6. Bibliografía

-Capítulo 1 y 2 (Introduction y Operating System Structures) de *Operating Systems Concepts* de A. Silberschatz, G. Gagne, P.B. Galvin.