

# **INSTITUTO POLITÉCNICO NACIONAL**

## **ESCUELA SUPERIOR DE CÓMPUTO**

### **INGENIERÍA EN SISTEMAS COMPUTACIONALES**

Academia de Sistemas Distribuidos

Guía de estudio de Sistemas Operativos

Profesor: Tanibet Pérez de los Santos Mondragon

# ÍNDICE

<b>UNIDAD 1</b>	5
<b>1.1 Aspectos básicos del sistema operativo</b>	5
1.1.1 Objetivos	6
1.1.2 Evolución	7
1.1.3 Tipos: mainframe, servidor, multiprocesador, embebidos y tiempo real	10
<b>1.2 Fundamentos del núcleo del sistema operativo</b>	12
1.2.1 Tipos: monolítico, por capas, micronúcleo, máquina virtual	13
<b>1.3 Modos de operación de un sistema operativo</b>	14
1.3.1 Interrupciones	15
1.3.2 Usuario y Kernel	16
1.3.3 Contadores	17
<b>1.4 Llamadas a sistema</b>	19
1.4.1 Procesos y señales	19
1.4.2 Directorios y archivos	20
1.4.3 Protección	22
1.4.4 Diversos	23
<b>1.5 Shell</b>	24
1.5.1 Argumentos y variables	25
1.5.2 Estructuras de control	26
1.5.3 Manipulación de cadenas	27
1.5.4 Funciones Interpretación de Comandos: Analiza la sintaxis de lo que el usuario escribe.	27
Programación (Shell Scripting): Permite crear archivos de texto con una serie de comandos que se ejecutan automáticamente.	28
<b>UNIDAD 2</b>	29
<b>2.1 Modelo de procesos</b>	29
2.1.1 Fundamentos: jerarquía, bloque de control de proceso (BCP)	29
2.1.2 Operaciones	30

2.1.3 Estados .....	31
2.1.4 Implementación .....	32
2.2 Modelo multi-hilo .....	33
2.2.1 Bibliotecas .....	34
2.2.2 Implementación .....	35
2.2.3 Hilos en modo usuario y kernel .....	35
2.3 Comunicación entre procesos .....	37
2.3.1 Memoria compartida .....	38
2.3.2 Paso de mensajes .....	38
2.3.3 Tuberías .....	39
2.4 Sincronización entre procesos/hilos .....	40
2.4.1 Principios de la sincronización: .....	41
2.4.2 Semáforos y mutex .....	41
2.4.3 Monitores .....	42
2.5 Planificación .....	42
2.5.1 Criterios .....	43
2.5.2 Algoritmos .....	43
2.5.3 Planificación de hilos .....	44
UNIDAD 3 .....	46
3.1 Abstracción de la memoria .....	46
3.1.1 Organización de la memoria .....	46
3.1.2 Administración del almacenamiento .....	47
3.1.3 Memoria de intercambio .....	48
3.1.4 Manejo de memoria con mapa de bits .....	48
3.1.5 Manejo de memoria con listas ligadas .....	49
3.2 Memoria virtual .....	49
3.2.1 Paginación .....	50
3.2.2 Tablas de página .....	50
3.2.3 Algoritmos de sustitución de páginas .....	51

3.3 Segmentación .....	52
3.3.1 Implementación de la segmentación pura .....	53
3.3.2 Segmentación con paginación.....	54
UNIDAD 4 .....	55
4.1 Estructura del sistema de archivos .....	55
4.1.1 Capa del sistema de archivos .....	56
4.2. Implementación.....	57
4.2.1 De archivos .....	57
4.2.2 De directorios .....	57
4.2.3 Archivos compartidos .....	58
4.3 Métodos de asignación de espacio .....	59
4.3.1 Contigua.....	59
4.3.2 Ligada .....	59
4.3.3 Indexada.....	60
4.3.4 Administración del espacio en disco .....	61
4.4 Sistemas de archivos estructurados .....	61
4.4.1 Por bitácoras.....	61
4.4.2 Por diario. ....	62
4.5 Optimización del sistema de archivos .....	62
4.5.1 Recuperación.....	62
4.5.2 Consistencia .....	63
4.5.3 Rendimiento.....	63
UNIDAD 5 .....	65
5.1 Principios del hardware de E/S .....	65
5.1.1 Dispositivos de E/S.....	65
5.1.2 Controladores de dispositivos .....	66
5.1.3 Interrupciones.....	67
5.1.4 Acceso de memoria directo (DMA).....	67
5.2 Principios del software de E/S.....	68

5.2.1 Objetivos del software de E/S .....	68
5.2.2 E/S programadas .....	69
5.2.3 E/S manejadas por interrupciones .....	70
5.2.4 E/S usando DMA .....	70
5.3 Capas de software .....	70
5.3.1 Manejador de interrupciones .....	71
5.3.2 Controladores de los dispositivos .....	72
5.3.3 Software modo usuario para E/S .....	73
UNIDAD 6 .....	74
6.1 El ambiente de seguridad .....	74
6.1.1 Seguridad en los sistemas operativos .....	74
6.1.2 Control de acceso a los recursos .....	75
6.1.3 Implementación de matrices de acceso .....	75
6.1.4 Modelos formales de seguridad .....	76
6.2 Virtualización .....	76
6.2.1 Emulación .....	77
6.2.2 Virtualización asistida por hardware .....	77
6.2.3 Paravirtualización .....	78
6.2.4 Contenedores .....	79
BIBLIOGRAFÍA .....	80

# UNIDAD 1

## Estructura de un Sistema operativo

### 1.1 Aspectos básicos del sistema operativo

Un Sistema Operativo es el programa (o conjunto de programas) que actúa como intermediario entre el usuario de una computadora y el hardware de esta. Su propósito es proporcionar un entorno en el que el usuario pueda ejecutar programas de manera conveniente y eficiente.

Sus objetivos principales son:

- **Conveniencia y Eficiencia:** Hacer que el sistema informático sea cómodo de usar y que el hardware se utilice de manera eficiente.
- **Simplificación:** Su función es "vestir" a la "máquina desnuda" (hardware sin software) para simplificar su manejo y utilización, haciéndolo seguro.
- **Ejecución de programas:** Proporcionar un entorno en el cual los usuarios puedan ejecutar programas y resolver problemas computacionales fácilmente.

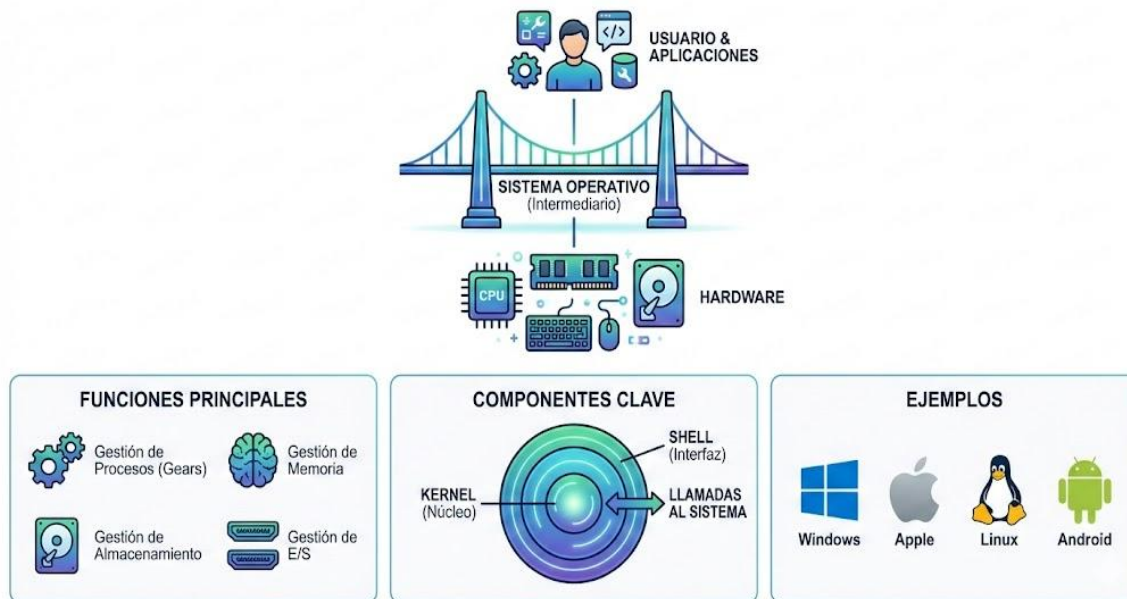
Estructura y Ubicación:

- **El Kernel (Núcleo):** Es el programa que se ejecuta en todo momento en la computadora. Es la capa más cercana al hardware y gestiona los recursos y la funcionalidad básica.
- **Programas del Sistema:** Son programas asociados al sistema operativo pero que no necesariamente forman parte del kernel.
- **Shell (Intérprete de mandatos):** Es la capa que permite el diálogo interactivo con el usuario; aunque es esencial, algunos autores consideran que no es parte del sistema operativo en sí, sino un programa que utiliza sus servicios.

Ejemplos de Sistemas Operativos:

- **De Escritorio:** Microsoft Windows, macOS, distribuciones de GNU/Linux (Ubuntu, Fedora, Debian).
- **Móviles:** Android (basado en el kernel de Linux), iOS.
- **Servidores y Especializados:** Red Hat Enterprise Linux, FreeBSD, Solaris.

## ASPECTOS BÁSICOS DEL SISTEMA OPERATIVO



### 1.1.1 Objetivos

Los objetivos principales buscan encontrar un equilibrio entre la experiencia del usuario y el aprovechamiento óptimo del hardware, sin embargo, se pueden encontrar más objetivos según la capa que se analice.

Para el usuario final el sistema operativo debe cumplir con:

- **Ejecución de programas:** Proporcionar un entorno en el cual el usuario pueda ejecutar programas de manera conveniente y eficiente. El objetivo fundamental es facilitar la resolución de los problemas del usuario.
- **Simplificación y Comodidad:** Simplificar el manejo y la utilización de la computadora. Su fin es hacer que el sistema informático sea más cómodo de usar.
- **Eficiencia:** Gestionar los recursos del sistema (hardware y software) de manera eficiente y correcta.

Internamente a nivel sistema operativo debe cumplir con:

- **Asignación de Recursos:** El objetivo es proporcionar una asignación ordenada y controlada de procesadores, memorias y dispositivos de E/S entre los variados programas que compiten por ellos.
- **Operación Eficiente:** Asegurar la operación eficiente del sistema en sí mismo, maximizando el rendimiento.

- **Protección:** Garantizar que los programas y usuarios no interfieran entre sí ni con el funcionamiento correcto del sistema. Esto incluye controlar el acceso a los recursos y asegurar la autenticación de los usuarios.



### 1.1.2 Evolución

La evolución de los sistemas operativos es una historia de adaptación constante para aprovechar el hardware cada vez más potente y satisfacer las necesidades de los usuarios. Se divide tradicionalmente por "generaciones" ligadas a la tecnología electrónica del momento.

#### Prehistoria y Primera Generación (1945 - 1955): Tubos de Vacío

- **Estado Inicial:** En los primeros años (década de 1940), no existían los sistemas operativos. Las máquinas (como ENIAC) eran enormes, usaban tubos de vacío y se programaban mediante tableros de conexiones o lenguaje máquina.
- **Modo de Operación:** El trabajo era en serie. El programador debía reservar la máquina, cargar su programa manualmente y depurar examinando registros y memoria. El tiempo de preparación era muy alto en comparación con el tiempo de ejecución.

#### Segunda Generación (1955 - 1965): Transistores y Sistemas por Lotes

Es la introducción de los transistores, haciendo las computadoras más fiables. Se distinguían líneas de computadoras para cálculo científico (orientadas a palabras) y comercial (orientadas a caracteres).



Predominio del procesamiento por lotes. Para mejorar la eficiencia, se leían los trabajos de tarjetas a cintas magnéticas en máquinas satélite baratas (como la IBM 1401), la computadora principal procesaba la cinta, y la salida se imprimía off-line.

Innovaciones:

- Desarrollo de la independencia del dispositivo: el usuario especificaba que quería usar una unidad, y el SO la asignaba dinámicamente.
- Primeros sistemas de Tiempo Real para aplicaciones militares.

### Tercera Generación (1965 - 1980): Circuitos Integrados y Multiprogramación

Un solo sistema operativo podía servir a varios propósitos y usuarios. Se introduce la Multiprogramación (varios programas en memoria para que la CPU siempre esté ocupada) y el Tiempo Compartido (varios usuarios conectados a la vez). IBM introdujo la familia System/360 basada en Circuitos Integrados (ICs), diseñada para manejar tanto aplicaciones científicas como comerciales con un mismo sistema operativo (OS/360).

- Multiprogramación: Se introdujo para evitar que la CPU estuviera ociosa mientras esperaba operaciones de E/S. La memoria se particionaba para mantener varios trabajos activos simultáneamente; cuando uno esperaba E/S, la CPU cambiaba a otro.
- Spooling: (Simultaneous Peripheral Operation On Line). Capacidad de leer trabajos del disco tan pronto llegaban, eliminando la necesidad de cargar cintas manualmente para cada lote.
- Tiempo Compartido (Timesharing): Variante de la multiprogramación donde cada usuario tiene una terminal en línea. La CPU alterna rápidamente entre usuarios, dando la ilusión de dedicación exclusiva. Pioneros: CTSS (MIT) y MULTICS (MIT/Bell Labs/GE).
- Nacimiento de UNIX: Ken Thompson, tras trabajar en MULTICS, desarrolló una versión simplificada (UNIX) en una minicomputadora PDP-7. Se reescribió en C, facilitando su portabilidad.

### Cuarta Generación (1980 - Presente): Computadoras Personales

En esta generación llega los microprocesadores (LSI y VLSI), el SO se vuelve accesible para el individuo y surgieron sistemas operativos para computadoras personales como CP/M y MS-DOS los cuales ya incluían una interfaz gráfica (GUI); Se pasó de interfaces de línea de comandos a interfaces gráficas amigables (ventanas, iconos, ratón), popularizadas por Macintosh y Windows.

Redes y Sistemas Distribuidos:

- **Sistemas Operativos de Red:** Los usuarios son conscientes de la existencia de múltiples computadoras y pueden acceder a recursos remotos (login remoto, transferencia de archivos).
- **Sistemas Distribuidos:** Aparecen como un único sistema ante el usuario, aunque ejecutan en múltiples máquinas. Gestionan la migración de datos y procesos.
- **Middleware:** Capa de software sobre el SO que permite gestionar sistemas distribuidos heterogéneos.

### Tendencias Recientes y Quinta Generación (1990 - Presente)

Se presentan los sistemas que gestionan recursos en red de forma transparente (como si fueran una sola máquina) y optimizados para dispositivos táctiles y portátiles. Sistemas como Android e iOS dominan, gestionando recursos limitados (batería, memoria) y nuevas interfaces (táctiles).

- **Virtualización:** Resurgimiento de tecnologías de los años 60 (como en VM/370) que permiten ejecutar múltiples sistemas operativos (máquinas virtuales) sobre un mismo hardware. Es la base de la computación en la nube.
- **Computación en la Nube (Cloud):** Entrega de computación, almacenamiento y aplicaciones como servicio a través de la red (SaaS, PaaS, IaaS).
- **Código Abierto:** Movimiento significativo con sistemas como Linux, permitiendo estudiar y modificar el código fuente del sistema operativo.



### **1.1.3 Tipos: mainframe, servidor, multiprocesador, embebidos y tiempo real**

#### **a) Sistemas de Mainframe**

Son los "gigantes" de la computación. Están diseñados para manejar miles de procesos y cantidades masivas de entrada/salida (E/S) de forma simultánea. Son sistemas orientados al procesamiento de grandes volúmenes de datos y transacciones críticas en entornos corporativos. Se encuentran principalmente en centros de datos de grandes bancos, aseguradoras y agencias gubernamentales.

Servicios Clave:

- Procesamiento por lotes (Batch): Ejecución de trabajos rutinarios sin interacción (ej. nóminas).
- Procesamiento de transacciones: Manejo de miles de pequeñas peticiones por segundo (ej. reservaciones aéreas).
- Tiempo compartido: Varios usuarios ejecutando tareas a la vez.

Ejemplos: IBM z/OS (usado en los modelos modernos como el IBM z17).

#### **b) Sistemas de Servidor**

Se ejecutan en servidores, que pueden ser desde estaciones de trabajo potentes hasta mainframes. Son sistemas diseñados para dar servicio a múltiples usuarios a través de una red, permitiendo compartir recursos de hardware y software. Sus funciones se centran en la gestión de impresión, archivos, bases de datos o servicios web.

Se encuentran principalmente en proveedores de servicios de internet (ISP), empresas de hosting y redes locales corporativas.

Ejemplos: Windows Server, Red Hat Enterprise Linux (RHEL), FreeBSD.

#### **c) Sistemas Multiprocesador**

Una tendencia creciente para obtener potencia de "grandes ligas" conectando varias CPUs en un solo sistema. Son sistemas operativos diseñados para coordinar la comunicación y el trabajo entre múltiples procesadores que comparten el mismo bus, reloj, memoria y dispositivos periféricos. Internamente utilizan SMP (Symmetric Multiprocessing), donde cada procesador ejecuta una copia idéntica del SO y se comunican entre sí.

Se encuentran principalmente en estaciones de trabajo de alto rendimiento para diseño 3D, investigación científica y servidores de alto tráfico.

Ejemplos: Variantes especializadas de Solaris, Linux y Windows 11/10 Pro (con soporte para múltiples sockets).

#### d) Sistemas Embebidos (Integrados)

Son sistemas "invisibles" que residen en dispositivos que no solemos llamar computadoras. Son sistemas operativos diseñados para funcionar en dispositivos de propósito especial con recursos de hardware muy limitados (memoria y CPU reducidas). Sin embargo poseen alta eficiencia, fiabilidad y, a menudo, no tienen una interfaz de usuario compleja.

Se encuentran en: Hornos de microondas, sistemas de frenado ABS, televisores inteligentes y cajeros automáticos.

Ejemplos: TinyOS, Embedded Linux, Windows IoT, QNX.

#### e) Sistemas de Tiempo Real (RTOS)

En estos sistemas, el tiempo es el recurso más crítico. No basta con que la operación sea correcta; debe ocurrir en un intervalo exacto. Son sistemas que garantizan una respuesta determinista ante eventos externos dentro de límites de tiempo estrictos. Se encuentran principalmente en la robótica industrial, equipos médicos (monitores cardíacos) y sistemas de control aeroespacial.

Clasificación:

- Tiempo Real Duro (Hard): El incumplimiento de un plazo es fatal (catástrofe). Ej: Control de vuelo, reactores nucleares.
- Tiempo Real Suave (Soft): El incumplimiento degrada el rendimiento, pero no destruye el sistema. Ej: Streaming de video, audio digital.

Ejemplos: VxWorks, FreeRTOS, RTLinux.



### 1.2 Fundamentos del núcleo del sistema operativo

El Núcleo (Kernel) es la parte central y fundamental de un sistema operativo. Es un software que se ejecuta en un nivel privilegiado y actúa como el gestor principal de los recursos del sistema. Es el primer programa que se carga al iniciar la computadora y permanece en la memoria RAM hasta que el sistema se apaga.

La función esencial del núcleo es gestionar directamente los recursos de la máquina, como el procesador, el tratamiento de interrupciones y las funciones básicas de manipulación de memoria. Interactúa directamente con el hardware y proporciona la funcionalidad básica sobre la cual se construyen las capas de servicios y las aplicaciones de usuario.

Para garantizar la seguridad y protección del sistema, el hardware distingue entre al menos dos niveles de ejecución:

- I. **Modo Núcleo (o Supervisor):** Es el nivel más permisivo. El procesador puede ejecutar todas las instrucciones de máquina y acceder a todos los registros y mapas de memoria y E/S sin restricción. Es donde se ejecutan las funciones críticas del sistema operativo.
- II. **Modo Usuario:** Es el nivel menos permisivo. La computadora ejecuta solo un subconjunto de instrucciones y el acceso a ciertos registros y zonas de memoria está prohibido. Los programas de usuario se ejecutan en este modo.

### 1.2.1 Tipos: monolítico, por capas, micronúcleo, máquina virtual

#### a) Estructura Monolítica:

Todos los componentes del sistema operativo se integran en un único programa (un solo archivo binario estático) que se ejecuta en un único espacio de direcciones. Es la estructura más sencilla y antigua. Todos los servicios del sistema (gestión de memoria, drivers, sistema de archivos) residen dentro del mismo núcleo.

- Ventaja: Eficiencia en la ejecución.
- Desventaja: Difícil de mantener y modificar; un error en un componente puede afectar a todo el sistema.

#### b) Estructura por Capas (Layered)

El sistema se divide en una jerarquía de capas. La capa 0 es el hardware y la capa más alta es la interfaz de usuario. Cada capa solo puede invocar funciones de la capa inmediatamente inferior.

- Ventaja: Facilita la depuración. Si hay un error en la capa 3, sabemos que el problema no está en las capas inferiores.
- Desventaja: Cada petición debe atravesar múltiples capas antes de llegar al hardware, lo que añade latencia.

#### c) Microkernel (Micronúcleo):

Estructura el sistema operativo eliminando componentes no esenciales del núcleo y moviéndolos al espacio de usuario en forma de procesos servidores. El micronúcleo solo conserva funciones mínimas (gestión de interrupciones, memoria básica y comunicación entre procesos).

- Ventaja: Alta fiabilidad, seguridad y facilidad de extensión.
- Desventaja: Posible sobrecarga de rendimiento debido al paso de mensajes entre módulos

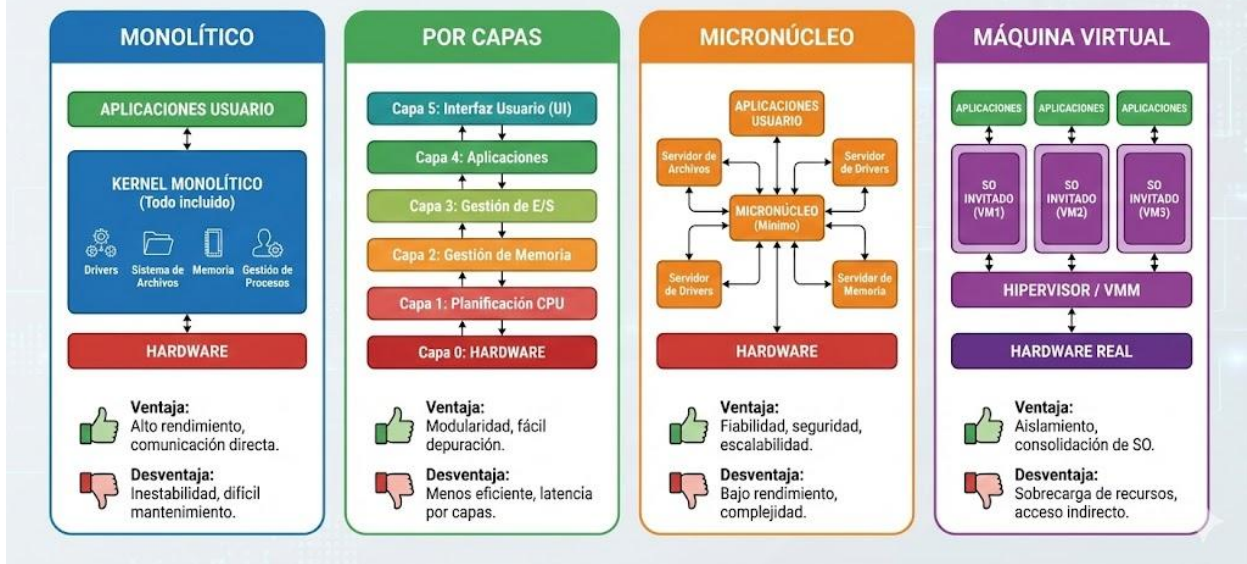
#### d) Máquina Virtual (Virtual Machine)

Proporciona una interfaz idéntica al hardware subyacente. El sistema operativo cree que tiene el hardware para él solo, pero en realidad está corriendo sobre una capa de software llamada Hypervisor.

- Ventaja: Un virus o fallo en una máquina virtual no afecta a las otras ni al sistema anfitrión.
- Desventaja: Ejecutar varios sistemas operativos a la vez consume mucha RAM y CPU.



## COMPARATIVA DE ESTRUCTURAS DE SISTEMAS OPERATIVOS



### 1.3 Modos de operación de un sistema operativo

Para garantizar la ejecución correcta del sistema y evitar que los programas de usuario interfieran con el sistema operativo o entre sí, el hardware de la computadora debe ser capaz de distinguir entre la ejecución de código del sistema operativo y código definido por el usuario.

La mayoría de los sistemas computacionales resuelven esto ofreciendo soporte de hardware para dos modos de operación diferenciados:

#### a) Modo Usuario (User Mode)

Es el modo en el que se ejecutan las aplicaciones del usuario (navegadores, editores de texto, juegos).

**Restricciones:** El hardware impide que el software en este modo ejecute instrucciones privilegiadas (como apagar la computadora, acceder directamente al disco duro o gestionar la memoria de otros programas).

**Seguridad:** Si un programa falla en este modo, solo se detiene ese programa, pero el sistema operativo sigue funcionando.

**Bit de modo:** Generalmente se representa con el valor 1.

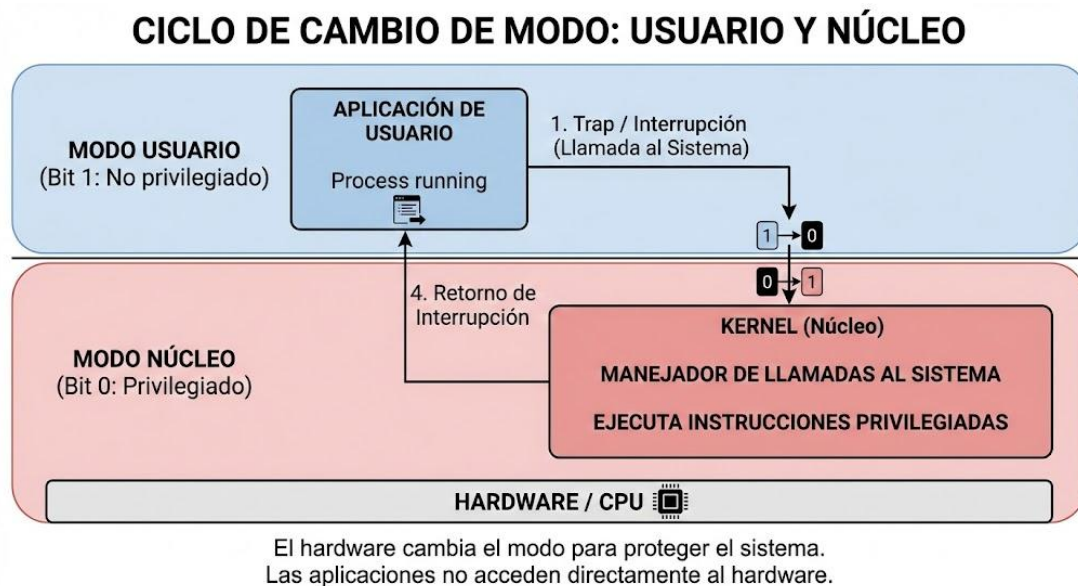
#### b) 2. Modo Núcleo / Privilegiado (Kernel Mode)

También llamado Modo Monitor, Modo Sistema o Modo Supervisor. Es el modo en el que se ejecuta el código del núcleo (Kernel).

Capacidades: Tiene acceso total e irrestricto al hardware y a todas las instrucciones de la CPU. Puede gestionar la memoria, controlar dispositivos de entrada/salida y detener procesos.

Instrucciones Privilegiadas: Solo pueden ejecutarse en este modo. Ejemplos: Cambiar el estado del reloj del sistema, gestionar tablas de interrupción o limpiar la memoria RAM.

Bit de modo: Generalmente se representa con el valor 0.



### 1.3.1 Interrupciones

Una interrupción es un evento que altera la secuencia de ejecución del procesador. Es el mecanismo mediante el cual el hardware (como los dispositivos de E/S) o el software notifican a la CPU que un evento requiere atención inmediata.

Permite que la CPU reaccione a eventos asíncronos (como la pulsación de una tecla o la finalización de una lectura de disco) sin necesidad de estar comprobando constantemente el estado de los dispositivos (técnica conocida como polling o espera activa), mejorando así la eficiencia y permitiendo la concurrencia.

Los sistemas operativos modernos son "impulsados por interrupciones" (interrupt-driven). Si no hay procesos que ejecutar ni dispositivos que atender, el SO espera ocioso a que ocurra una interrupción.

Ciclo de Procesamiento de una Interrupción:



- I. Solicitud: El dispositivo envía una señal a la CPU a través del bus del sistema.
- II. Detención: La CPU completa la instrucción actual y detiene su ejecución.
- III. Salvaguarda: El hardware guarda el estado crítico del proceso interrumpido (como el Contador de Programa y el registro de estado/PSW) para poder reanudarlo después. Generalmente, esto se guarda en la pila (stack).
- IV. Cambio de Modo: La interrupción fuerza automáticamente el cambio de modo usuario a modo núcleo (kernel), otorgando al procesador acceso total a los recursos.
- V. Despacho (Dispatch): La CPU utiliza un número asociado a la interrupción como índice en una tabla llamada Vector de Interrupciones. Esta tabla contiene las direcciones de memoria de las rutinas de servicio (manejadores) específicas para cada dispositivo o evento.
- VI. Ejecución del Manejador: Se ejecuta la Rutina de Servicio de Interrupción (ISR). Esta rutina determina la causa, realiza la tarea necesaria (como transferir datos o gestionar un error) y confirma al controlador que la interrupción ha sido atendida.
- VII. Restauración (RETI): Al finalizar, se ejecuta una instrucción de "retorno de interrupción" (como IRET o RETI). Esto restaura el estado guardado (registros y contador de programa) y devuelve la CPU al modo usuario, reanudando el proceso interrumpido exactamente donde se quedó

### **1.3.2 Usuario y Kernel**

Para garantizar el funcionamiento correcto del sistema, el hardware de la computadora debe ser capaz de distinguir entre la ejecución del código del sistema operativo y el código definido por el usuario,. Esto se logra mediante un bit de modo añadido al hardware (generalmente en el registro de estado o PSW), que identifica el modo actual:

- 0: Modo Kernel (o supervisor).
- 1: Modo Usuario.

#### **a) Modo Kernel (Núcleo / Supervisor)**

Es el modo más permisivo y seguro, donde se ejecuta el sistema operativo (o sus componentes principales). El procesador tiene acceso completo a todo el hardware y puede ejecutar cualquier instrucción que la máquina sea capaz de realizar, además tiene acceso a todos los registros, mapas de memoria y dispositivos de E/S sin restricciones.

Se utiliza para ejecutar las funciones críticas del sistema, gestionar interrupciones y asignar recursos.

#### **b) Modo Usuario**

Es un modo restringido donde se ejecutan los programas de los usuarios (compiladores, editores, navegadores). Solo se permite ejecutar un subconjunto de las instrucciones de la máquina.

Los programas en este modo no pueden ejecutar instrucciones que afecten el control de la máquina ni realizar Entrada/Salida (E/S) directamente. Tampoco pueden modificar los registros de gestión de memoria ni el bit de modo.

Si un programa en modo usuario intenta ejecutar una instrucción privilegiada o acceder a memoria protegida, el hardware genera una excepción (trap) y transfiere el control al sistema operativo, que generalmente termina el programa.

La arquitectura de los procesadores modernos (como x86) utiliza "anillos" para implementar esta jerarquía de privilegios:

- Anillo 0 (Kernel): El nivel con más privilegios. Aquí reside el núcleo.
- Anillos 1 y 2: Históricamente para controladores, aunque hoy en día se usan poco en sistemas comerciales comunes.
- Anillo 3 (Usuario): El nivel con menos privilegios. Aquí residen todas las aplicaciones de usuario.

### **1.3.3 Contadores**

Los contadores son componentes esenciales, tanto a nivel de hardware como de software, que permiten coordinar la ejecución de instrucciones, controlar el tiempo y medir el rendimiento.

#### **a) Contador de Programa (Program Counter - PC)**

Es el contador más importante para la ejecución de cualquier software. También se le conoce como Instruction Pointer (IP). Es un registro especializado de la CPU que contiene la dirección de memoria de la siguiente instrucción que debe ser ejecutada.

Su función en el SO es durante un Cambio de Contexto, el SO debe guardar el valor del PC del proceso que se detiene en su PCB (Process Control Block) para saber exactamente dónde reanudarlo después.

Se incrementa automáticamente después de captar cada instrucción, a menos que haya un salto o una interrupción. Este contador está integrado directamente en la Unidad de Control de la CPU.

#### **b) Reloj de Intervalo o Temporizador (Timer)**

Es un tipo de contador de hardware que el sistema operativo utiliza para mantener el control de la computadora. Un contador que se decrementa a intervalos regulares

mediante un oscilador de cristal. Cuando el contador llega a cero, genera una interrupción de hardware.

Su función en el SO es la multiprogramación; El SO carga un valor en este contador antes de darle la CPU a un usuario. Esto garantiza que ningún programa monopolice el procesador (si el tiempo se acaba, el contador llega a cero y el SO recupera el control).

Permite al sistema llevar la hora y fecha actual. Se encuentra en el chipset de la placa base o integrado en el procesador (ej. el PIT o Programmable Interval Timer).

### c) Contadores de Rendimiento (Performance Counters)

Son registros de hardware especiales que se utilizan para el monitoreo y la optimización. Registros que cuentan eventos específicos relacionados con el hardware durante la ejecución.

Ejemplos:

- Número de instrucciones ejecutadas.
- Fallos de caché (Cache misses).
- Ciclos de reloj perdidos esperando a la memoria RAM.

Se encuentran en procesadores modernos (Intel PMU o AMD Performance Monitoring).



## 1.4 Llamadas a sistema

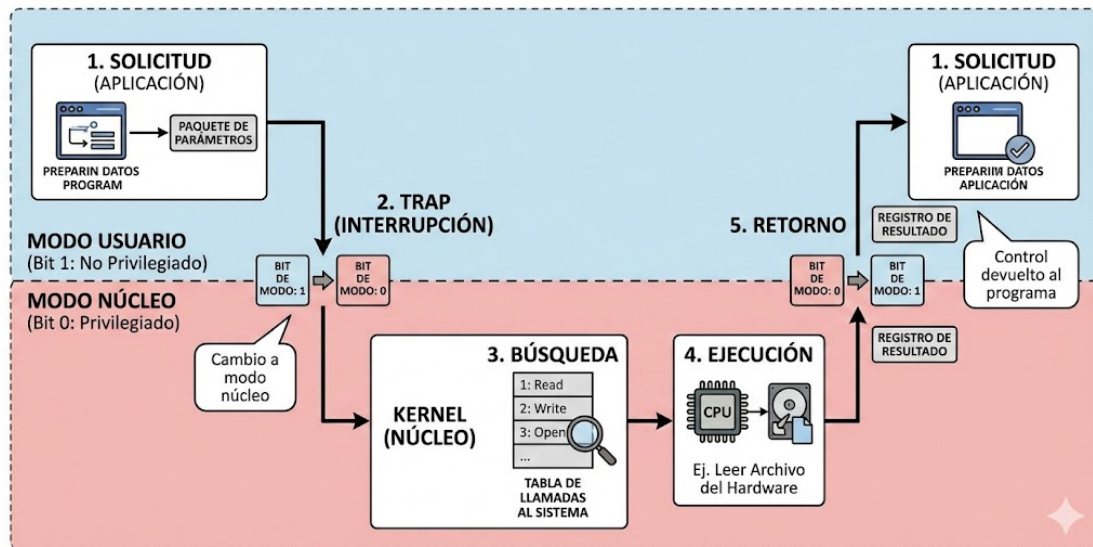
Una Llamada al Sistema es la interfaz programática entre una aplicación en ejecución y el núcleo del sistema operativo. Es el único método por el cual un proceso que se ejecuta en Modo Usuario puede solicitar un servicio al núcleo (que se ejecuta en Modo Kernel)

Dado que los programas de usuario se ejecutan en un modo restringido (modo usuario), no pueden realizar directamente operaciones de E/S o acceder a memoria protegida; deben usar estas llamadas para pedir al SO que lo haga por ellos.

La ejecución de una llamada al sistema implica una transición controlada del modo usuario al modo núcleo:

- I. Solicitud: El programa prepara los argumentos necesarios (parámetros).
- II. Trap: Se genera una instrucción especial (frecuentemente una interrupción de software) que detiene el programa y cambia el bit de modo de 1 (usuario) a 0 (núcleo).
- III. Búsqueda: El núcleo consulta la Tabla de Llamadas al Sistema para encontrar la función correspondiente al número de llamada solicitado.
- IV. Ejecución: El núcleo ejecuta la tarea (ej. leer un archivo).
- V. Retorno: El núcleo coloca el resultado en un registro, cambia el modo de vuelta a 1 (usuario) y devuelve el control al programa.

### MECANISMO DE UNA LLAMADA AL SISTEMA (SYSTEM CALL)



### 1.4.1 Procesos y señales

Un proceso es un programa en ejecución. A diferencia de un "programa" (que es una entidad pasiva almacenada en el disco, como un archivo .exe), un proceso es una entidad activa que consume recursos (CPU, memoria, archivos).

Un proceso incluye más que solo el código del programa (sección de texto); también incluye:

- Texto: El código ejecutable.
- Datos: Variables globales y estáticas.
- Heap (Montículo): Memoria dinámica reservada durante la ejecución.
- Stack (Pila): Datos temporales (parámetros de funciones, direcciones de retorno y variables locales).

Un proceso cambia de estado a medida que se ejecuta. El ciclo de vida estándar incluye:

- I. Nuevo (New): El proceso se está creando.
- II. Listo (Ready): El proceso espera a que se le asigne un procesador.
- III. En ejecución (Running): Se están ejecutando las instrucciones.
- IV. Bloqueado/Esperando (Waiting): El proceso espera a que ocurra algún evento (como la terminación de una E/S).
- V. Terminado (Terminated): El proceso ha finalizado su ejecución.

Una señal es una interrupción de software que se envía a un proceso para notificarle que ha ocurrido un evento específico. Es un mecanismo de comunicación asíncrona: el proceso no sabe exactamente cuándo recibirá la señal.

Una señal se genera por un evento, se entrega a un proceso y, una vez entregada, debe ser manejada, su funcionamiento es el siguiente:

- I. Generación: Un evento (como un error de división por cero o un usuario presionando teclas) genera la señal.
- II. Entrega: El núcleo envía la señal al proceso destino.
- III. Manejo: El proceso puede reaccionar de tres formas:
  - Ignorar la señal (algunas no se pueden ignorar).
  - Ejecutar la acción por defecto (generalmente terminar el proceso).
  - Capturar la señal con una función personalizada (Signal Handler).

#### **1.4.2 Directorios y archivos**

El Sistema de Archivos es la parte del SO que permite al usuario ver el almacenamiento como una colección lógica de unidades de información, en lugar de ver sectores físicos de un disco.

Es la unidad de almacenamiento lógico no volátil que define el sistema operativo para abstraer las propiedades físicas de los dispositivos de almacenamiento. Es una colección nombrada de información relacionada registrada en almacenamiento

secundario. Desde la perspectiva del usuario, es la parte más pequeña del almacenamiento lógico; los datos no pueden escribirse en almacenamiento secundario a menos que estén dentro de un archivo.

Estructura Interna:

- Secuencia de bytes: El archivo es una serie no estructurada de bytes. El SO no sabe qué hay dentro; el significado lo impone el programa de usuario (Modelo usado en UNIX y Windows).
- Secuencia de registros: El archivo es una colección de registros de longitud fija con estructura interna.
- Árbol: Árbol de registros, no necesariamente de la misma longitud, cada uno con un campo clave para búsquedas rápidas (usado en mainframes)

Atributos de un Archivo

- Nombre: La única información en formato legible para humanos.
- Identificador: Un número único (como el inode en Linux) que identifica al archivo dentro del sistema.
- Tipo: Necesario para sistemas que soportan diferentes formatos (.exe, .txt, .png).
- Ubicación: Puntero al dispositivo y a la posición física en el disco.
- Tamaño: Tamaño actual del archivo en bytes o bloques.
- Protección: Control de acceso (quién puede leer, escribir o ejecutar).
- Fechas: Registro de creación, última modificación y último acceso.

Operaciones Básicas:

- Crear y Escribir: Reservar espacio y grabar datos.
- Leer: Buscar la posición en disco y transferir a memoria.
- Reposicionar (Seek): Mover el puntero del archivo a una posición específica.
- Borrar y Truncar: Liberar espacio o vaciar el contenido manteniendo los metadatos.

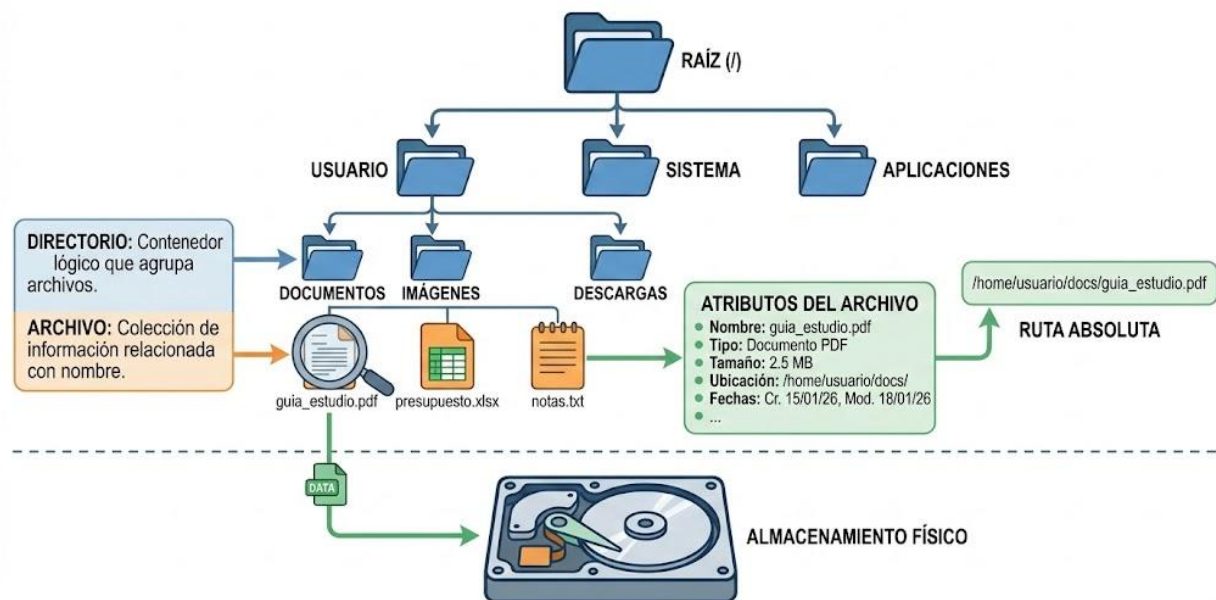
El directorio es un objeto (usualmente implementado como un archivo) que actúa como una tabla de símbolos para organizar los archivos, traduciendo nombres de archivos en sus descriptores internos (ej. inodos).

Estructuras de Directorio:

- I. Nivel Único: Todos los archivos de todos los usuarios están en el mismo directorio. Problema: colisión de nombres (nombres únicos requeridos para todos).

- II. Dos Niveles: Un Directorio Maestro (MFD) indexado por usuario, y cada usuario tiene su propio Directorio de Archivos de Usuario (UFD). Resuelve colisiones de nombres entre usuarios, pero aísla a los usuarios entre sí.
- III. Estructura de Árbol (Jerárquica): Es la más común. Existe un directorio raíz y cada archivo tiene un nombre de ruta único. Permite a los usuarios crear subdirectorios y agrupar archivos lógicamente.
- IV. Grafo Acíclico: Permite compartir archivos o directorios (que un mismo archivo aparezca en dos directorios diferentes). Se implementa mediante enlaces (links). Es más flexible, pero complica el borrado (se requiere conteo de referencias).

### CONCEPTOS DE DIRECTORIOS Y ARCHIVOS: ORGANIZACIÓN LÓGICA



#### 1.4.3 Protección

La protección es un aspecto crítico de los sistemas operativos modernos, ya que asegura que los recursos del sistema sean utilizados únicamente por aquellos procesos o usuarios que tienen los permisos adecuados.

Aunque a menudo se usan indistintamente, existe una distinción técnica entre seguridad y protección:

- **Seguridad**: Es un concepto más amplio que trata sobre la confianza en que el sistema y sus datos se preservarán ante amenazas (ataques externos, virus, desastres físicos, etc.). Se enfoca en el entorno.
- **Protección**: Es el conjunto de mecanismos internos del sistema operativo que controlan el acceso de los procesos y usuarios a los recursos definidos por el sistema (CPU, memoria, archivos, etc.)

Objetivos de la Protección:

- Prevención de errores: Evitar que un proceso acceda a la memoria o archivos de otro por un fallo en la programación.
- Asegurar la fiabilidad: Garantizar que los componentes del sistema funcionen de manera predecible.
- Principio de Mínimo Privilegio: Asegurar que cada proceso o usuario tenga acceso únicamente a la información y recursos necesarios para completar su tarea legítima.

#### **1.4.4 Diversos**

##### **a) Sistemas Operativos de Red (NOS - Network Operating Systems)**

Son sistemas diseñados para permitir que computadoras independientes compartan recursos y datos a través de una red local (LAN). El sistema operativo reside en un servidor y proporciona funciones para administrar datos, usuarios, grupos, seguridad y aplicaciones en red.

Los usuarios son plenamente conscientes de la existencia de múltiples computadoras y deben iniciar sesión en ellas explícitamente para acceder a recursos remotos.

Ejemplos: Versiones antiguas de Novell NetWare, Windows Server.

##### **b) Sistemas Distribuidos (DOS - Distributed Operating Systems)**

A diferencia de los de red, en un sistema distribuido los usuarios no son conscientes de que existen múltiples máquinas. Un conjunto de computadoras independientes que se presentan ante el usuario como un único sistema coherente. El SO distribuye las tareas automáticamente entre los diferentes nodos.

Ventajas:

- Compartición de recursos: Acceso a hardware especializado en cualquier nodo.
- Aceleración del cómputo: Una tarea se divide en varias máquinas.
- Fiabilidad: Si un nodo falla, los demás pueden continuar (tolerancia a fallos).

Ejemplo: Sistemas de búsqueda de Google, arquitecturas de microservicios masivos.



## 1.5 Shell

El Shell es la capa más externa del sistema operativo, encargada de permitir la interacción directa entre el usuario y los servicios del núcleo (Kernel). Es, esencialmente, el intérprete de órdenes del sistema.

Aunque es esencial, el shell no es parte del sistema operativo (no se ejecuta en modo kernel). Es un programa especial que se ejecuta en modo usuario, lo que permite que sea modificado o reemplazado fácilmente por otros intérpretes. Se inicia típicamente cuando un usuario inicia sesión o cuando se arranca un sistema interactivo.

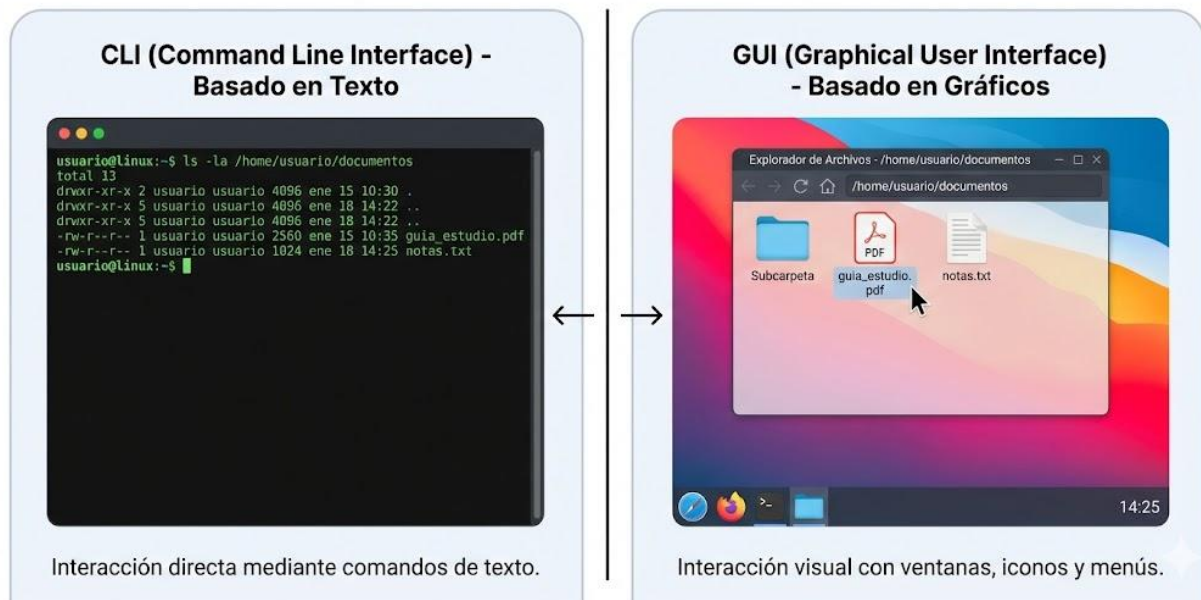
Aunque a menudo se asocia con una ventana de texto negro, el Shell puede presentarse de dos formas:

- CLI (Command Line Interface): Interfaz de línea de comandos basada totalmente en texto.
- GUI (Graphical User Interface): Interfaz gráfica de usuario que utiliza iconos, ventanas y punteros (como el Escritorio de Windows o el Finder de macOS).

Funcionamiento Básico:

- I. Prompt: Muestra una indicación en pantalla (ej. \$) señalando que está listo para recibir órdenes.
- II. Lectura: Espera y lee la línea de texto escrita por el usuario.
- III. Análisis: Extrae la primera palabra de la línea, asumiendo que es el nombre del programa a ejecutar, y procesa los argumentos.
- IV. Ejecución: Inicia la ejecución del mandato solicitado.
- V. Espera: Generalmente, el shell se suspende (espera) hasta que el mandato termina, tras lo cual vuelve a mostrar el prompt

## TIPOS DE SHELL: CLI vs. GUI



### 1.5.1 Argumentos y variables

Cuando el shell interpreta una orden del usuario, divide la línea de texto en palabras (tokens) para pasarlas al programa que se va a ejecutar.

Los argumentos son datos que se le pasan a un programa o comando en el momento de invocarlo para modificar su comportamiento o indicarle sobre qué objetos trabajar.

Parámetros Posicionales: Los argumentos se identifican por su posición:

- \$0: El nombre del script o comando.
- \$1, \$2, \$3...: El primer, segundo y tercer argumento respectivamente.
- \$#: El número total de argumentos pasados.
- \$\* o \$@: Representan todos los argumentos pasados como una lista.

Ejemplo: Si se ejecuta el comando `cp archivo1.txt backup/`, los argumentos son:

- archivo1.txt (Origen)
- backup/ (Destino)

Una variable es un nombre que representa un valor almacenado en la memoria. A diferencia de lenguajes como C o Java, en el Shell las variables suelen ser tratadas como cadenas de texto por defecto.

### Variables de Usuario (Locales):

Son definidas por el programador dentro de una sesión de terminal o un script.

- Definición: `NOMBRE="Archivo_Final"` (Nota: No debe haber espacios alrededor del `=`).
- Acceso: Se usa el símbolo `$` para obtener su valor. Ej: `echo $NOMBRE`.

### Variables de Entorno (Environment Variables):

Son variables globales que afectan el comportamiento de todos los procesos que se ejecutan en el shell. Se crean usando el comando `export` (ej. `export EDITOR=nano`).

Ejemplos comunes:

- `PATH`: Lista de directorios donde el shell busca los programas ejecutables.
- `HOME`: La ruta del directorio personal del usuario.
- `USER`: El nombre del usuario actual.
- `PWD`: El directorio de trabajo actual.

## 1.5.2 Estructuras de control

Las estructuras de control son instrucciones que alteran el flujo de ejecución lineal de un script. En lugar de ejecutar los comandos uno tras otro de arriba hacia abajo, permiten saltar secciones, elegir entre varias rutas o repetir un bloque de código.

### a) Estructuras Condicionales (Decisión)

Permiten ejecutar fragmentos de código solo si se cumple una condición específica (usualmente evaluando el "estado de salida" de un comando).

- `if - then - else`: Es la estructura básica. Si la prueba es verdadera, se ejecuta el bloque `then`; si no, el bloque `else`.

Ejemplo: Verificar si un archivo existe antes de borrarlo.

- `case`: Ideal para cuando tienes múltiples opciones posibles para una sola variable. Es más limpio que usar muchos `if` anidados.

Ejemplo: Un menú de opciones donde el usuario elige A, B o C.

### b) Estructuras de Repetición (Bucles / Loops)

Permiten ejecutar un mismo bloque de comandos varias veces.

- `for`: Se utiliza cuando conoces de antemano el número de iteraciones o tienes una lista definida de elementos.

Ejemplo: Cambiar la extensión a todos los archivos .txt de una carpeta.

- while: Repite el código mientras una condición sea verdadera. Se evalúa la condición antes de cada iteración.

Ejemplo: Leer un archivo línea por línea hasta llegar al final.

- until: Es lo opuesto al while; repite el código hasta que la condición se vuelva verdadera (es decir, mientras sea falsa).

### 1.5.3 Manipulación de cadenas

La manipulación de cadenas es el conjunto de operaciones que permiten modificar, extraer o analizar fragmentos de texto almacenados en variables. Es una herramienta poderosa para el procesamiento de datos sin necesidad de recurrir a lenguajes de programación externos.

Operaciones Fundamentales:

- a) Concatenación: Es la unión de dos o más cadenas para formar una nueva.
- b) Longitud de la Cadena: Permite obtener el número total de caracteres que contiene una variable.  
Sintaxis: `${#variable}`  
Ejemplo: Si `archivo="guia.txt"`, `${#archivo}` devolverá 8.
- c) Extracción de Subcadenas (Slicing): Permite obtener una porción de la cadena indicando la posición inicial y la longitud.  
Sintaxis: `${variable:inicio:longitud}`  
Ejemplo: Si `fecha="20260118"`, `${fecha:0:4}` devolverá 2026 (el año).
- d) Búsqueda y Reemplazo: Permite buscar un patrón dentro de la cadena y sustituirlo por otro texto.  
Sintaxis: `${variable/patrón/reemplazo}`  
Ejemplo: Si `ruta="/home/user/test.txt"`, `${ruta/test/final}` devolverá `/home/user/final.txt`.

### 1.5.4 Funciones

Interpretación de Comandos: Analiza la sintaxis de lo que el usuario escribe.

- Administración de Variables: Maneja variables de entorno (como PATH) que definen el comportamiento del sistema.
- Redirección y Tuberías (Pipes): Permite enviar la salida de un programa como entrada de otro (ej. `ls | grep .txt`).

Programación (Shell Scripting): Permite crear archivos de texto con una serie de comandos que se ejecutan automáticamente.

## UNIDAD 2

### Administración de procesos

#### 2.1 Modelo de procesos

Un proceso es esencialmente un programa en ejecución, incluyendo los valores actuales del contador de programa (PC), los registros y las variables.

Conceptualmente, cada proceso tiene su propia CPU virtual. Por supuesto, en la realidad, la CPU física conmuta de un proceso a otro de forma ultra rápida (multiprogramación).

Proceso vs. Programa

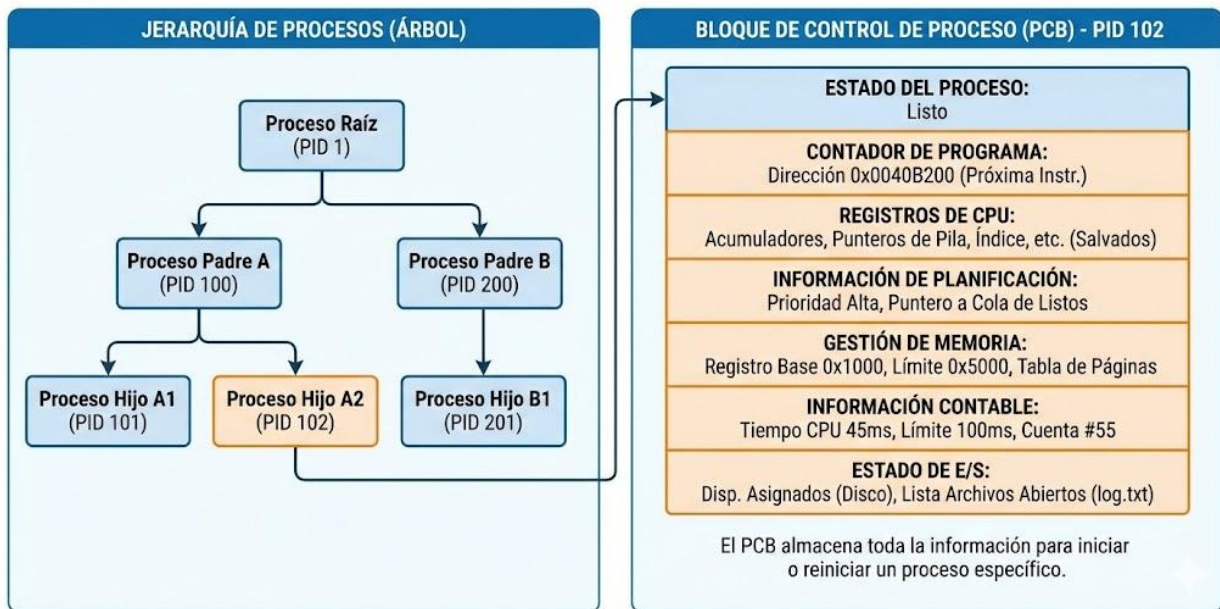
- Programa (Entidad Pasiva): Es un archivo almacenado en el disco (como un algoritmo escrito en un papel). No hace nada por sí mismo.
- Proceso (Entidad Activa): Es el programa cuando se carga en memoria y se le asignan recursos. Es la "lectura y ejecución" del algoritmo.

##### 2.1.1 Fundamentos: jerarquía, bloque de control de proceso (BCP)

El sistema operativo mantiene una estructura de datos por cada proceso llamada Bloque de Control de Proceso (PCB) o tabla de procesos. Sirve como repositorio de toda la información necesaria para iniciar o reiniciar un proceso. Contiene:

- Estado del proceso: (nuevo, listo, ejecución, etc.).
- Contador de programa: Dirección de la próxima instrucción.
- Registros de CPU: Acumuladores, punteros de pila, etc. Esta información debe salvarse cuando ocurre una interrupción.
- Información de planificación: Prioridad, punteros a colas de planificación.
- Gestión de memoria: Valores de registros base/límite, tablas de páginas o segmentos.
- Información contable: Tiempo de CPU usado, límites de tiempo, números de cuenta.
- Estado de E/S: Dispositivos asignados, lista de archivos abiertos.

## JERARQUÍA Y BLOQUE DE CONTROL DE PROCESOS (PCB)



### 2.1.2 Operaciones

Para que un sistema operativo gestione el ciclo de vida de los procesos, debe proporcionar un conjunto de operaciones básicas. Estas permiten al sistema y a los usuarios controlar cuándo nace, qué hace y cuándo muere un proceso.

Las operaciones fundamentales que un núcleo (kernel) debe permitir son la creación y la terminación, aunque existen otras de control intermedio.

#### a) Creación de Procesos (Process Creation)

Un proceso puede ser creado por cuatro razones principales:

- Inicialización del sistema: Al arrancar el SO, se crean procesos de fondo (daemons o servicios).
- Solicitud del usuario: Abrir una aplicación (ej. hacer clic en el icono de un navegador).
- Inicio de un trabajo por lotes (Batch): Común en servidores y mainframes.
- Creación por otro proceso: Un proceso en ejecución (padre) crea uno nuevo (hijo).

#### b) Terminación de Procesos (Process Termination)

Un proceso termina cuando finaliza su última instrucción o el SO lo detiene por fuerza. Las causas comunes son:

- Salida normal (Voluntaria): El proceso termina su tarea con éxito (ej. cerrar un editor).
- Salida por error (Voluntaria): El proceso detecta que falta un archivo o parámetro y decide terminar.
- Error fatal (Involuntaria): Un error de software (ej. división por cero o acceso ilegal a memoria).
- Eliminado por otro proceso (Involuntaria): Un proceso con permisos suficientes (como el administrador) mata al proceso (comando kill en Linux o "Finalizar tarea" en Windows).

#### c) Otras Operaciones Comunes

- Bloqueo: El proceso se detiene a la espera de un evento (E/S).
- Despertar (Wake-up): El proceso pasa de bloqueado a listo cuando ocurre el evento.
- Cambio de prioridad: El SO ajusta qué tan importante es el proceso para recibir CPU.
- Suspensión: Mover un proceso de la memoria principal al disco para liberar espacio (Swap).

### 2.1.3 Estados

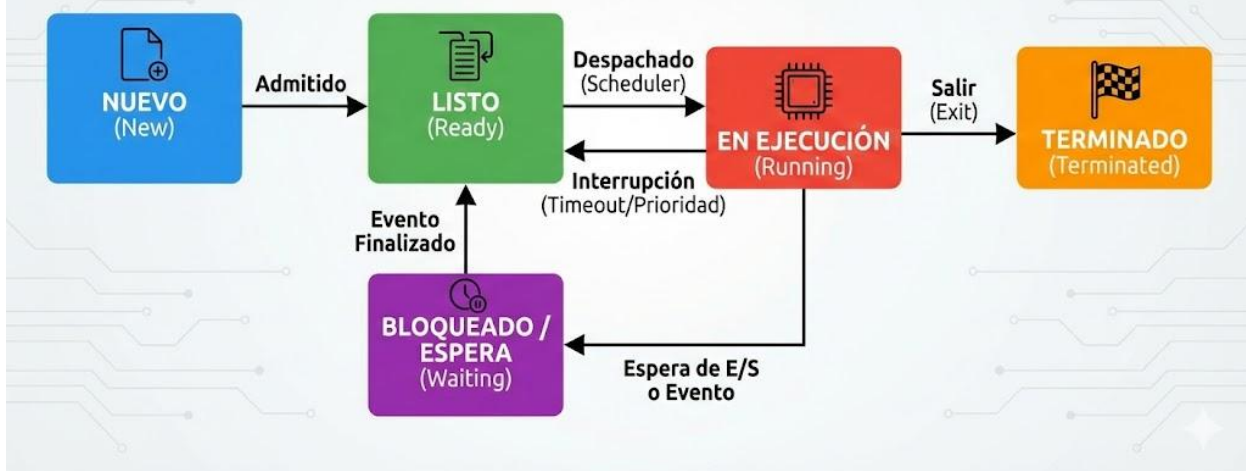
A medida que un proceso se ejecuta, cambia de estado. Aunque los nombres varían, los estados generales identificados son:

- I. Nuevo (New): El proceso se está creando.
- II. Ejecución (Running): Las instrucciones se están ejecutando en la CPU. En un sistema monoprocesador, solo un proceso puede estar en este estado a la vez.
- III. Listo (Ready): El proceso está listo para ejecutar, pero espera a que se le asigne el procesador.
- IV. Bloqueado/En espera (Waiting/Blocked): El proceso no puede ejecutarse hasta que ocurra un evento externo (como completar una E/S o recibir una señal).
- V. Terminado (Terminated): El proceso ha finalizado su ejecución.

Suspendido: Algunos sistemas incluyen estados de suspensión (listo y suspendido, bloqueado y suspendido) cuando el proceso es expulsado de la memoria principal al disco (zona de intercambio)



## DIAGRAMA DE TRANSICIÓN DE ESTADOS DEL PROCESO



### 2.1.4 Implementación

La implementación se basa en el uso de la Tabla de Procesos y el mecanismo de Interrupciones. La tabla contiene una entrada por cada proceso que existe en el sistema. Cada entrada se conoce como BCP (o Process Control Block) y es el repositorio central de toda la información necesaria para iniciar o reiniciar un proceso, actuando como el "esqueleto" del proceso cuando este no está en ejecución.

La implementación del modelo de procesos depende de lo que sucede cuando ocurre una interrupción (por ejemplo, el reloj del sistema avisa que el tiempo del proceso terminó). Los pasos técnicos son:

- **Hardware:** El hardware apila el contador de programa (PC) y otros registros.
- **Hardware:** El procesador carga un nuevo PC desde el Vector de Interrupciones.
- **Software (Ensamblador):** Una rutina en lenguaje ensamblador guarda el resto de los registros de la CPU en el PCB del proceso actual.
- **Software (Lenguaje C):** El sistema operativo ejecuta el manejador de la interrupción (ej. lee datos del disco o gestiona el temporizador).
- **Planificador (Scheduler):** El SO decide qué proceso se ejecutará a continuación.
- **Software (Ensamblador):** Se cargan los registros del nuevo proceso seleccionado desde su PCB y se inicia su ejecución.

## 2.2 Modelo multi-hilo

Un hilo (o proceso ligero) es la unidad básica de utilización de la CPU. Comprende un identificador de hilo, un contador de programa, un conjunto de registros y una pila. Mientras que un proceso tradicional tiene un solo hilo de ejecución, un proceso multi-hilo puede realizar varias tareas simultáneamente compartiendo el mismo espacio de memoria y recursos.

Beneficios del Multi-hilo:

- Capacidad de respuesta: Permite que una aplicación siga funcionando, aunque parte de ella esté bloqueada (ej. una interfaz de usuario que no se congela mientras se descarga un archivo).
- Compartición de recursos: Los hilos comparten la memoria del proceso por defecto, lo que facilita la comunicación entre ellos sin necesidad de mecanismos complejos como la memoria compartida.
- Economía: Crear y cambiar de contexto entre hilos es mucho más rápido y consume menos recursos que hacerlo entre procesos.
- Escalabilidad: En arquitecturas de multiprocesador, los hilos pueden ejecutarse en paralelo en diferentes núcleos de la CPU.

Existen tres formas comunes de establecer la relación entre los hilos de usuario y los hilos del kernel para lograr la ejecución concurrente:

a) Modelo Muchos a Uno (Many-to-One):

- Asigna muchos hilos de nivel de usuario a un solo hilo del kernel.
- La gestión se realiza en el espacio de usuario, lo que lo hace eficiente. Sin embargo, si un hilo realiza una llamada al sistema bloqueante, todo el proceso se bloquea.
- No permite la ejecución en paralelo en sistemas multinúcleo porque solo un hilo puede acceder al kernel a la vez.

b) Modelo Uno a Uno (One-to-One):

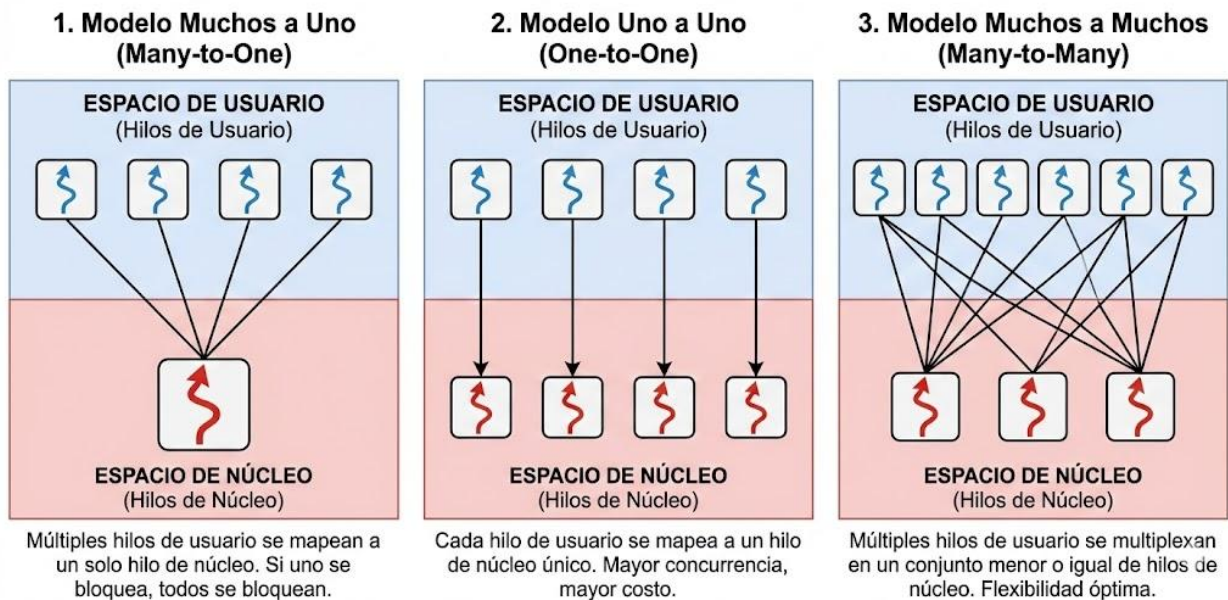
- Asigna cada hilo de usuario a un hilo del kernel.
- Proporciona mayor concurrencia; si un hilo se bloquea, otros pueden seguir ejecutándose. Permite el paralelismo real en multiprocesadores.
- La desventaja es la sobrecarga de crear un hilo de kernel por cada hilo de usuario, lo que puede afectar el rendimiento. Es el modelo usado por Windows y Linux.

c) Modelo Muchos a Muchos (Many-to-Many):

- Multiplexa muchos hilos de usuario a un número menor o igual de hilos del kernel.

- Ofrece flexibilidad: el desarrollador puede crear tantos hilos como necesite, y los hilos del kernel pueden ejecutarse en paralelo en un multiprocesador.
- Existe una variación llamada Modelo de dos niveles, que permite además ligar un hilo de usuario específico a un hilo de kernel.

## COMPARATIVA DE MODELOS DE MAPEO DE HILOS (THREADS)



### 2.2.1 Bibliotecas

Una biblioteca de hilos proporciona al programador una API para crear y gestionar hilos. Existen dos formas principales de implementarlas: como una biblioteca totalmente en el espacio de usuario (sin apoyo del kernel) o como una biblioteca a nivel del kernel (apoyada por el SO).

Las tres bibliotecas principales en uso son:

- I. **POSIX Pthreads:** Puede ser una biblioteca a nivel de usuario o de kernel. Es una especificación estándar (IEEE 1003.1c) utilizada en sistemas tipo UNIX como Linux, Solaris y macOS.
- II. **Windows API:** Es una biblioteca a nivel de kernel disponible en sistemas Windows.
- III. **Java Threads:** Dado que la JVM corre sobre un sistema operativo anfitrión, la API de hilos de Java se implementa generalmente usando la biblioteca de hilos del sistema subyacente (Pthreads o Windows API).

### 2.2.2 Implementación

La implementación de los hilos depende de si el sistema operativo es consciente de su existencia o si se gestionan internamente en el proceso.

a) Estructuras de Datos:

- Para gestionar hilos, se necesita una tabla de hilos (análoga a la tabla de procesos) que registre propiedades por hilo como el contador de programa, el puntero de pila y los registros.
- En implementaciones a nivel de usuario, esta tabla reside en el espacio de usuario dentro del sistema en tiempo de ejecución. En implementaciones de kernel, reside en el espacio del núcleo.

b) Ejemplos en Sistemas Reales:

- Linux: Utiliza la llamada al sistema clone(). Linux no distingue estrictamente entre procesos y hilos; ambos son "tareas" (tasks) que pueden compartir distintos niveles de recursos (memoria, descriptores de archivos) según las banderas (flags) pasadas a clone().
- Windows: Implementa el modelo uno a uno. Cada hilo contiene estructuras como el ETHREAD (bloque de hilo ejecutivo), KTHREAD (bloque de hilo del kernel) y TEB (bloque de entorno del hilo en modo usuario).

c) Activaciones del Planificador:

En sistemas híbridos o de muchos a muchos, se utiliza un mecanismo de comunicación entre el kernel y la biblioteca de hilos (como scheduler activations) mediante "upcalls" para notificar a la biblioteca sobre eventos de bloqueo y mantener la eficiencia.

### 2.2.3 Hilos en modo usuario y kernel

a) Hilos de Nivel de Usuario (User-Level Threads - ULT)

En este modelo, la gestión de hilos se realiza íntegramente en el Espacio de Usuario mediante una biblioteca de hilos. El núcleo del sistema operativo no tiene conocimiento de la existencia de estos hilos; para el núcleo, el proceso es una entidad única con un solo hilo de ejecución.

La creación, destrucción y el cambio de contexto de los hilos se realizan mediante llamadas a funciones de la biblioteca (como POSIX Pthreads o Java), no mediante llamadas al sistema.

Ventajas:

- Velocidad: El cambio de contexto es extremadamente rápido porque no requiere cambiar del Modo Usuario al Modo Kernel (no hay interrupciones ni traps de hardware).

- Portabilidad: Puede ejecutarse en cualquier sistema operativo, incluso si el núcleo no soporta hilos nativamente.
- Planificación a medida: Cada aplicación puede tener su propio algoritmo de planificación de hilos optimizado para sus necesidades.

#### Desventajas:

- Bloqueo Total: Si un hilo realiza una llamada al sistema que lo bloquea (como esperar por una lectura de disco), el núcleo bloquea a todo el proceso, deteniendo a todos los demás hilos.
- Sin Paralelismo Real: El núcleo solo asigna una CPU al proceso completo. Aunque el proceso tenga 100 hilos de usuario, solo uno podrá ejecutarse a la vez, incluso en una computadora con múltiples núcleos.

#### b) Hilos de Nivel de Kernel (Kernel-Level Threads - KLT)

En este modelo, el sistema operativo gestiona directamente los hilos en el Espacio de Kernel. El núcleo mantiene la información de contexto tanto para el proceso completo como para cada uno de sus hilos individuales. Todas las operaciones de gestión de hilos se realizan mediante llamadas al sistema.

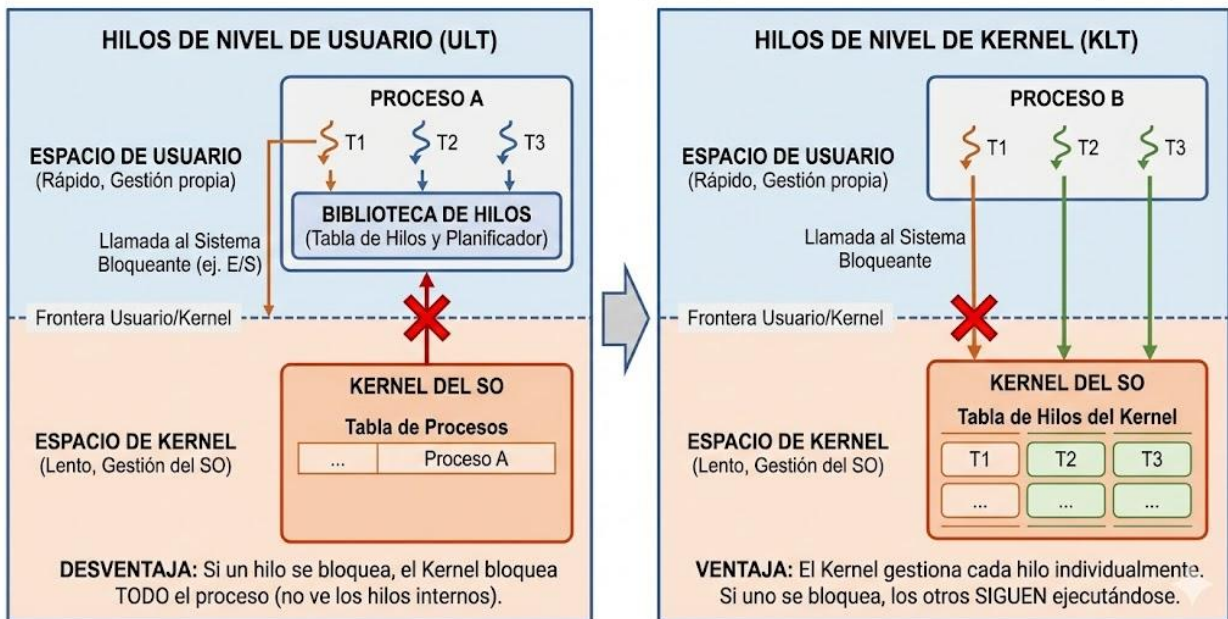
#### Ventajas:

- Multiprocesamiento: El núcleo puede planificar diferentes hilos del mismo proceso en diferentes núcleos de CPU de forma simultánea.
- Independencia de Bloqueo: Si un hilo se bloquea, el núcleo puede planificar otro hilo del mismo proceso para que continúe trabajando.
- Eficiencia del Núcleo: Las propias funciones del sistema operativo pueden ser multihilo (por ejemplo, para manejar múltiples interrupciones de red).

#### Desventajas:

- Costo de Rendimiento: Cualquier operación con hilos (creación, cambio de contexto) requiere un cambio de modo (de usuario a kernel), lo cual es significativamente más lento que en los ULT.
- Consumo de Recursos: El núcleo debe mantener una estructura de datos (como una pila de kernel y una entrada de tabla) para cada hilo.

## COMPARATIVA: HILOS DE USUARIO (ULT) vs. HILOS DE KERNEL (KLT)



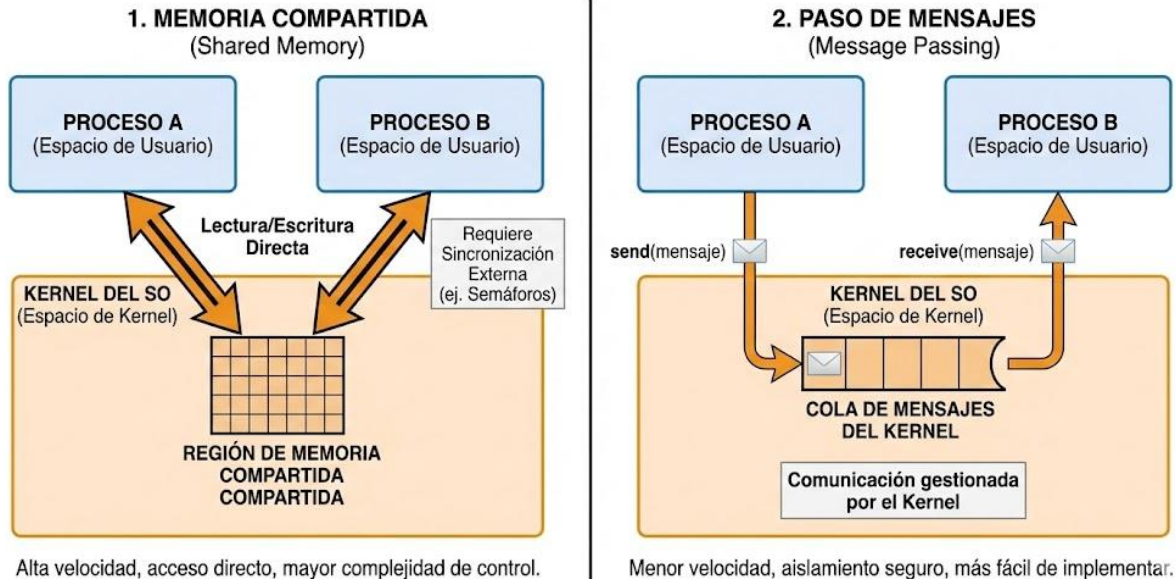
### 2.3 Comunicación entre procesos

Los procesos que se ejecutan concurrentemente en un sistema pueden ser independientes o cooperantes. Un proceso es cooperante si puede afectar o verse afectado por otros procesos, y estos necesitan mecanismos para comunicarse (intercambiar datos) y sincronizar sus acciones. Dado que los procesos tienen espacios de memoria aislados por seguridad, no pueden leer ni escribir en la memoria de otro proceso directamente; necesitan que el núcleo intervenga para facilitar esta comunicación.

Existen dos modelos fundamentales de IPC: memoria compartida y paso de mensajes.

- La memoria compartida es generalmente más rápida, pues las transferencias ocurren a velocidad de memoria y sin intervención del kernel (salvo para establecer la región), pero es más difícil de implementar en sistemas distribuidos.
- El paso de mensajes es útil para intercambiar cantidades menores de datos y es más fácil de implementar en sistemas distribuidos, aunque suele requerir llamadas al sistema que consumen más tiempo.

## COMPARATIVA DE MODELOS DE IPC: MEMORIA COMPARTIDA vs. PASO DE MENSAJES



### 2.3.1 Memoria compartida

En este modelo, dos o más procesos acuerdan eliminar la restricción del sistema operativo que normalmente impide que un proceso acceda a la memoria de otro, estableciendo una región de memoria común a ambos.

Una vez establecida la región, los procesos pueden leer y escribir datos en ella como si fuera memoria normal. La forma y ubicación de los datos la determinan los procesos, no el sistema operativo.

A menudo se implementa mediante archivos proyectados en memoria (memory-mapped files). Un archivo se asocia a una región de memoria virtual; las lecturas y escrituras en esa memoria equivalen a operaciones sobre el archivo.

- POSIX: Utiliza llamadas como `shm_open()` para crear un objeto de memoria compartida y `mmap()` para proyectarlo en el espacio de direcciones del proceso.
- Windows: Utiliza objetos de sección (file mapping objects) que pueden proyectarse en los espacios de direcciones de múltiples procesos.

El sistema operativo no se encarga de sincronizar los accesos. Son los procesos los que deben garantizar que no escriban en la misma ubicación simultáneamente, utilizando mecanismos externos (como semáforos).

### 2.3.2 Paso de mensajes

Este mecanismo permite a los procesos comunicarse y sincronizarse sin compartir el mismo espacio de direcciones. Es especialmente útil en entornos



distribuidos donde los procesos residen en computadoras diferentes conectadas por una red.

Se requieren al menos dos primitivas: `send(mensaje)` y `receive(mensaje)`. Los procesos deben tener una forma de referirse entre sí:

- **Comunicación Directa:** El proceso debe nombrar explícitamente al destinatario o remitente (ej. `send(P, mensaje)`). Puede ser simétrica (ambos se nombran) o asimétrica (solo el emisor nombra al receptor).
- **Comunicación Indirecta:** Los mensajes se envían y reciben a través de buzones (mailboxes) o puertos. Dos procesos se comunican solo si comparten un buzón. Esto permite mayor flexibilidad y modularidad.

El paso de mensajes puede ser bloqueante (síncrono) o no bloqueante (asíncrono):

- **Bloqueante:** El emisor se bloquea hasta que el mensaje es recibido, o el receptor se bloquea hasta que hay un mensaje disponible. La combinación de ambos (`send` y `receive` bloqueantes) se conoce como cita (rendezvous).
- **No bloqueante:** El emisor envía y continúa; el receptor recibe un mensaje válido o un nulo si no hay nada, permitiendo concurrencia.

Los mensajes residen temporalmente en una cola que puede tener capacidad cero (sin almacenamiento, obliga a la sincronización), capacidad limitada (el emisor se bloquea si está llena) o capacidad ilimitada.

### **2.3.3 Tuberías**

Una tubería actúa como un conducto que permite a dos procesos comunicarse, funcionando como un pseudoarchivo mantenido por el sistema operativo. Es un canal de comunicación unidireccional que actúa como un búfer (buffer) de tipo FIFO (First-In, First-Out). Un proceso escribe datos en un extremo de la tubería (el extremo de escritura) y otro proceso los lee desde el otro extremo (el extremo de lectura).

**Características Generales:**

- **Unidireccionalidad:** Los datos fluyen en un solo sentido. Si se requiere comunicación bidireccional, se deben crear dos tuberías.
- **Capacidad Limitada:** El búfer tiene un tamaño fijo gestionado por el núcleo. Si la tubería se llena, el proceso que escribe se bloquea hasta que el otro proceso lea algo.
- **Sincronización Implícita:** El sistema operativo maneja automáticamente el bloqueo de los procesos si la tubería está vacía (al leer) o llena (al escribir).

**Tipos de Tuberías:**



#### a) Tuberías Ordinarias (Anónimas):

Son temporales y solo existen mientras los procesos que las usan están en ejecución. Solo permiten comunicación entre procesos que tienen una relación padre-hijo (o ancestro-descendiente), ya que el descriptor de la tubería se hereda.

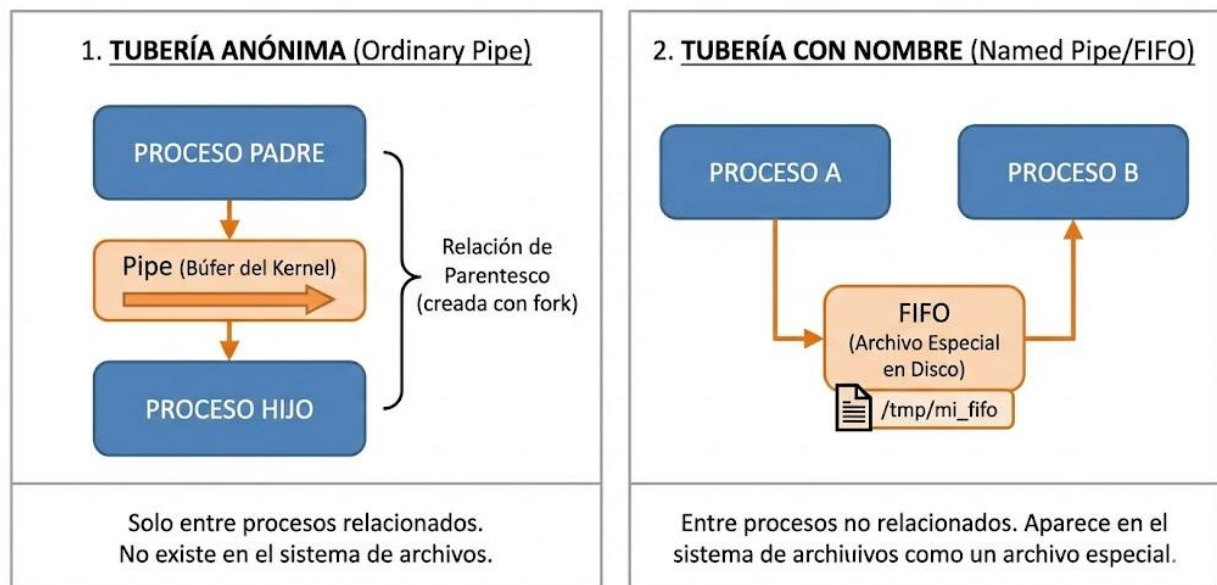
- Dejan de existir cuando los procesos terminan,.
- Solo pueden ser utilizadas por procesos que tengan una relación de parentesco (típicamente entre un proceso padre y su proceso hijo creado con `fork()`).

#### b) Tuberías con Nombre (Named Pipes / FIFOs):

Existen como un archivo especial en el sistema de archivos y tienen un nombre. La tubería existe en el disco incluso si los procesos terminan, hasta que sea borrada explícitamente.

- Permiten la comunicación entre procesos sin relación de parentesco.
- Son bidireccionales (aunque en UNIX suelen ser *half-duplex*) y persisten después de que los procesos terminan,.
- En UNIX se crean con `mkfifo()`; en Windows con `CreateNamedPipe()`,.

### DIFERENCIA VISUAL: TUBERÍA ANÓNIMA vs. TUBERÍA CON NOMBRE (FIFO)



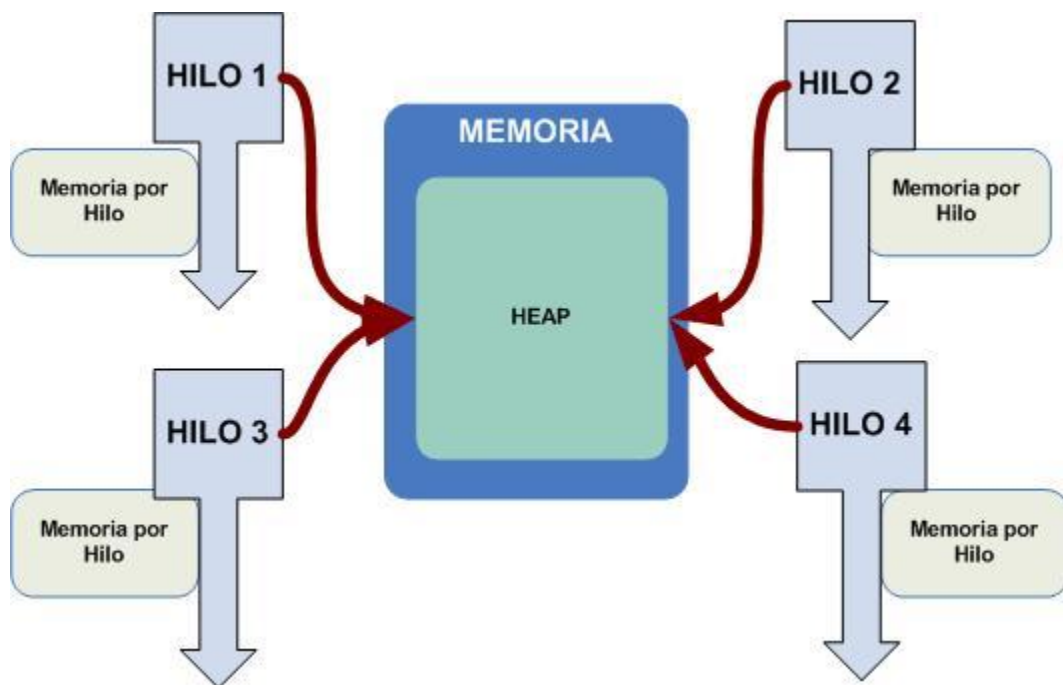
## 2.4 Sincronización entre procesos/hilos

La sincronización es el conjunto de técnicas que aseguran que los procesos o hilos cooperativos se ejecuten de manera ordenada, especialmente cuando acceden a recursos compartidos. Sin ella, los sistemas multihilo o multiproceso sufrirían de inconsistencia de datos.

### 2.4.1 Principios de la sincronización:

La necesidad de sincronización surge para evitar las condiciones de carrera, situaciones donde varios procesos acceden y manipulan datos compartidos concurrentemente, y el resultado final depende del orden particular en que se ejecutaron los accesos. Para prevenir estos errores, se identifican las secciones críticas, que son los segmentos de código donde se accede a recursos compartidos; el sistema debe garantizar la exclusión mutua, asegurando que, si un proceso está ejecutando su sección crítica, ningún otro pueda hacerlo simultáneamente.

Para implementar estas protecciones, es fundamental la atomicidad, que implica que ciertas operaciones (como comprobar y fijar un valor) sean indivisibles e ininterrumpibles por otros procesos. Sin embargo, el uso incorrecto de herramientas de sincronización puede llevar a un abrazo mortal (o *deadlock*), una situación donde un conjunto de procesos queda bloqueado indefinidamente porque cada uno posee un recurso y espera por otro que está retenido por otro proceso del mismo conjunto, creando un ciclo de dependencias del que no pueden salir.



### 2.4.2 Semáforos y mutex

Los semáforos, introducidos por Dijkstra, son variables enteras que actúan como herramientas de señalización mediante dos operaciones atómicas: wait (o P/down), que decrementa el valor y bloquea al proceso si el resultado es negativo, y signal (o V/up), que incrementa el valor y despierta a un proceso en espera. Los semáforos pueden ser contadores, utilizados para controlar el acceso a un número

finito de instancias de un recurso, o binarios, que oscilan entre 0 y 1 para garantizar exclusión mutua.

Por otro lado, los mutex (abreviatura de *mutual exclusion*) son cerrojos diseñados específicamente para proteger secciones críticas, operando de manera similar a un semáforo binario con estados de "bloqueado" y "desbloqueado". A diferencia de los semáforos generales, los mutex suelen incorporar el concepto de propiedad, requiriendo que el mismo hilo que adquirió el cerrojo sea el que lo libere, y son ampliamente utilizados en APIs como Pthreads y Windows por su eficiencia para prevenir condiciones de carrera en hilos.

### **2.4.3 Monitores**

Los monitores son construcciones de alto nivel o tipos abstractos de datos que encapsulan variables compartidas y los procedimientos para manipularlas, garantizando automáticamente que solo un proceso pueda estar activo dentro del monitor en un momento dado. Al delegar la responsabilidad de la exclusión mutua al compilador o al lenguaje de programación (como `synchronized` en Java), los monitores reducen los errores de programación comunes asociados con el uso explícito de semáforos.

Para gestionar la sincronización interna (por ejemplo, esperar a que un recurso esté disponible), los monitores utilizan variables de condición con operaciones `wait` y `signal`. Cuando un proceso invoca `wait` en una condición, se suspende y libera el bloqueo del monitor para que otro proceso pueda entrar; la operación `signal` reanuda a un proceso suspendido, existiendo diferentes semánticas (como "señalar y esperar" o "señalar y continuar") para determinar si el proceso que avisa cede inmediatamente el control o continúa ejecutándose,

### **2.5 Planificación**

La planificación de procesos es el corazón de la multiprogramación. Es la función del sistema operativo que decide qué proceso tendrá el control de la CPU en cada momento, buscando maximizar la eficiencia del sistema y la satisfacción del usuario.

En un sistema con un solo núcleo de CPU, solo un proceso puede ejecutarse a la vez. El resto debe esperar hasta que la CPU esté libre y pueda ser reprogramada. El objetivo de la planificación es mantener la CPU ocupada el mayor tiempo posible.

Objetivos principales:

- Equidad (Fairness): Que todos los procesos tengan una oportunidad justa de usar la CPU.
- Eficiencia: Maximizar el uso de la CPU (mantenerla al 100%).
- Rendimiento (Throughput): Maximizar el número de procesos completados por unidad de tiempo.
- Tiempo de respuesta: Minimizar el tiempo que tarda un proceso desde que se envía una solicitud hasta que se empieza a recibir respuesta (vital en sistemas interactivos).

### **2.5.1 Criterios**

El objetivo principal de la planificación es optimizar el comportamiento del sistema seleccionando entre criterios a menudo contradictorios, tales como maximizar la utilización de la CPU (manteniéndola ocupada entre el 40% y el 90% del tiempo) y maximizar el rendimiento o tasa de procesamiento (número de procesos completados por unidad de tiempo). Desde la perspectiva del proceso individual, los criterios clave son minimizar el tiempo de retorno (el intervalo desde la solicitud hasta la finalización), minimizar el tiempo de espera (la suma de los periodos pasados en la cola de listos) y optimizar el tiempo de respuesta, el cual es crítico en sistemas interactivos ya que mide el lapso desde la solicitud hasta que se produce la primera respuesta.

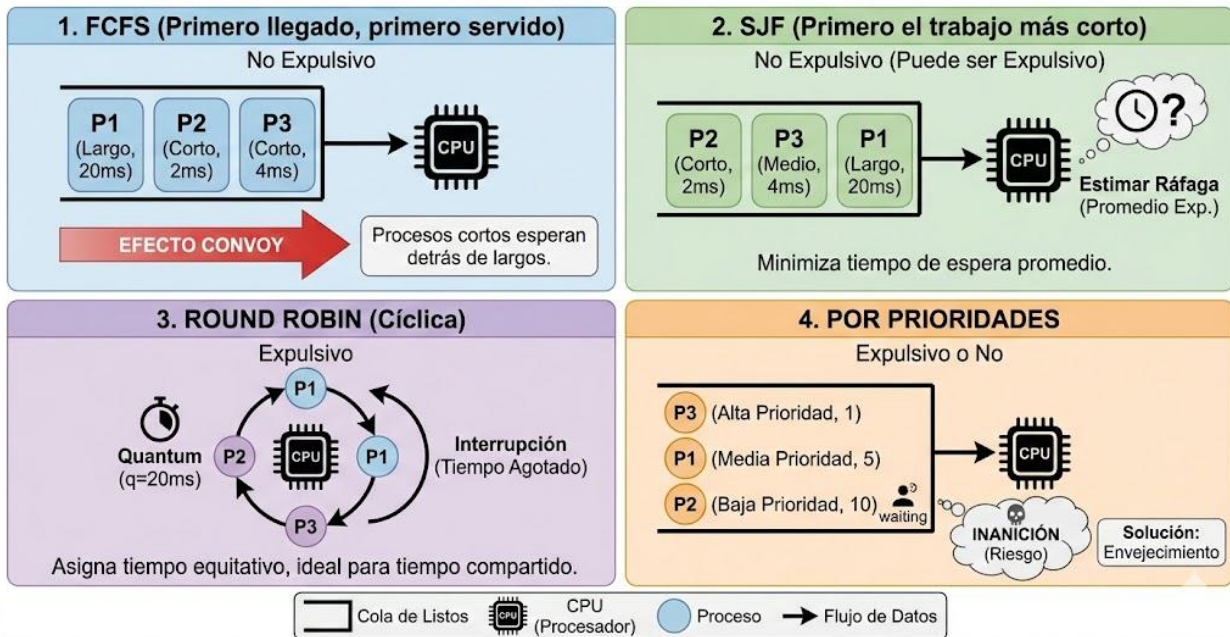
Otros criterios fundamentales buscan la justicia, asegurando que los procesos con características similares reciban un trato equitativo y evitando la postergación indefinida o inanición de procesos de baja prioridad, a veces mediante mecanismos de envejecimiento que aumentan su prioridad con el tiempo. Además, se valora la previsibilidad, para que un mismo trabajo tome aproximadamente el mismo tiempo en completarse independientemente de la carga del sistema, y el equilibrio en el uso de recursos, penalizando a los procesos que consumen recursos sobreutilizados o favoreciendo a aquellos que tienen un comportamiento deseable (como alta interactividad).

### **2.5.2 Algoritmos**

Existen diversos algoritmos clásicos, comenzando por el esquema "Primero llegado, primero servido" (FCFS), que es el más simple y no expulsivo pero sufre del efecto convoy donde procesos cortos esperan detrás de largos, y el algoritmo "Primero el trabajo más corto" (SJF o SPN), que minimiza el tiempo de espera promedio pero requiere estimar la duración de la próxima ráfaga de CPU, a menudo usando un promedio exponencial. La planificación por prioridades asigna la CPU al proceso con la prioridad más alta (externa o interna), pudiendo ser expulsiva o no, pero conlleva el riesgo de inanición para procesos de baja prioridad si no se implementan técnicas dinámicas de ajuste.

Para sistemas de tiempo compartido, el estándar es la planificación Cíclica o Round Robin, que es expulsiva y asigna a cada proceso una pequeña unidad de tiempo o *quantum* (típicamente entre 10 y 200 ms); si el proceso no termina en ese lapso, es interrumpido y colocado al final de la cola de listos. Algoritmos más avanzados, como las colas multinivel con retroalimentación, permiten que los procesos se muevan entre diferentes colas con distintas prioridades y tamaños de *quantum* basándose en su comportamiento histórico (CPU-bound vs I/O-bound), mientras que otros enfoques como la planificación por lotería distribuyen el tiempo de CPU de forma probabilística mediante la asignación de "boletos" a los procesos.

## COMPARATIVA DE ALGORITMOS DE PLANIFICACIÓN DE CPU



### 2.5.3 Planificación de hilos

La planificación de hilos distingue entre el ámbito de contención del proceso (PCS), donde la biblioteca de hilos planifica los hilos de usuario sobre los procesos ligeros disponibles (común en modelos muchos-a-uno o muchos-a-muchos), y el ámbito de contención del sistema (SCS), donde el núcleo planifica los hilos compitiendo por la CPU con todos los demás hilos del sistema. En sistemas modernos que siguen el modelo uno-a-uno, como Windows, Linux y Solaris, la planificación se realiza a nivel de hilos del núcleo, tratándolos como la unidad básica de ejecución y asignándoles prioridades, afinidad a procesadores y contextos propios.

El estándar POSIX define políticas específicas para la planificación de hilos, incluyendo SCHED\_FIFO (primero en entrar, primero en salir para tiempo real), SCHED\_RR (cíclica para tiempo real) y SCHED\_OTHER (definida por la

implementación, usualmente para tiempo compartido), permitiendo ajustar prioridades y ámbitos mediante atributos en la creación. En implementaciones como Windows NT, el sistema utiliza un esquema de prioridades con 32 niveles, separando clases de prioridad variable para procesos interactivos (donde la prioridad se ajusta dinámicamente) y clases de tiempo real con prioridades fijas para garantizar la respuesta de tareas críticas.

## **UNIDAD 3**

### **Administración de memoria**

#### **3.1 Abstracción de la memoria**

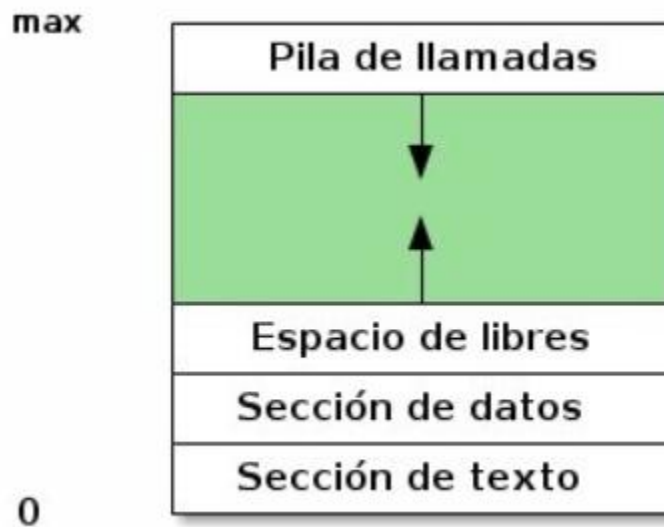
La abstracción de la memoria es fundamental porque, idealmente, los programadores desearían una memoria infinitamente grande, rápida y no volátil, pero la realidad tecnológica impone una jerarquía con registros, caché, memoria RAM y discos. El sistema operativo debe coordinar estos recursos para ofrecer a cada proceso un espacio lógico propio, independiente y protegido, evitando que los procesos interfieran entre sí o accedan a direcciones físicas de manera descontrolada.

Para lograr esto, el sistema operativo y el hardware (como la MMU) colaboran para separar la visión lógica que tiene el programa de la ubicación física real de los datos. Esta abstracción permite la multiprogramación, donde varios procesos residen en memoria simultáneamente, y facilita el uso de técnicas como la reubicación y la memoria virtual, haciendo que el usuario perciba una memoria lineal y contigua aunque físicamente esté fragmentada o dispersa.

##### **3.1.1 Organización de la memoria**

La organización de la memoria se estructura típicamente como una jerarquía piramidal basada en velocidad, costo y capacidad: en la cima están los registros del procesador y la memoria caché, seguidos por la memoria principal (RAM) y finalmente el almacenamiento secundario como discos magnéticos o de estado sólido. A nivel de sistema operativo, la memoria se organiza lógicamente para cada proceso en regiones o segmentos definidos, que incluyen el texto (código del programa), los datos con y sin valor inicial, la pila (stack) para las llamadas a funciones y el montículo (heap) para la memoria dinámica.

Desde una perspectiva física, la memoria se ve como un arreglo lineal de bytes direccionables, y su gestión implica dividir este espacio para alojar tanto al sistema operativo (generalmente en direcciones bajas o altas protegidas) como a los procesos de usuario. En sistemas modernos, esta organización física se maneja mediante marcos de página de tamaño fijo, lo que permite que el espacio lógico de un proceso no necesite ser contiguo físicamente, resolviendo problemas de asignación mediante tablas de paginación gestionadas por el hardware y el sistema operativo.



### 3.1.2 Administración del almacenamiento

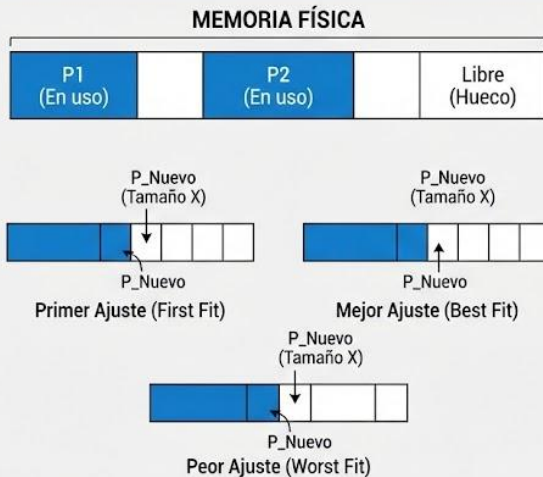
La administración del almacenamiento se refiere a la tarea del sistema operativo de llevar el registro de qué partes de la memoria están en uso y cuáles están libres, asignando espacio a los procesos cuando lo necesitan y recuperándolo cuando terminan. En esquemas de asignación contigua, el administrador debe decidir dónde cargar un proceso en la memoria disponible utilizando algoritmos como el primer ajuste (el primer hueco suficiente), mejor ajuste (el hueco que deje menos sobrante) o peor ajuste (el hueco más grande disponible).

Un desafío crítico en esta administración es la fragmentación: la fragmentación externa ocurre cuando hay suficiente memoria total libre pero está dividida en pequeños huecos no contiguos, mientras que la fragmentación interna sucede cuando se asigna más memoria de la necesaria debido a tamaños de bloque fijos. Para mitigar esto, los administradores de memoria pueden emplear técnicas de compactación para reunir el espacio libre, o utilizar esquemas más avanzados como la paginación que evitan la necesidad de compactación al permitir la asignación no contigua.



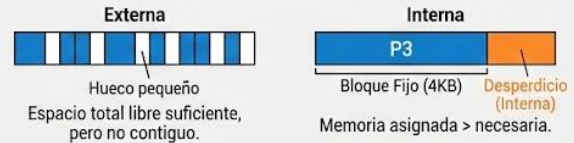
# ADMINISTRACIÓN DEL ALMACENAMIENTO (MEMORIA)

## ASIGNACIÓN CONTIGUA Y ALGORITMOS

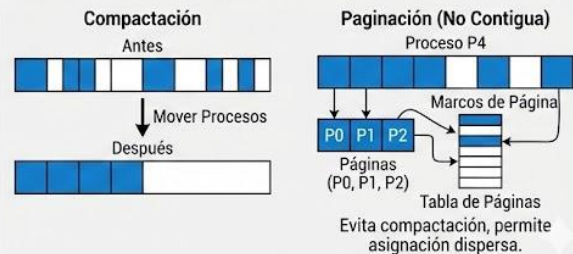


## DESAFÍO: FRAGMENTACIÓN Y SOLUCIONES

### FRAGMENTACIÓN



### SOLUCIONES



### 3.1.3 Memoria de intercambio

La memoria de intercambio, o *swapping*, es una técnica que permite al sistema operativo ejecutar más procesos de los que caben físicamente en la memoria principal moviendo procesos enteros (o páginas) entre la memoria RAM y un almacenamiento secundario rápido, conocido como espacio de swap o respaldo. Cuando la memoria se llena o un proceso está inactivo (bloqueado), el sistema lo "expulsa" (swap out) al disco, liberando su memoria para otros procesos activos; posteriormente, cuando el proceso necesita ejecutarse, se trae de vuelta (swap in).

Aunque el intercambio tradicional movía procesos completos, lo cual era lento debido a la velocidad del disco, los sistemas modernos combinan esto con la memoria virtual para intercambiar solo las páginas necesarias, mejorando la eficiencia. El sistema operativo debe gestionar este espacio en disco, decidiendo si preasignar espacio para todo el proceso al crearlo o asignar espacio en el disco dinámicamente solo cuando es necesario expulsar una página.

### 3.1.4 Manejo de memoria con mapa de bits

En la gestión con mapas de bits, la memoria se divide en unidades de asignación de tamaño fijo (que pueden ir desde unas pocas palabras hasta varios kilobytes), y cada unidad se corresponde con un bit en el mapa: un 0 indica que está libre y un 1 que está ocupada (o viceversa). El tamaño de la unidad de asignación

es una decisión de diseño crucial; si es muy pequeña, el mapa de bits crece considerablemente consumiendo mucha memoria, pero si es muy grande, se desperdicia memoria en el último bloque del proceso debido a la fragmentación interna.

La principal ventaja de este método es que es sencillo de implementar y eficiente en espacio para representar recursos de tamaño fijo, permitiendo visualizar rápidamente el estado global de la memoria. Sin embargo, su mayor desventaja es la velocidad: para encontrar un espacio libre de tamaño, el sistema debe buscar en el mapa una secuencia de bits consecutivos en cero, lo cual es una operación lenta que puede requerir recorrer gran parte del mapa si la memoria está llena o fragmentada

### 3.1.5 Manejo de memoria con listas ligadas

Este método gestiona la memoria manteniendo una lista enlazada de segmentos asignados y libres (huecos), donde cada entrada en la lista especifica si es un proceso (P) o un hueco (H), la dirección de inicio y la longitud del segmento. La lista suele mantenerse ordenada por direcciones de memoria, lo que facilita enormemente la actualización cuando un proceso termina: si un proceso liberado tiene huecos vecinos, estos se pueden fusionar (*coalescing*) para formar un bloque libre más grande, reduciendo la fragmentación externa.

Para asignar memoria, el sistema operativo recorre esta lista aplicando algoritmos como el primer ajuste (*first-fit*) o el mejor ajuste (*best-fit*) para encontrar un hueco adecuado para el nuevo proceso. Aunque el uso de listas, especialmente si son doblemente enlazadas, facilita la fusión de huecos y la gestión dinámica, requiere almacenar punteros y estructuras adicionales, y la búsqueda de un espacio adecuado puede ser lenta si la lista se fragmenta en muchos elementos pequeños.



## 3.2 Memoria virtual

La memoria virtual es una de las técnicas más potentes y fundamentales en los sistemas operativos modernos. Permite que el sistema ejecute procesos que requieren más memoria de la que físicamente está disponible en la memoria RAM.

La memoria virtual es un esquema que abstrae la memoria física (RAM) en una vista lógica uniforme para cada proceso. El objetivo principal es separar la memoria lógica del usuario de la memoria física real.

Esto permite que:

- Los programas puedan ser más grandes que la RAM física.
- El grado de multiprogramación aumente, ya que solo se cargan las partes del programa que se están usando realmente.
- El programador no tenga que preocuparse por las limitaciones de la capacidad del hardware.

### **3.2.1 Paginación**

La paginación es un esquema de gestión de memoria que permite que el espacio de direcciones físicas de un proceso no sea contiguo, eliminando la fragmentación externa y la necesidad de compactación. En este método, la memoria física se divide en bloques de tamaño fijo llamados marcos (frames), y la memoria lógica se divide en bloques del mismo tamaño llamados páginas; cuando un proceso se ejecuta, sus páginas se cargan en cualquier marco de memoria disponible, sea contiguo o no, gestionado por el sistema operativo mediante una lista de marcos libres. El tamaño de la página, definido por el hardware, suele ser una potencia de 2 (típicamente entre 4 KB y 1 GB), lo que facilita la traducción de direcciones lógicas a físicas al dividir la dirección en un número de página y un desplazamiento (offset) dentro de ella.

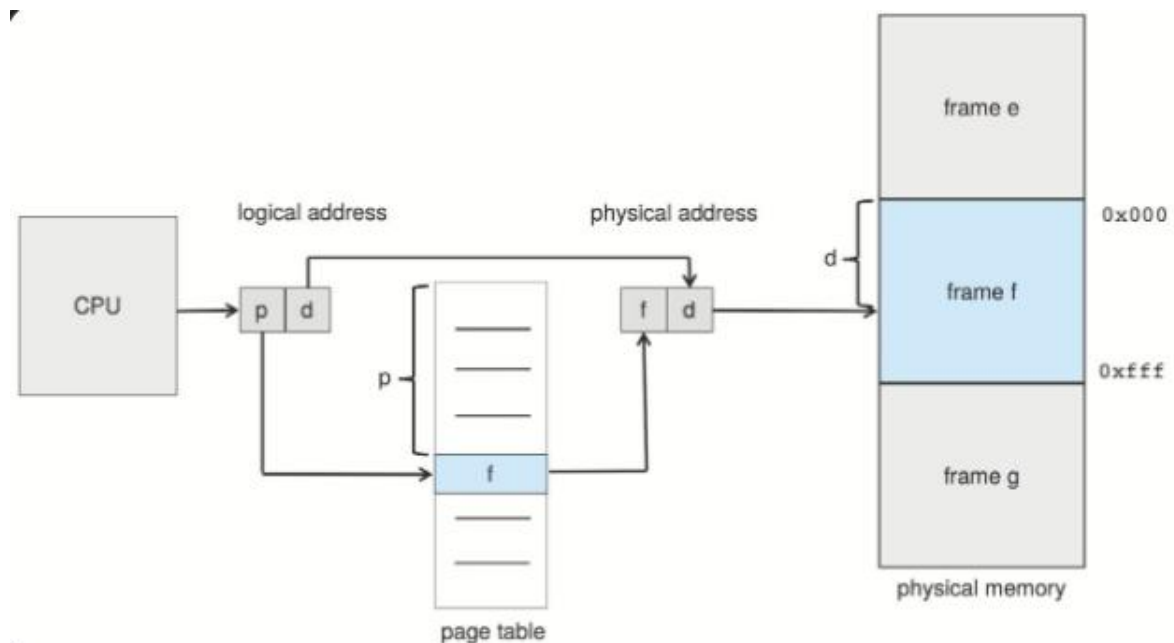
Aunque la paginación resuelve la fragmentación externa, introduce el problema de la fragmentación interna, ya que si un proceso requiere un tamaño de memoria que no es múltiplo exacto del tamaño de página, el último marco asignado no estará completamente lleno. Además, este esquema es la base para la implementación de la paginación por demanda, donde las páginas se cargan en memoria solo cuando son requeridas durante la ejecución; si un proceso intenta acceder a una página que no está en memoria física, el hardware genera una excepción de fallo de página (page fault), lo que activa al sistema operativo para traer la página desde el almacenamiento secundario (disco o swap) a un marco libre.

### **3.2.2 Tablas de página**

La tabla de páginas es la estructura de datos fundamental que utiliza el sistema operativo para mapear las direcciones lógicas (páginas) a direcciones físicas (marcos), permitiendo que el procesador acceda a los datos reales en memoria. Cada proceso tiene su propia tabla de páginas, y cada entrada en esta tabla contiene el número de marco físico correspondiente y bits de control, como el

bit de validez (que indica si la página está en memoria o en disco) y bits de protección (lectura/escritura). Dado que estas tablas pueden ser muy grandes (por ejemplo, en sistemas de 32 o 64 bits), se emplean técnicas como las tablas de páginas multinivel, que dividen la tabla en jerarquías para no tener que mantenerla toda en memoria contigua, o las tablas de páginas invertidas, que tienen una entrada por cada marco físico en lugar de por cada página virtual, ahorrando espacio considerablemente.

Para evitar la penalización de rendimiento que implicaría acceder a la tabla de páginas en memoria RAM por cada instrucción (lo que duplicaría el tiempo de acceso a memoria), el hardware utiliza una memoria caché asociativa de alta velocidad llamada TLB (Translation Lookaside Buffer). La TLB almacena las traducciones de direcciones utilizadas recientemente; cuando el procesador genera una dirección virtual, la MMU (Unidad de Gestión de Memoria) consulta primero la TLB y, si encuentra la traducción (TLB hit), accede directamente al marco físico sin consultar la tabla de páginas en memoria principal, logrando así que el tiempo efectivo de acceso sea casi tan rápido como sin paginación.



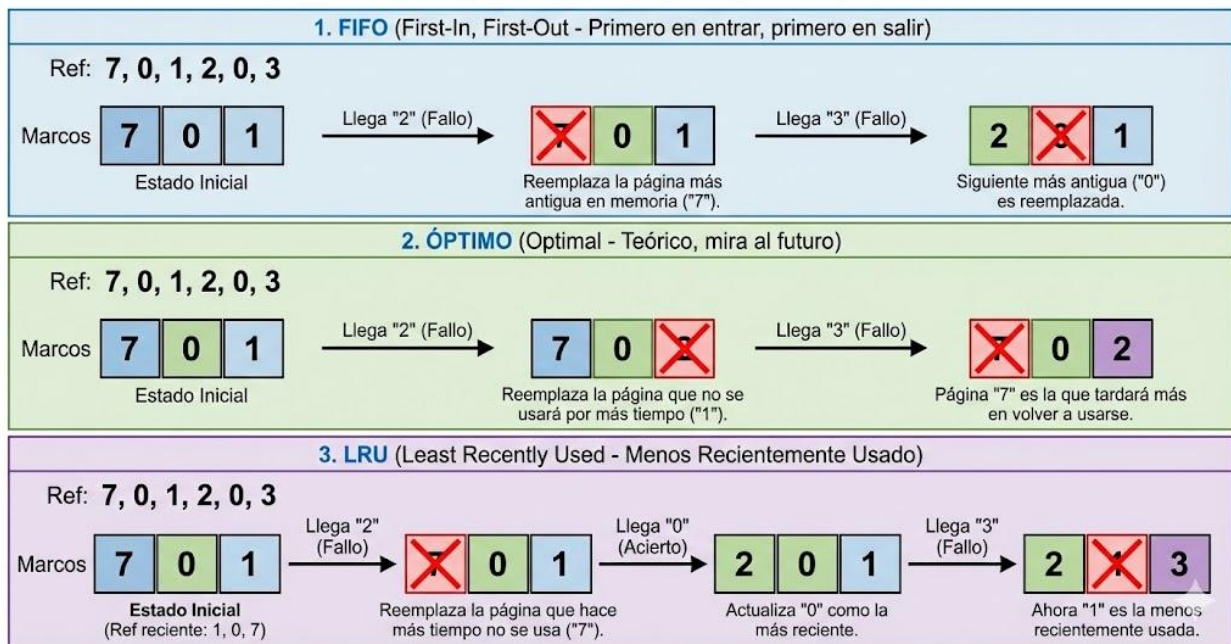
### 3.2.3 Algoritmos de sustitución de páginas

Cuando ocurre un fallo de página y no hay marcos libres disponibles en la memoria física, el sistema operativo debe seleccionar una página existente para desalojarla (o reemplazarla) y hacer espacio para la nueva página, una decisión crítica gobernada por los algoritmos de reemplazo. El objetivo principal es minimizar la tasa de fallos de página a largo plazo; si la página elegida como "víctima" ha sido modificada (tiene el *dirty bit* activo), debe escribirse primero en el disco antes de ser

sobrescrita, lo cual duplica el costo de la operación de E/S. Un algoritmo básico es el FIFO (First-In, First-Out), que reemplaza la página más antigua, aunque su rendimiento puede ser pobre y sufrir la anomalía de Belady, donde aumentar la memoria física paradójicamente incrementa el número de fallos.

Para optimizar el rendimiento, se busca aproximar el algoritmo Óptimo (OPT), que reemplazaría la página que tardará más tiempo en volver a ser usada, aunque este es imposible de implementar en tiempo real porque requiere conocer el futuro. En la práctica, se utilizan aproximaciones como el algoritmo LRU (Least Recently Used), que reemplaza la página que no ha sido usada por más tiempo, basándose en la heurística de que el pasado reciente predice el futuro cercano; dado que LRU puro requiere costoso soporte de hardware, muchos sistemas emplean el algoritmo de Segunda Oportunidad (o Reloj), que usa un bit de referencia para dar una "vida extra" a las páginas usadas recientemente antes de considerarlas para reemplazo.

### COMPARATIVA DE ALGORITMOS DE REEMPLAZO DE PÁGINAS



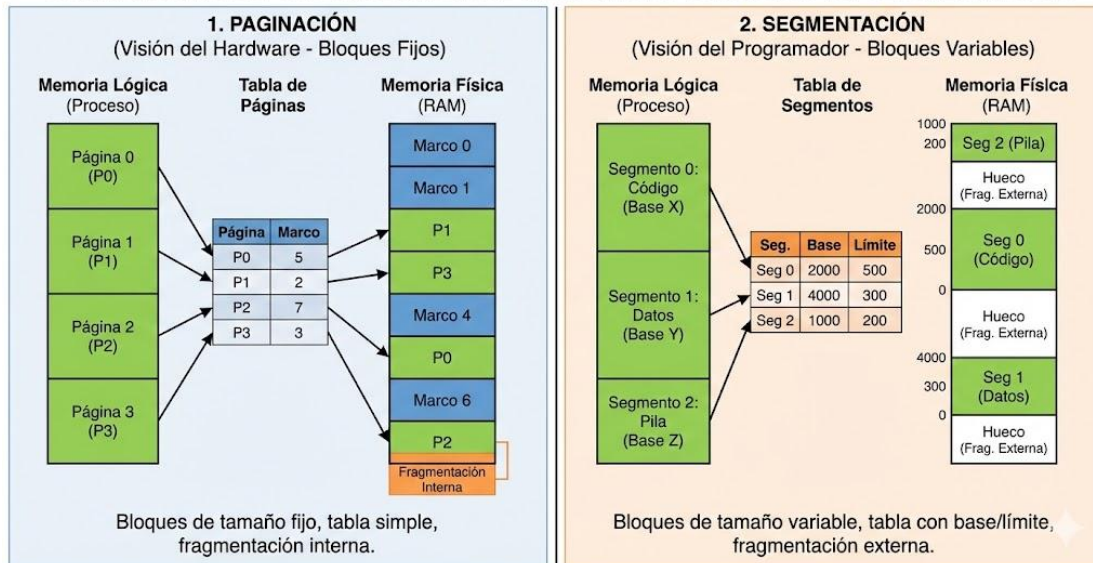
### 3.3 Segmentación

La segmentación es un esquema de administración de memoria que refleja la visión que el programador tiene de un programa. A diferencia de la paginación, que divide la memoria en bloques de tamaño fijo (visión del hardware), la segmentación la divide en unidades lógicas de tamaño variable.

Un programa no es simplemente un arreglo lineal de bytes; es una colección de componentes lógicos como el código principal, funciones, variables globales, la pila

(stack) y el montículo (heap). La segmentación permite que cada uno de estos elementos se trate como un segmento separado con su propio nombre (número) y longitud.

### COMPARATIVA DE ORGANIZACIÓN DE MEMORIA: PAGINACIÓN vs. SEGMENTACIÓN



### 3.3.1 Implementación de la segmentación pura

La implementación de la segmentación pura requiere un soporte de hardware específico para manejar el mapa de memoria de un proceso como una colección de segmentos independientes de tamaño variable, los cuales reflejan la estructura lógica del programa (como código, pila y datos) en lugar de la estructura física de la memoria. Para realizar la traducción de direcciones, la Unidad de Gestión de Memoria (MMU) emplea una tabla de segmentos por proceso; cada entrada en esta tabla almacena un registro base, que señala la dirección física de inicio del segmento, y un registro límite que especifica su longitud exacta. Cuando la CPU genera una dirección lógica (compuesta por número de segmento y desplazamiento), el hardware verifica primero que el desplazamiento se encuentre dentro del rango definido por el límite y, si es válido, lo suma a la dirección base para obtener la dirección física real.

Sin embargo, este esquema presenta desventajas significativas en la administración del almacenamiento físico, principalmente el problema de la fragmentación externa o "tablero de ajedrez". Dado que los segmentos varían en tamaño y se asignan y liberan dinámicamente, la memoria libre termina dividida en muchos huecos pequeños no contiguos que, sumados, podrían ser suficientes para alojar un nuevo proceso, pero individualmente son demasiado pequeños para contener un segmento completo. Para resolver esto, el sistema operativo debe realizar ocasionalmente una compactación de memoria, moviendo los segmentos ocupados

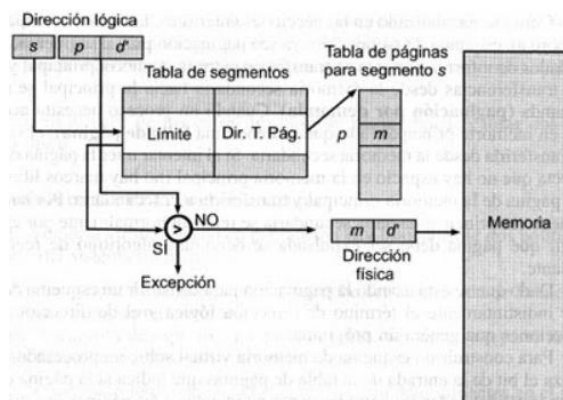


para consolidar el espacio libre en un solo bloque grande, una operación que consume muchos recursos del procesador.

### 3.3.2 Segmentación con paginación

La segmentación con paginación es una técnica híbrida que busca combinar las ventajas lógicas de la segmentación (protección y organización modular) con la eficiencia física de la paginación, eliminando la necesidad de que los segmentos ocupen zonas contiguas de memoria. En este diseño, cada segmento se divide internamente en páginas de tamaño fijo; en consecuencia, la tabla de segmentos ya no contiene la dirección base física del segmento, sino que apunta a una tabla de páginas específica para ese segmento. Esto permite que las páginas que componen un segmento se distribuyan de manera no contigua por toda la memoria física, eliminando por completo la fragmentación externa y la necesidad de compactación, aunque incrementa la complejidad del hardware y el tiempo de acceso debido a las múltiples búsquedas en tablas.

Un ejemplo prominente de esta arquitectura se encuentra en los procesadores Intel (como el Pentium o IA-32), donde la gestión de memoria se realiza en dos etapas secuenciales: segmentación y luego paginación. Primero, la unidad de segmentación toma la dirección lógica y, sumando el desplazamiento a la base del segmento, genera una dirección lineal de 32 bits. Si la paginación está habilitada, esta dirección lineal es interpretada por la unidad de paginación (utilizando un directorio de páginas y tablas de páginas) para finalmente traducirla a la dirección del marco físico, permitiendo así gestionar segmentos que pueden ser incluso más grandes que la memoria física disponible.

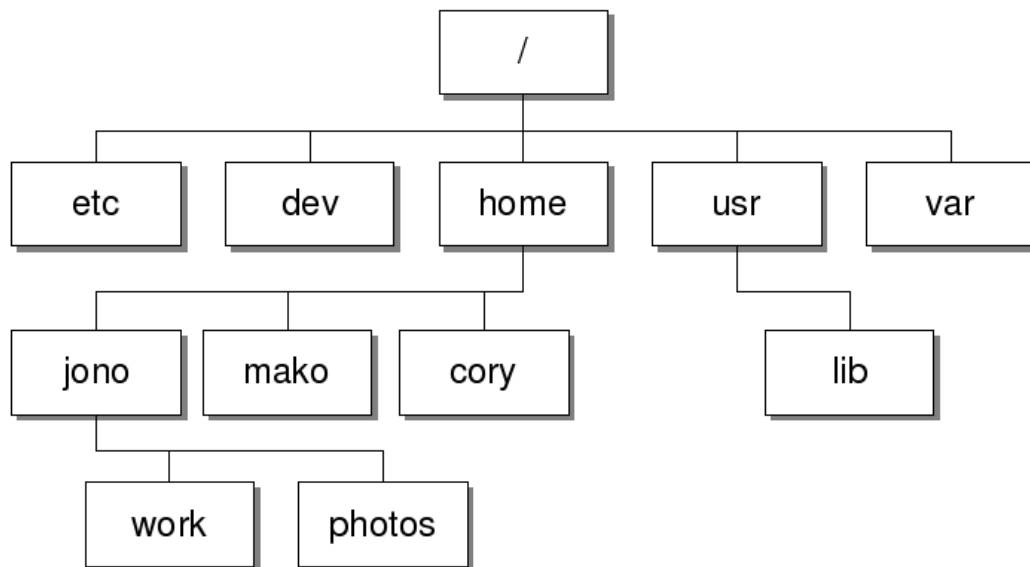


## UNIDAD 4

### Sistema de archivos

Un sistema de archivos es un mecanismo de abstracción que permite a los usuarios y programas almacenar y acceder a datos en dispositivos de almacenamiento sin conocer los detalles físicos del hardware. Proporciona una vista lógica uniforme de la información, definiendo archivos como unidades de almacenamiento lógico y directorios para organizarlos. Sus características clave incluyen una estructura jerárquica, tratamiento consistente de datos, y la capacidad de crear, borrar, leer y escribir archivos, protegiendo la información contenida.

Se utiliza mediante llamadas al sistema (como open, write, read) que permiten manipular los datos, gestionando internamente la correspondencia entre la visión lógica del usuario y los bloques físicos del disco. Es fundamental en todos los sistemas operativos de propósito general (como Windows, Linux o UNIX) para gestionar el almacenamiento secundario, permitiendo la persistencia de datos de aplicaciones, bases de datos y el propio sistema operativo tras el apagado del equipo



#### 4.1 Estructura del sistema de archivos

La estructura de un sistema de archivos define la organización lógica y física de la información dentro de un dispositivo de almacenamiento, residiendo habitualmente dentro de particiones o volúmenes que actúan como entidades lógicas independientes. Esta arquitectura en disco se compone típicamente de elementos fundamentales como el bloque de arranque (boot block), necesario para

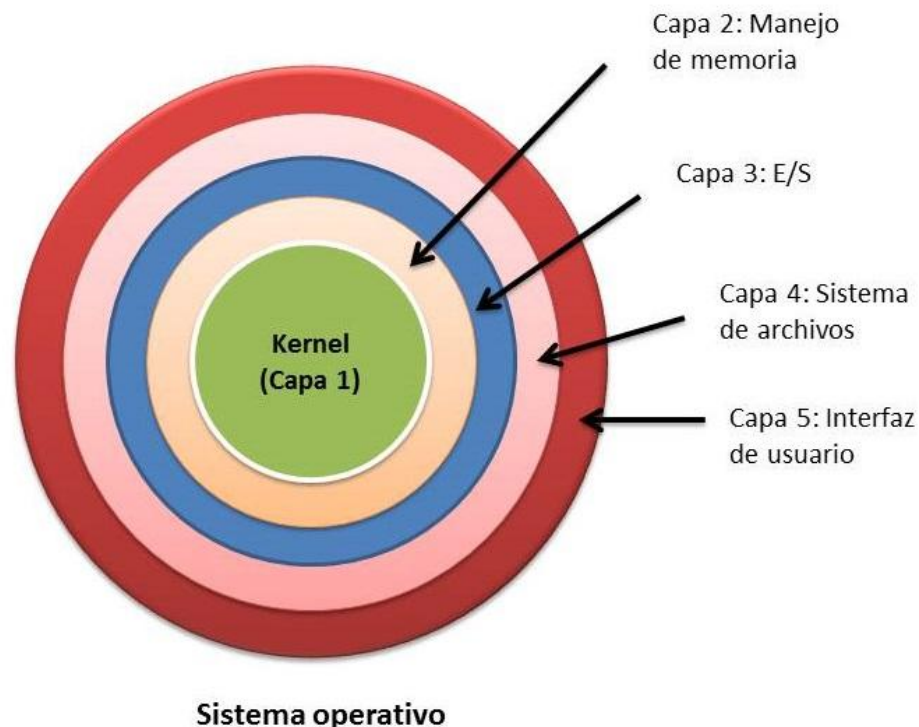


iniciar el sistema operativo; el superbloque, que contiene parámetros críticos como el número mágico, el tamaño del sistema y listas de espacio libre; los descriptores de archivos (como los nodos-i en UNIX o la MFT en Windows NT), que almacenan metadatos y la ubicación de los bloques; y finalmente los bloques de datos o agrupaciones donde se almacena el contenido real de los archivos.

El sistema operativo utiliza esta estructura cargando en memoria copias de los metadatos esenciales, como el superbloque, durante el proceso de montaje para poder traducir las rutas de acceso lógicas de los archivos a sus direcciones físicas en el disco. Esta organización se aplica fundamentalmente mediante herramientas de administración que dan formato a las particiones, inicializando las tablas de gestión y mapas de bits necesarios para que las aplicaciones y usuarios puedan crear, leer y organizar archivos en una jerarquía de directorios coherente sin interactuar directamente con los sectores físicos del dispositivo

#### 4.1.1 Capa del sistema de archivos

El diseño del sistema de archivos se organiza frecuentemente en una arquitectura de capas o niveles funcionales para gestionar la complejidad y minimizar la duplicación de código, donde cada nivel utiliza las características de los inferiores para crear nuevas funcionalidades. En la base de esta jerarquía se encuentra el control de E/S (manejadores de dispositivos), que gestiona la



transferencia física de información entre la memoria y el disco; sobre este se sitúa el sistema de archivos básico, encargado de emitir órdenes genéricas de lectura y escritura de bloques físicos y gestionar los buffers de memoria. Los niveles superiores, como el módulo de organización de archivos y el sistema de archivos lógico, se abstraen de los detalles físicos para traducir direcciones lógicas, administrar el espacio libre y gestionar los metadatos (como los inodos o bloques de control de archivo) y la estructura de directorios.

## **4.2. Implementación**

### **4.2.1 De archivos**

La implementación de archivos se centra en el desafío de asignar espacio en el almacenamiento secundario y realizar el seguimiento de los bloques de datos asociados a cada archivo lógico. Los sistemas operativos emplean diversos métodos de asignación, siendo los principales la asignación contigua, que requiere bloques consecutivos y sufre de fragmentación externa; la asignación ligada, donde cada bloque apunta al siguiente; y la asignación indexada, que utiliza una estructura para listar los punteros a los bloques de datos. En sistemas como UNIX y Windows NT, se favorecen los esquemas de asignación no contigua mediante índices multinivel, ya que eliminan la necesidad de compactar el disco y permiten que los archivos crezcan dinámicamente mientras exista espacio libre disponible en el dispositivo.

Para gestionar esta estructura, el sistema operativo utiliza una estructura de datos interna, comúnmente llamada nodo-i (en UNIX) o Bloque de Control de Archivo (FCB), que actúa como un descriptor que contiene los atributos del archivo y el mapa de sus bloques físicos. En el caso de UNIX, el nodo-i almacena punteros directos a los primeros bloques de datos y punteros indirectos (simples, dobles y triples) que permiten direccionar archivos de gran tamaño sin que la tabla de índices ocupe excesiva memoria para archivos pequeños. Esta separación entre la descripción interna del archivo y su nombre permite una gestión eficiente y flexible del almacenamiento, donde los metadatos residen en estas estructuras protegidas y no directamente en las entradas del directorio

### **4.2.2 De directorios**

La implementación de directorios se basa en tratarlos como archivos especiales cuyo contenido es una tabla de entradas que relacionan nombres de archivos con sus descriptors internos (como el número de nodo-i). La estructura interna de estas tablas afecta significativamente la eficiencia; la implementación

más simple es una lista lineal de nombres, que es fácil de programar, pero ineficiente para búsquedas en directorios grandes, mientras que sistemas más avanzados utilizan tablas hash o árboles B para acelerar la localización de archivos y la inserción de nuevas entradas. Además, los directorios suelen contener entradas especiales como "." (directorío actual) y ".." (directorío padre) para facilitar la navegación relativa dentro de la jerarquía.

En cuanto al contenido de cada entrada del directorio, existen dos enfoques principales de diseño: almacenar todos los atributos del archivo (tipo, tamaño, fechas) directamente en la entrada del directorio, como hace MS-DOS, o almacenar únicamente el nombre y un puntero al descriptor del archivo, como ocurre en UNIX. El enfoque de UNIX, que almacena solo el nombre y el número de nodo-i en el directorio, ofrece ventajas significativas, como mantener el tamaño del directorio pequeño y permitir que un mismo archivo tenga múltiples nombres (enlaces) en diferentes ubicaciones sin duplicar sus metadatos o contenido físico.

#### **4.2.3 Archivos compartidos**

La implementación de archivos compartidos se logra permitiendo que los directorios formen un grafo acíclico en lugar de un árbol estricto, lo que posibilita que un mismo archivo aparezca en diferentes directorios simultáneamente. El método más común para esto es el uso de enlaces físicos (hard links), donde múltiples entradas de directorio apuntan al mismo nodo-i o descriptor físico; para gestionar esto correctamente, el sistema mantiene un contador de enlaces dentro del nodo-i, y el archivo real y sus datos solo se eliminan del disco cuando este contador llega a cero, asegurando que el archivo persista mientras exista al menos una referencia a él.

Una alternativa flexible es el uso de enlaces simbólicos (soft links o accesos directos), que se implementan como un tipo de archivo especial cuyo contenido es la ruta de acceso al archivo original. A diferencia de los enlaces físicos, los enlaces simbólicos pueden hacer referencia a archivos en sistemas de archivos distintos o a directorios, y no evitan que el archivo original sea borrado; si el archivo destino se elimina, el enlace simbólico permanece, pero se convierte en un enlace roto o "huérfano" que apunta a una ubicación inexistente.

## 4.3 Métodos de asignación de espacio

### 4.3.1 Contigua

La asignación contigua requiere que cada archivo ocupe un conjunto de bloques consecutivos en el dispositivo de almacenamiento, definidos por la dirección del bloque inicial y la longitud total del archivo en bloques. Este método es extremadamente sencillo de implementar y ofrece un rendimiento excelente tanto para el acceso secuencial como para el directo, ya que minimiza el movimiento de las cabezas del disco al mantener los datos físicamente adyacentes; por esta razón, sigue siendo utilizado en medios de solo lectura como CD-ROM y DVD donde los tamaños de los archivos se conocen de antemano y no cambian.

Sin embargo, este esquema presenta problemas significativos en dispositivos de lectura y escritura debido a la fragmentación externa, ya que el espacio libre se divide en pequeños huecos que pueden no ser suficientes para alojar nuevos archivos grandes, aunque la suma total del espacio libre sea adecuada. Además, resulta difícil gestionar el crecimiento de los archivos: si un archivo necesita expandirse y no hay espacio contiguo adyacente disponible, el sistema operativo debe mover el archivo completo a un nuevo hueco más grande o compactar el disco, operaciones que son costosas en términos de tiempo y recursos.



### 4.3.2 Ligada

La asignación ligada resuelve los problemas de la asignación contigua tratando a cada archivo como una lista enlazada de bloques de disco, donde cada bloque contiene un puntero al siguiente y el directorio solo necesita almacenar la dirección del primero. Este método elimina la fragmentación externa, ya que cualquier bloque libre en el disco puede ser utilizado para satisfacer una solicitud de espacio, y permite que los archivos crezcan dinámicamente sin necesidad de compactar el disco.

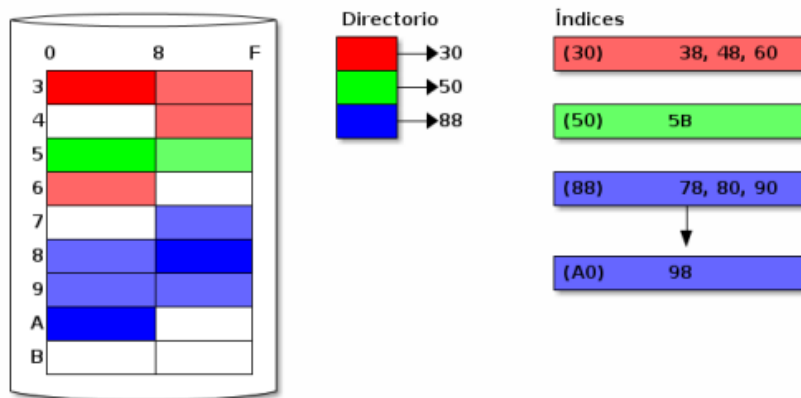
No obstante, la asignación ligada pura es ineficiente para el acceso directo o aleatorio, puesto que, para encontrar un bloque específico en la mitad de un archivo, el sistema debe leer y recorrer secuencialmente todos los bloques anteriores siguiendo la cadena de punteros. Una variación importante que mitiga este defecto es el uso de una Tabla de Asignación de Archivos (FAT), utilizada por MS-DOS y Windows, donde los punteros se extraen de los bloques y se almacenan en una tabla centralizada en memoria; esto permite seguir la cadena de bloques sin acceder al disco, acelerando significativamente las búsquedas aleatorias.



#### 4.3.3 Indexada

La asignación indexada soluciona la dispersión de punteros de la asignación ligada reuniendo todos los punteros a los bloques de datos en un solo bloque llamado bloque índice o índice. Este esquema permite el acceso directo eficiente a cualquier parte del archivo sin sufrir de fragmentación externa, ya que el sistema simplemente consulta la entrada correspondiente en el bloque índice para obtener la dirección del bloque deseado, funcionando de manera análoga a la paginación en memoria.

El principal desafío de este método es gestionar el tamaño del bloque índice: si es muy grande se desperdicia espacio para archivos pequeños, y si es muy pequeño no alcanza para archivos grandes. Para resolver esto, sistemas como UNIX utilizan un esquema combinado en sus nodos-i, que incluye punteros directos para los primeros bloques de datos y punteros de indirección (simple, doble y triple) que apuntan a otros bloques de índices, permitiendo así direccionar archivos desde muy pequeños hasta extremadamente grandes sin una sobrecarga excesiva.



#### 4.3.4 Administración del espacio en disco

Para reutilizar el espacio liberado por archivos eliminados y asignar espacio a nuevos archivos, el sistema operativo debe mantener un registro de los bloques disponibles mediante una lista de espacio libre. Un método común es el mapa de bits o vector de bits, donde cada bloque se representa con un bit (1 si está libre, 0 si está ocupado); este enfoque es eficiente para encontrar grupos de bloques libres contiguos y es simple de implementar, aunque puede consumir una cantidad considerable de memoria principal para discos de gran capacidad.

Otra técnica consiste en utilizar una lista enlazada de bloques libres, donde cada bloque libre apunta al siguiente, o variantes como la agrupación (guardar las direcciones de bloques libres en el primer bloque libre) y el conteo (guardar la dirección del primer bloque libre y la cantidad de bloques contiguos libres que le siguen). Sistemas de archivos más avanzados, como ZFS, evitan los mapas de bits tradicionales debido a su ineficiencia en discos masivos y utilizan árboles balanceados o "mapas de espacio" basados en registros (logs) para gestionar la asignación y liberación de grandes extensiones de almacenamiento de manera transaccional.

#### 4.4 Sistemas de archivos estructurados

##### 4.4.1 Por bitácoras

Los sistemas de archivos estructurados por bitácora (Log-Structured File Systems o LFS) se diseñaron con el objetivo de mejorar tanto el rendimiento como la consistencia del almacenamiento de datos. En este tipo de organización, toda la información, incluyendo datos y metadatos, se escribe de manera secuencial en una estructura continua similar a un registro o bitácora, lo cual busca superar los cuellos de botella de entrada/salida (I/O) que sufren los sistemas tradicionales debido a las escrituras aleatorias y la fragmentación.

Este enfoque ha mantenido su relevancia y ha evolucionado con la aparición de nuevas tecnologías de hardware; por ejemplo, se ha identificado que los principios de diseño de los sistemas estructurados por bitácora son particularmente aplicables y beneficiosos para las unidades de estado sólido (SSD). Además de la eficiencia en la escritura, la implementación de estos sistemas facilita la recuperación de datos y la verificación de la integridad del sistema de archivos, ya que la estructura secuencial del registro simplifica la reconstrucción del estado del sistema tras un fallo.

#### **4.4.2 Por diario.**

Los sistemas de archivos por diario (*journaling*) se basan en el concepto de actualización atómica o indivisible para garantizar que las operaciones se completen satisfactoriamente o no tengan ningún efecto en absoluto, protegiendo así la integridad de los datos ante fallos. Para lograr esto, el sistema utiliza archivos de registro o bitácoras donde se anotan las transacciones en un estado provisional; solo cuando el sistema confirma que la operación es segura, esta se convierte en definitiva, permitiendo descartar acciones incompletas si ocurre un error durante el proceso.

La principal ventaja de este mecanismo es su capacidad para asegurar que el sistema de archivos se mantenga consistente incluso después de una interrupción abrupta, como un corte de energía, evitando la corrupción de metadatos crítica. Al reiniciar el sistema, en lugar de realizar un escaneo completo y lento de todo el volumen para buscar errores, el sistema operativo puede simplemente consultar el diario para completar o deshacer las operaciones pendientes, asegurando la reversibilidad y reduciendo drásticamente el tiempo de recuperación.

### **4.5 Optimización del sistema de archivos**

#### **4.5.1 Recuperación**

La recuperación en los sistemas de archivos se centra en mecanismos para restaurar la disponibilidad y corrección de los datos tras fallos físicos o lógicos, utilizando técnicas como la replicación, que mantiene copias de los datos en diferentes servidores o dispositivos. En modelos de gestión de réplicas, si un servidor maestro o primario falla, las operaciones de lectura y escritura pueden ser gestionadas por las copias restantes, lo que elimina puntos únicos de fallo y asegura que la información crítica no se pierda.

A nivel de estructura lógica, la recuperación se apoya en la redundancia de metadatos, como el mantenimiento de copias adicionales de las tablas de asignación (como en FAT) o el uso de listas doblemente ligadas que permiten recorrer la estructura en ambos sentidos para detectar y corregir inconsistencias. Además, los sistemas que implementan atomicidad poseen la capacidad intrínseca de reversibilidad, lo que permite deshacer operaciones que no concluyeron correctamente, devolviendo el sistema a un estado estable conocido y evitando la presencia de datos corruptos o parciales.

#### **4.5.2 Consistencia**

La consistencia del sistema de archivos implica asegurar que la visión lógica de los archivos coincida con el estado físico de los bloques en el disco, evitando anomalías como bloques reutilizados (asignados a dos archivos simultáneamente) o bloques perdidos que no pertenecen a ningún archivo pero figuran como ocupados. Las herramientas de verificación del sistema analizan las listas de recursos y mapas de bits para detectar situaciones como archivos cruzados, donde dos entradas de directorio apuntan erróneamente a la misma cadena de datos, o cadenas huérfanas que deben ser recuperadas o liberadas.

Para mantener esta coherencia, algunos sistemas de archivos modernos utilizan sumas de comprobación (*checksums*) tanto para datos como para metadatos, lo que permite detectar corrupciones de integridad de manera proactiva. Dado que mantener una consistencia estricta en todo momento puede afectar el rendimiento, algunos diseños permiten inconsistencias no destructivas temporalmente, exigiendo que solo las actualizaciones críticas de metadatos se realicen de manera síncrona para prevenir daños estructurales graves en el sistema de archivos.

#### **4.5.3 Rendimiento**

La optimización del rendimiento se logra principalmente mediante la implementación de una memoria caché de bloques (block cache o buffer cache), la cual almacena en memoria principal los bloques de disco accedidos frecuentemente para reducir el número de operaciones lentas de lectura y escritura física. La gestión eficiente de esta caché suele emplear algoritmos de reemplazo como LRU (Least Recently Used) y aprovecha el principio de proximidad referencial, el cual establece que los programas tienden a acceder a datos que están espacial o temporalmente cercanos a los que ya han utilizado.



Además del almacenamiento en caché, el rendimiento se mejora seleccionando el método de acceso adecuado (secuencial o aleatorio) según el tipo de operación y mediante el uso de paralelismo en el hardware de almacenamiento. Técnicas como la distribución de datos en bandas (*striping*) en arreglos RAID permiten que múltiples unidades procesen datos simultáneamente, aumentando el caudal de transferencia para accesos pequeños mediante el balanceo de carga y reduciendo el tiempo de respuesta para accesos grandes al leer o escribir en varios discos a la vez.

## **UNIDAD 5**

### **Dispositivos de entrada y salida**

#### **5.1 Principios del hardware de E/S**

El hardware de entrada y salida se organiza generalmente alrededor de una arquitectura de bus que conecta la unidad central de procesamiento (CPU) y la memoria principal con una variedad de dispositivos periféricos a través de componentes electrónicos intermedios. En este esquema, cada dispositivo o grupo de dispositivos cuenta con un controlador (o adaptador) que gestiona la comunicación de bajo nivel y se conecta al bus del sistema o un bus de expansión (como PCI o PCIe), actuando como una interfaz que oculta las complejidades electromecánicas del dispositivo real al resto del sistema.

Para gestionar estas operaciones, el procesador se comunica con los controladores mediante la lectura y escritura de patrones de bits en registros específicos ubicados en el controlador: registros de entrada de datos, salida de datos, estado y control. Esta comunicación puede realizarse mediante instrucciones especiales de E/S que dirigen datos a puertos específicos, o mediante E/S mapeada en memoria (memory-mapped I/O), donde los registros del dispositivo se asignan a direcciones del espacio de memoria principal, permitiendo el uso de instrucciones estándar de transferencia de datos para interactuar con los periféricos.

##### **5.1.1 Dispositivos de E/S**

Los dispositivos de E/S se clasifican típicamente en dos categorías fundamentales según su unidad de transferencia y características de acceso: dispositivos de bloques y dispositivos de caracteres. Los dispositivos de bloques, como los discos duros, CD-ROMs y cintas, almacenan la información en bloques de tamaño fijo (por ejemplo, 512 bytes a 32 KB) y permiten leer o escribir cada bloque de manera independiente, siendo posible direccionarlos individualmente para realizar accesos aleatorios.

Por otro lado, los dispositivos de caracteres, como teclados, ratones, impresoras e interfaces de red, envían o aceptan un flujo de caracteres sin una estructura de bloques definida y, generalmente, no son direccionables, lo que significa que no soportan operaciones de búsqueda (seek) para moverse a una posición específica. Esta clasificación permite al sistema operativo diseñar interfaces de software uniformes, aunque existen dispositivos como los relojes que no encajan perfectamente en ninguna de estas categorías y generan interrupciones a intervalos definidos.



### 5.1.2 Controladores de dispositivos

Los controladores de dispositivos son los componentes electrónicos (tarjetas de circuitos o chips integrados) que actúan como intermediarios entre el bus de la computadora y el dispositivo físico mecánico. Su función principal es convertir el flujo de bits serial que proviene del dispositivo (como la señal magnética de un disco) en un bloque de bytes comprensible para la CPU, realizar corrección de errores (ECC) y almacenar temporalmente los datos en una memoria interna o *buffer* antes de transferirlos a la memoria principal.

Desde la perspectiva del sistema operativo, el controlador es la entidad con la que se interactúa, no el dispositivo físico en sí; el sistema operativo envía comandos al controlador (como "leer sector") escribiendo en sus registros de control y verifica el resultado leyendo los registros de estado. Los controladores modernos pueden ser muy sofisticados, conteniendo sus propios microprocesadores y memoria caché para optimizar el rendimiento, permitiendo incluso solapar operaciones de búsqueda en múltiples unidades simultáneamente.

### 5.1.3 Interrupciones

Las interrupciones son el mecanismo de hardware que permite a los dispositivos notificar a la CPU que han completado una operación o que requieren atención, evitando que el procesador tenga que realizar una "espera activa" o sondeo (*polling*) constante e ineficiente. Cuando un controlador de dispositivo está listo (por ejemplo, ha terminado de leer un bloque de disco), envía una señal eléctrica a través de una línea de petición de interrupción (IRQ) al chip controlador de interrupciones, el cual a su vez señala a la CPU; esto provoca que la CPU detenga su ejecución actual, guarde su estado y salte a una dirección de memoria específica (vector de interrupción) para ejecutar una rutina de servicio de interrupción.

Los sistemas modernos implementan sistemas de interrupciones sofisticados que incluyen prioridades (permitiendo que interrupciones urgentes interrumpen a otras de menor prioridad) y la capacidad de enmascarar (ignorar temporalmente) ciertas interrupciones durante secciones críticas del código del sistema operativo. Al finalizar la rutina de servicio, la CPU restaura el estado guardado y reanuda la ejecución del proceso interrumpido, lo que es fundamental para soportar la multiprogramación y la concurrencia entre la CPU y los dispositivos de E/S.

### 5.1.4 Acceso de memoria directo (DMA)

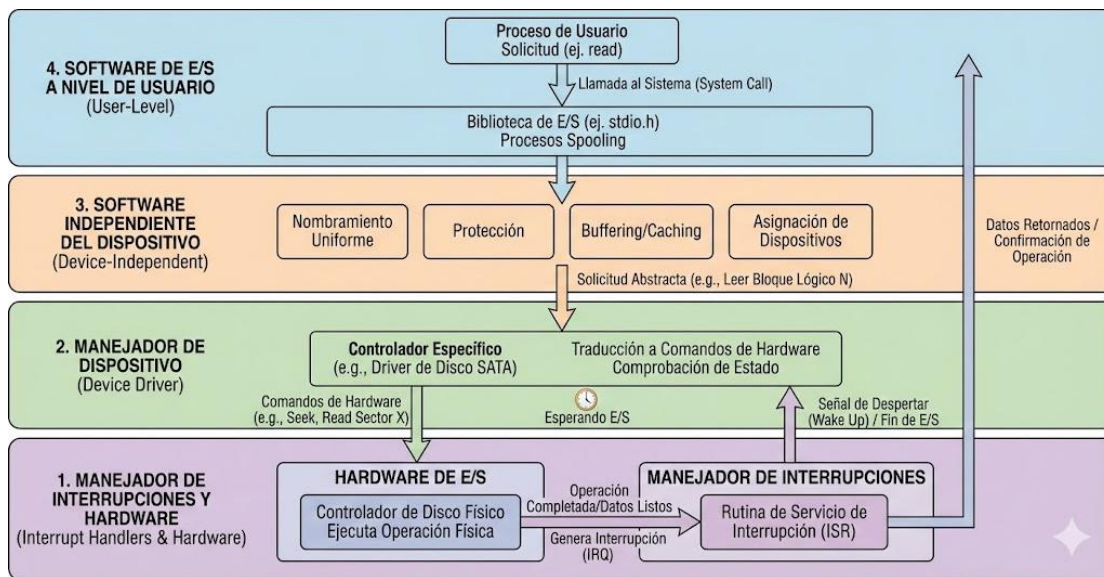
El Acceso Directo a Memoria (DMA) es una técnica utilizada para liberar a la CPU de la carga de transferir grandes volúmenes de datos bit a bit entre los dispositivos de E/S y la memoria principal, una tarea conocida como E/S programada (PIO) que desperdicia muchos ciclos de procesador. Para usar DMA, la CPU programa un controlador de DMA específico indicándole las direcciones de memoria origen y destino, y la cantidad de bytes a transferir; una vez configurado, el controlador de DMA toma el control del bus y gestiona la transferencia de datos directamente hacia o desde la memoria.

Durante la transferencia, la CPU puede continuar ejecutando otras instrucciones, aunque puede experimentar una ligera ralentización si necesita acceder al bus de memoria al mismo tiempo que el controlador DMA ("robo de ciclos"). Cuando la transferencia del bloque completo finaliza, el controlador DMA genera una única interrupción para informar a la CPU, reduciendo drásticamente la sobrecarga del sistema en comparación con la generación de una interrupción por cada byte transferido.

## 5.2 Principios del software de E/S

Los principios del software de Entrada/Salida (E/S) se centran en un concepto clave: la capas y la abstracción. El objetivo es organizar el software de tal manera que el sistema operativo sea modular y que los programas de usuario no tengan que conocer los detalles físicos de cada dispositivo.

El diseño del software de E/S se basa en la idea de que el software debe ser independiente del dispositivo, permitiendo que el kernel se comunice con el hardware sin importar si es un disco duro, un teclado o una impresora.

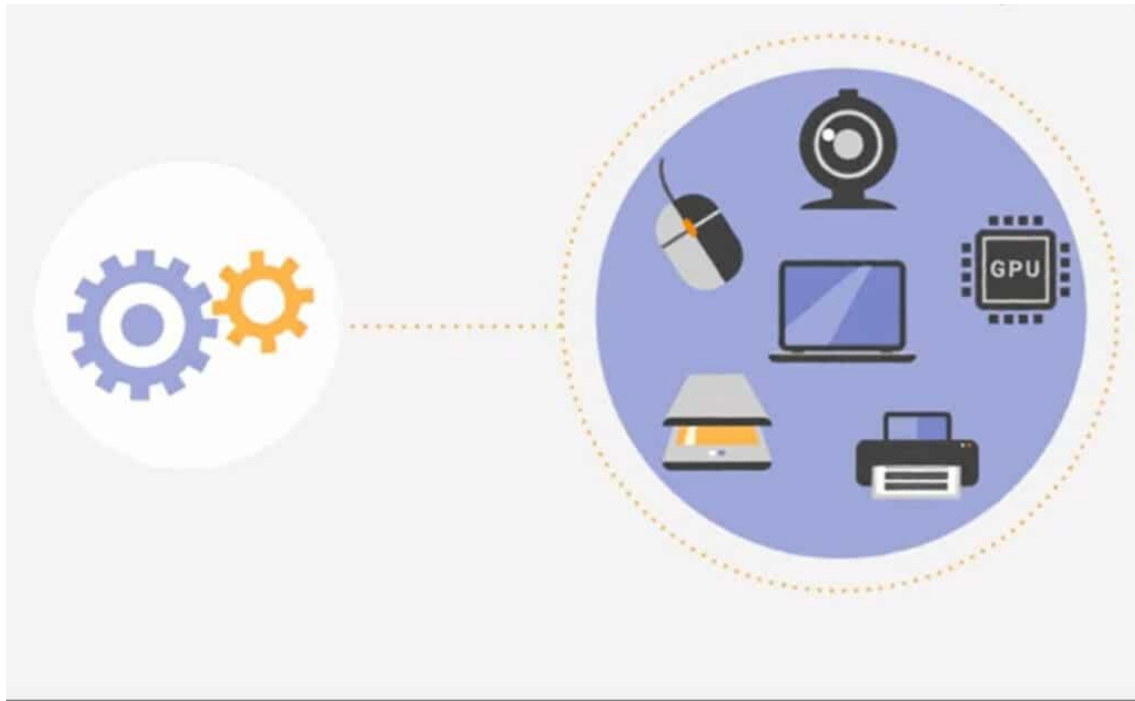


### 5.2.1 Objetivos del software de E/S

El objetivo fundamental del diseño del software de E/S es lograr la independencia del dispositivo, lo que significa que los programas deben poder acceder a cualquier dispositivo de E/S sin necesidad de especificarlo por adelantado ni modificar el código para cada tipo de hardware. Esto se complementa con el objetivo de nombrado uniforme, donde los archivos y dispositivos se direccionan mediante cadenas o enteros (como nombres de ruta en sistemas UNIX: /usr/ast/backup) que no dependen de la máquina física subyacente, permitiendo que el sistema operativo mapee estos nombres lógicos a los controladores adecuados de forma transparente para el usuario.

Otro objetivo crítico es el manejo de errores y la optimización mediante el almacenamiento intermedio (buffering). Los errores deben gestionarse lo más cerca posible del hardware; si un controlador descubre un error de lectura, debe intentar corregirlo él mismo (por ejemplo, reintentando la operación) y solo informar a las capas superiores si no puede solucionarlo. Simultáneamente, el software debe gestionar buffers para desacoplar la velocidad de producción de datos de la de

consumo, evitando desbordamientos o esperas innecesarias debido a las diferencias de velocidad entre dispositivos, y ofrecer una interfaz sencilla que oculte si las transferencias son síncronas (bloqueantes) o asíncronas.



### 5.2.2 E/S programadas

En la técnica de E/S programada, la transferencia de datos entre el procesador y el periférico se realiza mediante la ejecución continua de instrucciones de E/S por parte de la CPU, la cual es responsable de extraer o enviar cada dato. El procesador ejecuta un bucle de espera activa (*busy waiting*) o sondeo (*polling*), consultando repetidamente los registros de estado del controlador hasta que este indica que está listo para recibir o entregar un dato.

Este enfoque es sencillo de implementar, pero conlleva una gran desventaja en términos de eficiencia, ya que el procesador no puede realizar ningún otro trabajo útil mientras espera a que el dispositivo lento complete su operación, desperdiciando ciclos de CPU valiosos. Sin embargo, la E/S programada sigue siendo la técnica de elección en ciertos sistemas de tiempo real o embebidos donde la velocidad de E/S es tan alta que la sobrecarga de gestionar interrupciones sería prohibitiva, o en sistemas dedicados muy simples.

### **5.2.3 E/S manejadas por interrupciones**

Para superar las ineficiencias de la espera activa, esta técnica permite que la CPU envíe una orden de E/S al controlador del dispositivo y continúe ejecutando otros procesos, logrando así concurrencia entre la E/S y el procesador. Cuando el dispositivo ha completado su tarea o tiene datos listos, el controlador envía una señal de hardware (interrupción) a la CPU, lo que provoca que esta detenga su ejecución actual, guarde su estado y salte a una rutina de servicio de interrupción para gestionar la transferencia de datos o finalizar la operación.

Desde la perspectiva del software del sistema operativo, este mecanismo permite que el controlador del dispositivo bloquee al proceso que solicitó la E/S (por ejemplo, ejecutando una operación down en un semáforo), liberando la CPU para otros trabajos. La rutina de interrupción actúa entonces como el evento que desbloquea al conductor (driver) una vez finalizada la operación física, ocultando los detalles complejos de la interrupción en las profundidades del sistema operativo para presentar una interfaz más limpia a las capas superiores.

### **5.2.4 E/S usando DMA**

El Acceso Directo a Memoria (DMA) se utiliza para liberar a la CPU de la carga de transferir datos byte a byte (E/S programada) en dispositivos de alta velocidad y grandes volúmenes de datos, como los discos duros. En este esquema, la CPU programa un controlador de DMA específico con la dirección de memoria, el contador de bytes y el tipo de operación, y luego delega el control del bus de memoria a este controlador.

Una vez configurado, el controlador de DMA se encarga de transferir el bloque completo de datos directamente entre el periférico y la memoria principal sin intervención del procesador central, el cual puede seguir ejecutando otras instrucciones (aunque puede sufrir un ligero retraso por el robo de ciclos de bus). La CPU solo recibe una interrupción al final de la transferencia de todo el bloque, en lugar de una por cada byte, lo que mejora drásticamente el rendimiento global del sistema al reducir la sobrecarga de interrupciones.

## **5.3 Capas de software**

El software de E/S se organiza en cuatro capas principales. Cada capa tiene una interfaz bien definida con las capas adyacentes y realiza funciones específicas para transformar una solicitud de usuario en una operación de hardware.

- I. **Manejadores de Interrupciones (Interrupt Handlers):** Es la capa más cercana al hardware. Su función principal es ocultar las interrupciones al resto del sistema operativo.
- II. **Manejadores de Dispositivos (Device Drivers):** Cada tipo de dispositivo conectado al sistema (ratón, disco, tarjeta de red) tiene su propio driver. Es el único software que conoce los detalles específicos de los registros y protocolos de ese hardware.
- III. **Software de E/S Independiente del Dispositivo:** Esta capa se encuentra en el núcleo y proporciona funciones que son comunes a todos (o a grandes grupos) de dispositivos. Es la que permite que el SO sea modular.
- IV. **Software de E/S en el Espacio de Usuario:** Son las bibliotecas y programas que interactúan con el núcleo para realizar E/S.

CAPA	NIVEL DE EJECUCIÓN	FUNCIONES PRINCIPALES
4. Software de E/S a Nivel de Usuario (User-Level)	Espacio de Usuario	<ul style="list-style-type: none"> <li>• Formateo de E/S (printf, scanf).</li> <li>• Bibliotecas estándar (stdio.h).</li> <li>• Sistemas de Spooling (impresión).</li> </ul>
3. Software de E/S Independiente del Dispositivo (Device-Independent)	Espacio de Kernel	<ul style="list-style-type: none"> <li>• Interfaz uniforme para drivers.</li> <li>• Nombramiento y protección de dispositivos.</li> <li>• Buffering y Caching.</li> <li>• Gestión de bloques.</li> </ul>
2. Manejadores de Dispositivos (Device Drivers)	Espacio de Kernel	<ul style="list-style-type: none"> <li>• Comandos específicos de hardware.</li> <li>• Acceso a registros del controlador.</li> <li>• Gestión de errores de hardware.</li> <li>• Verificar estado del dispositivo.</li> </ul>
1. Manejadores de Interrupciones (Interrupt Handlers)	Espacio de Kernel	<ul style="list-style-type: none"> <li>• Ocultar interrupciones al resto del SO.</li> <li>• Salvar estado de la CPU.</li> <li>• Despertar al driver al finalizar la operación.</li> </ul>

### 5.3.1 Manejador de interrupciones

El manejador de interrupciones actúa como la primera línea de respuesta del sistema operativo ante eventos de hardware, con el objetivo principal de ocultar la complejidad de estas señales a las capas superiores. Cuando un dispositivo completa una operación, genera una señal eléctrica que fuerza a la CPU a detener su flujo actual, guardar el contexto de ejecución (registros y contador de programa) y saltar a una rutina de servicio ubicada en una dirección predefinida del vector de interrupciones. La labor fundamental de esta rutina es acusar recibo de la interrupción al controlador hardware, realizar las transferencias de datos críticas y, lo más importante, desbloquear al conductor del dispositivo (driver) que inició la operación y que se encontraba dormido esperando este evento.



Dado que las interrupciones bloquean otras actividades del procesador, los sistemas operativos modernos, como Linux o Windows, dividen este manejo en dos niveles para maximizar la eficiencia: una "mitad superior" o rutina de interrupción propiamente dicha, que es rápida y de alta prioridad para tareas urgentes, y una "mitad inferior" o procedimiento diferido, que se agenda para ejecutar tareas menos críticas posteriormente con interrupciones habilitadas. Este diseño permite que el sistema responda rápidamente a nuevos eventos sin perder datos, mientras que la parte compleja del procesamiento se realiza cuando la CPU está menos cargada, coordinando así la concurrencia entre el hardware y el software.

### **5.3.2 Controladores de los dispositivos**

Los controladores de dispositivos, o *drivers*, son módulos de software específicos escritos generalmente por el fabricante del hardware que actúan como traductores entre las peticiones abstractas del sistema operativo y el lenguaje de bajo nivel que entiende el hardware. Su función es aceptar comandos genéricos del software independiente del dispositivo (como "leer bloque N") y convertirlos en la secuencia exacta de escrituras en los registros de control del dispositivo físico necesario para ejecutar la acción, encapsulando así los detalles mecánicos y electrónicos del periférico. Esta arquitectura permite que el resto del sistema operativo permanezca agnóstico respecto al hardware específico, facilitando la conexión de nuevos dispositivos mediante la instalación de su driver correspondiente sin necesidad de recompilar el núcleo.

Internamente, un driver verifica los parámetros de entrada y comprueba si el dispositivo está libre; si está ocupado, suele encolar la petición para procesarla más tarde. Una vez que inicia la operación escribiendo en los registros del controlador, el driver determina si debe bloquearse esperando una interrupción (común en discos y redes) o si la operación es lo suficientemente rápida para terminar sin esperas; en sistemas modernos como Windows NT y Linux, los drivers se organizan a menudo en capas o clases, donde un manejador genérico gestiona la lógica común de una familia de dispositivos (como discos SCSI) y pasa las órdenes específicas a un "minipuerto" o driver específico que interactúa directamente con el hardware.



## UNIDAD 6

### Seguridad y Virtualización

#### 6.1 El ambiente de seguridad

El concepto de seguridad en sistemas computacionales es más amplio que el de protección; mientras que la protección se refiere a los mecanismos internos del sistema operativo para controlar el acceso a los recursos, la seguridad es una medida de confianza en que la integridad del sistema y sus datos se preservarán frente a amenazas tanto internas como externas. Un ambiente seguro debe garantizar tres objetivos fundamentales: la confidencialidad (que la información solo sea accesible por quienes están autorizados), la integridad (que la información no sea alterada indebidamente) y la disponibilidad (que los recursos estén accesibles cuando se necesiten).

Las amenazas a este ambiente pueden ser maliciosas, como intrusos (*hackers*) que intentan acceder sin autorización, virus, gusanos y ataques de denegación de servicio, o accidentales, como fallos de hardware o errores humanos que provocan pérdida de datos. Para defenderse, la seguridad debe implementarse en múltiples niveles o capas: físico (controlando el acceso a la sala de máquinas), de red (protegiendo la transmisión de datos), de sistema operativo y de aplicación, ya que la seguridad del sistema completo es tan fuerte como su eslabón más débil.

##### 6.1.1 Seguridad en los sistemas operativos

La seguridad en el nivel del sistema operativo se basa en principios de diseño robustos, como los identificados por Saltzer y Schroeder, que incluyen el diseño abierto (no basar la seguridad en el secreto del diseño), el privilegio mínimo (cada proceso o usuario debe operar con los permisos mínimos necesarios para su tarea) y la economía de mecanismos (mantener los sistemas de protección lo más simples y pequeños posible para facilitar su verificación). Además, el sistema debe aplicar la intermediación completa, verificando cada acceso a cada objeto, y fomentar la compartición mínima para evitar canales encubiertos de fuga de información.

En la práctica, los sistemas operativos deben defenderse activamente contra amenazas programáticas como el código malicioso y vulnerabilidades técnicas como los desbordamientos de búfer (*buffer overflows*), que permiten a un atacante inyectar código y secuestrar el flujo de ejecución de un proceso privilegiado. Para ello, implementan controles como la autenticación rigurosa de usuarios (verificando identidades mediante contraseñas u otros medios), la auditoría y registro de eventos para detectar comportamientos anómalos, y técnicas de mitigación como bits de no-ejecución (NX) en memoria y la aleatorización del espacio de direcciones (ASLR).



### 6.1.2 Control de acceso a los recursos

El control de acceso es la función que asegura que los recursos del sistema (CPU, memoria, archivos, dispositivos) sean utilizados únicamente por aquellos procesos que han obtenido la autorización adecuada según la política de seguridad. Este control se modela frecuentemente mediante el concepto de dominio de protección, que define un conjunto de pares objeto-derechos; un proceso que se ejecuta en un dominio específico solo puede realizar las operaciones permitidas sobre los objetos disponibles en ese dominio.

La implementación de estos dominios a menudo se apoya en identificadores de usuario (UID) y de grupo (GID), los cuales determinan los privilegios de un proceso en un momento dado, y en estructuras jerárquicas como los anillos de protección. En el modelo de anillos (usado por ejemplo en la arquitectura Intel y sistemas como Multics), el sistema operativo ocupa el anillo interior más privilegiado (modo núcleo), y las aplicaciones se ejecutan en anillos exteriores con acceso restringido (modo usuario), requiriendo mecanismos de hardware (como trampas o excepciones) para transitar de manera controlada y segura hacia los servicios privilegiados.

### 6.1.3 Implementación de matrices de acceso

La matriz de acceso es un modelo abstracto que representa la política de protección completa: las filas representan los dominios (o sujetos) y las columnas los objetos, donde cada celda indica los derechos de acceso específicos (leer, escribir, ejecutar). Dado que esta matriz suele ser enorme y dispersa (con la mayoría de las celdas vacías), no se almacena directamente, sino que se implementa descomponiéndola por columnas o por filas.

Si se descompone por columnas (objetos), se obtienen Listas de Control de Acceso (ACL), que asocian a cada objeto una lista de usuarios o dominios permitidos; este es el método estándar en sistemas como Windows NT y UNIX (donde se usan bits de protección simplificados). Si se descompone por filas (dominios), se obtienen listas de capacidades (*capabilities*), donde cada proceso posee un conjunto de "llaves" o tickets inmodificables que le otorgan acceso a objetos específicos;

mientras que las ACL facilitan la revocación de permisos sobre un objeto, las capacidades facilitan la delegación eficiente de derechos entre procesos sin intervención centralizada.

#### 6.1.4 Modelos formales de seguridad

Los modelos formales permiten especificar y verificar matemáticamente las políticas de seguridad de un sistema, clasificándose generalmente en modelos multinivel y modelos de acceso limitado o discrecional. El modelo más conocido es el de Bell-LaPadula, diseñado para entornos militares, que se centra en la confidencialidad mediante reglas estrictas como "no leer hacia arriba" (un sujeto no puede leer datos de mayor clasificación de seguridad) y "no escribir hacia abajo" (no se puede filtrar información clasificada a niveles inferiores).

Existen otros modelos como el de Biba, que se enfoca en la integridad de los datos (invirtiendo las reglas de Bell-LaPadula para evitar que datos de baja integridad corrompan datos críticos), y el modelo de Harrison-Ruzzo-Ullman (HRU), que analiza la seguridad en sistemas donde los derechos de acceso cambian dinámicamente. Estos modelos fundamentan los criterios de evaluación de seguridad, como los definidos en el "Libro Naranja" (*Orange Book*) del Departamento de Defensa de EE. UU., que clasifica los sistemas en niveles desde D (sin seguridad) hasta A (diseño verificado formalmente), pasando por el nivel C (protección discrecional común en sistemas comerciales) y B (protección obligatoria o MAC).



#### 6.2 Virtualización

La virtualización es una tecnología que permite crear múltiples entornos simulados o recursos dedicados a partir de un solo sistema físico de hardware. Es la capacidad de ejecutar varios sistemas operativos (invitados) de forma simultánea e independiente sobre un mismo hardware físico (anfitrión).

### 6.2.1 Emulación

La emulación es una técnica de virtualización que consiste en implementar mediante software un sistema completo que simula el comportamiento de una arquitectura de hardware específica, permitiendo que un sistema operativo o aplicación diseñado para una CPU (como PowerPC) se ejecute en otra totalmente distinta (como Intel x86). En este esquema, el emulador actúa como un traductor que convierte cada instrucción de la arquitectura fuente a la arquitectura anfitriona, además de tener que recrear por software el comportamiento de todos los chips de apoyo y periféricos necesarios para que el sistema huésped funcione.

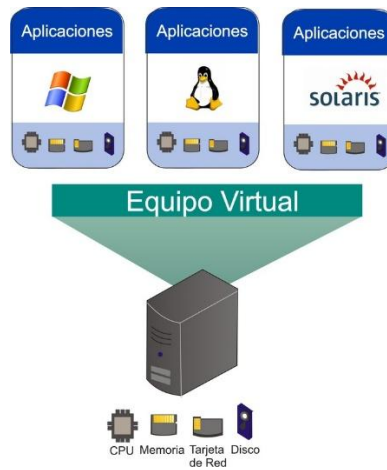
Debido a la sobrecarga que implica traducir instrucciones y simular componentes físicos, la emulación es inherentemente ineficiente y lenta, rindiendo a veces órdenes de magnitud menos que la ejecución nativa, a menos que el anfitrión sea extremadamente potente en comparación con el sistema emulado. Sin embargo, su utilidad es indiscutible para preservar software legado, ejecutar videojuegos de consolas antiguas o desarrollar software para sistemas embebidos y móviles en computadoras de escritorio antes de desplegarlos en el hardware real



### 6.2.2 Virtualización asistida por hardware

Esta tecnología surge para superar las limitaciones de rendimiento y complejidad del software de virtualización tradicional mediante la inclusión de extensiones específicas en los procesadores, como Intel VT-x y AMD-V (Pacífica), introducidas alrededor de 2005. Estas extensiones definen nuevos modos de operación en la CPU (modo anfitrión/raíz y modo huésped/no raíz), permitiendo que el hipervisor o monitor de máquina virtual (VMM) ejecute sistemas operativos invitados directamente en el hardware sin necesidad de técnicas complejas como la traducción binaria o la paravirtualización.

Además del procesador, la asistencia por hardware se extiende a la gestión de memoria y entrada/salida mediante tecnologías como las tablas de páginas anidadas (EPT o RVI), que permiten a la CPU realizar la traducción de direcciones físicas del huésped a físicas del anfitrión sin intervención constante del VMM, reduciendo significativamente los fallos de página y mejorando el rendimiento. También se incluyen mejoras como el remapeo de interrupciones y el paso directo de dispositivos (DMA pass-through), lo que permite que los sistemas virtualizados alcancen velocidades de ejecución muy cercanas a las nativas y aumenten la estabilidad al utilizar controladores estándar.



### 6.2.3 Paravirtualización

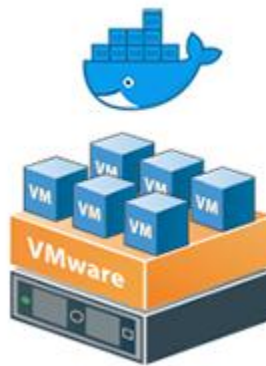
La paravirtualización es un enfoque que busca eficiencia modificando el sistema operativo huésped para que sea "consciente" de que se está ejecutando en un entorno virtualizado, en lugar de intentar engañarlo con una copia exacta del hardware. En lugar de ejecutar instrucciones privilegiadas que fallarían y tendrían que ser atrapadas por el hipervisor, el núcleo del sistema operativo invitado colabora con el anfitrión enviando llamadas explícitas, conocidas como "hiperllamadas" (hypercalls), para solicitar operaciones sensibles como la gestión de tablas de memoria o el acceso a dispositivos.

Este modelo, popularizado por el proyecto Xen, permite una estructura de virtualización más delgada y eficiente, eliminando la sobrecarga de la emulación de hardware y optimizando operaciones críticas de entrada y salida. Aunque requiere acceso al código fuente del sistema operativo para modificarlo (lo cual facilitó su adopción en Linux y sistemas libres), sus principios se aplican hoy en día mediante controladores paravirtualizados (como *virtio*) que ofrecen un rendimiento muy superior al de la emulación de dispositivos físicos tradicionales.

### 6.2.4 Contenedores

Los contenedores representan una forma de virtualización a nivel del sistema operativo, donde en lugar de duplicar el hardware para ejecutar múltiples núcleos (kernels), se utiliza un solo núcleo anfitrión para ejecutar múltiples instancias de espacio de usuario aisladas entre sí. El sistema operativo anfitrión emplea mecanismos como los espacios de nombres (*namespaces*) para segregar la visión que cada contenedor tiene de los recursos, proporcionando tablas de procesos, interfaces de red y sistemas de archivos independientes, de modo que cada contenedor cree tener el control total del sistema.

Esta tecnología es mucho más ligera que las máquinas virtuales tradicionales, ya que evita la sobrecarga de iniciar y mantener múltiples sistemas operativos completos; un contenedor inactivo no consume recursos de CPU, comportándose simplemente como un grupo de procesos dormidos. Herramientas de orquestación como Docker y Kubernetes han popularizado este enfoque para el despliegue rápido de aplicaciones y microservicios, permitiendo gestionar entornos complejos de manera eficiente y escalable en la nube.





## BIBLIOGRAFÍA

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts. Capítulo 1.2.1: "Interrupts".

**Fuente: "Computer System Engineering | Electrical Engineering and Computer Science | MIT OpenCourseWare"**

- LaCurts, K. (2018). *Computer System Engineering* [Material de curso]. MIT OpenCourseWare. Departamento de Ingeniería Eléctrica y Ciencias de la Computación.

**Fuente: "Operating System Engineering | Electrical Engineering and Computer Science | MIT OpenCourseWare"**

- Kaashoek, F. (2012). *Operating System Engineering* [Material de curso]. MIT OpenCourseWare. Departamento de Ingeniería Eléctrica y Ciencias de la Computación.

**Fuente: "Unix\_programacion\_avanzada\_3a\_ed\_Marquez.pdf"**

- Márquez García, F. M. (2004). *UNIX programación avanzada* (3.<sup>a</sup> ed.). RA-MA Editorial.

**Fuente: "Bash Pocket Reference"**

- Robbins, A. (2010). *Bash pocket reference*. O'Reilly Media.

**Fuente: "Operating System Concepts"**

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10.<sup>a</sup> ed.). Wiley.

**Fuente: "Operating Systems Design and Implementation"**

- Tanenbaum, A. S., & Woodhull, A. S. (2006). *Operating systems: Design and implementation* (3.<sup>a</sup> ed.). Pearson Prentice Hall.

**Fuente: "Fundamentos de sistemas operativos"**

- Wolf, G., Ruiz, E., Bergero, F., & Meza, E. (2015). *Fundamentos de sistemas operativos*. Universidad Nacional Autónoma de México.

**Fuente: "Sistemas operativos: Una visión aplicada"**

- Carretero Pérez, J., García Carballeira, F., de Miguel Anasagasti, P., & Pérez Costoya, F. (2001). *Sistemas operativos: Una visión aplicada*. McGraw-Hill Interamericana.