

Sistemas Informáticos

Unidad 03.

Fundamentos de sistemas operativos



Autores: Sergi García, Alfredo Oltra

Actualizado Septiembre 2025



Licencia



Reconocimiento - No comercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE

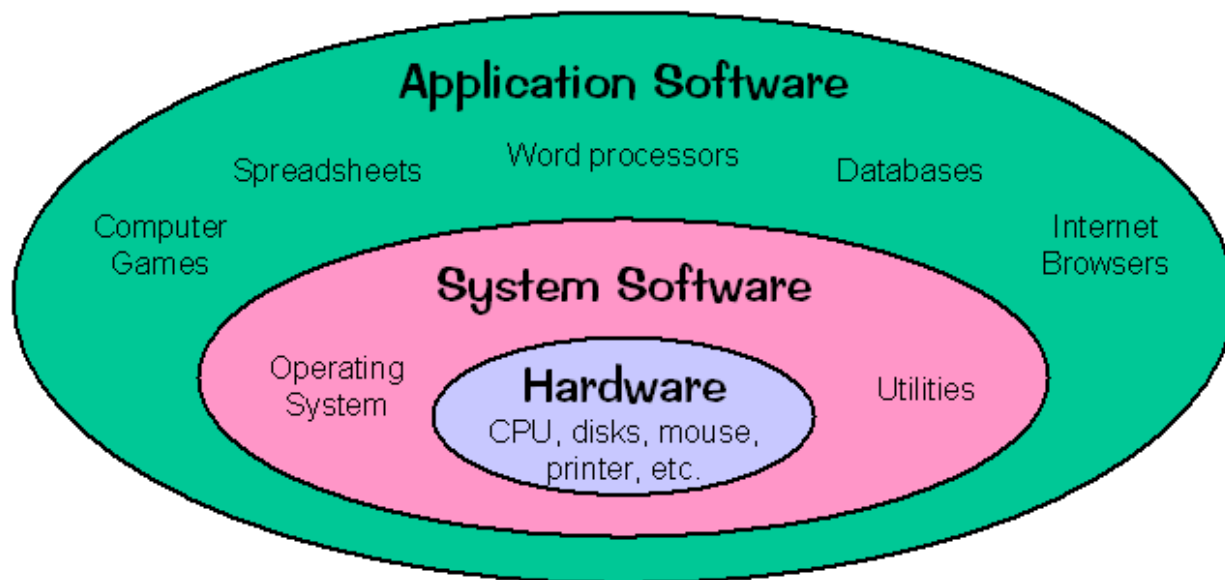
| | |
|--|-----------|
| 1. Introducción | 4 |
| 2. Historia | 4 |
| 3. Clasificación de los sistemas operativos | 5 |
| 4. Sistemas operativos más populares | 7 |
| 5. Funciones del sistema operativo | 8 |
| 6. Gestión de procesos | 8 |
| 6.1. ¿Qué es un proceso? | 8 |
| 6.2. Procesos e hilos (threads) | 9 |
| 6.3. Estados de los procesos | 9 |
| 6.4. Planificador a corto plazo | 11 |
| 6.5. Algoritmos de planificación de CPU | 11 |
| 6.6. ¿Cómo resolver ejercicios de "algoritmos de planificación"? | 14 |
| 7. Gestión de memoria | 14 |
| 7.1. Principales problemas de la gestión de memoria | 15 |
| 7.2. Problema de poca memoria: Swapping | 16 |
| 8. Gestión de entrada / Salida | 17 |
| 8.1. Formas de gestionar la entrada y salida | 17 |
| 8.2. Técnicas para incrementar el rendimiento de la E/S | 18 |
| 9. Gestión del sistema de archivos | 19 |
| 9.1. Estructura de un sistema de archivos | 19 |
| 9.2. Atributos de un sistema de archivos | 19 |
| 9.3. Organización interna | 19 |

| | |
|--|-----------|
| 9.4. Rutas absolutas y relativas | 20 |
| 9.5. Características de los sistemas de ficheros más populares | 22 |
| 10. Material adicional | 23 |
| 11. Bibliografía | 23 |

UNIDAD 03. FUNDAMENTOS DE SISTEMAS OPERATIVOS

1. INTRODUCCIÓN

Un sistema operativo es el software básico que gestiona un ordenador. Se encarga de controlar todos los recursos de hardware y de ocultar su complejidad al usuario final, quien normalmente interactúa a través de una interfaz gráfica. Además, es responsable de ejecutar las aplicaciones. Podemos decir que el sistema operativo actúa como un intermediario entre el hardware y las aplicaciones.



2. HISTORIA

La evolución de los sistemas operativos (SO) puede resumirse en los siguientes hitos:

- **Primeras computadoras (décadas de 1940-50):** no existían sistemas operativos. Los programadores interactuaban directamente con la máquina, modificando el hardware o introduciendo instrucciones de forma manual.
- **Años 50:** surge el concepto de sistema operativo, inicialmente con funciones muy básicas, como cargar un programa, ejecutarlo y, al finalizar, preparar el siguiente.
- **Años 60:** se desarrollan los principales conceptos de los sistemas operativos modernos: multiusuario, multitarea, multiprocesador y sistemas en tiempo real. A finales de esta década aparece **Unix**, que marcará un antes y un después.
- **Años 70:** los ordenadores comienzan a popularizarse más allá de gobiernos y grandes corporaciones. En este contexto nace el lenguaje **C**, y Unix se reescribe en él, lo que facilita su portabilidad y difusión.
- **Años 80:** se da un gran impulso a la usabilidad. Aparecen las primeras interfaces gráficas, que buscan hacer los sistemas más accesibles al usuario común.

- **Años 90:** se consolidan sistemas como **Windows** y **Linux**, que dominan gran parte del mercado hasta la actualidad.
- **Años 2000 en adelante:** surgen sistemas operativos adaptados a dispositivos móviles, como **Android** e **iOS**, que hoy son los más utilizados en smartphones y tabletas.

3. CLASIFICACIÓN DE LOS SISTEMAS OPERATIVOS

Los sistemas operativos se pueden clasificar en función de distintos parámetros. Es importante entender que un mismo sistema puede pertenecer a varias categorías a la vez, ya que las clasificaciones son aproximadas y responden a características técnicas y de uso.

1. Según el número de usuarios simultáneos

- **Monousuario:** solo una persona puede interactuar con el sistema en un momento dado. Son típicos de los primeros ordenadores personales.
 - *Ejemplo:* **MS-DOS**.
- **Multiusuario:** varios usuarios pueden trabajar de forma concurrente sobre el mismo sistema, ya sea desde diferentes terminales físicos o mediante sesiones remotas.
 - *Ejemplo:* **Unix, Linux, Windows Server**.

👉 La capacidad multiusuario no significa necesariamente que cada usuario tenga un ordenador distinto, sino que comparten un mismo sistema con mecanismos de control de acceso, seguridad y gestión de recursos.


2. Según el número de procesadores soportados

- **Monoprocesador:** el sistema operativo está diseñado para funcionar con un solo procesador. Suelen ser más simples y con menos capacidades de paralelismo.
 - *Ejemplo:* sistemas antiguos como **MS-DOS** o las primeras versiones de **Windows 3.x**.
- **Multiprocesador:** el sistema puede aprovechar más de un procesador físico o núcleo. Dentro de esta categoría distinguimos:
 - **SMP (Multiprocesamiento Simétrico):** todos los procesadores comparten las mismas funciones y los procesos se distribuyen equitativamente.
 - *Ejemplo:* sistemas modernos como **Linux** o **Windows 10/11**.
 - **AMP (Multiprocesamiento Asimétrico):** algunos procesadores se dedican a tareas específicas, como gestión del sistema, mientras que otros ejecutan procesos de usuario. Se utilizaba en sistemas más antiguos y en algunos sistemas embebidos.

👉 Hoy en día, con la popularización de procesadores multinúcleo, la mayoría de sistemas operativos de uso general implementan **SMP**.

3. Según el número de tareas simultáneas

- **Monotarea:** solo permite ejecutar un programa a la vez. Si el usuario quiere abrir otro, debe cerrar el primero.
 - *Ejemplo: MS-DOS.*
- **Multitarea:** el sistema operativo gestiona varias aplicaciones a la vez.
 - **Multitarea cooperativa:** cada programa decide cuándo cede el control al sistema. Era menos fiable, pues un programa defectuoso podía bloquear todo el sistema.
 - *Ejemplo: Windows 3.x, Windows 95.*
 - **Multitarea preventiva:** el sistema operativo decide cuándo interrumpir un proceso para dar paso a otro, garantizando una mejor estabilidad y seguridad.
 - *Ejemplo: todos los sistemas modernos (Linux, Windows NT/2000/XP en adelante, macOS).*

 **Interesante:** Un sistema monoprocesador también puede ser multitarea gracias al cambio de contexto. Aunque solo ejecuta una instrucción a la vez, el procesador cambia tan rápido de tarea que el usuario percibe que se ejecutan varias a la vez (ejemplo: escuchar música mientras se navega por internet).

Clasificación de los sistemas operativos

| Sistema operativo | Número de usuarios | Número de tareas simultáneas | Número de procesadores soportados |
|---|--------------------|------------------------------|-----------------------------------|
| MS-DOS | Mono | Mono | Mono |
| Windows 9x, Me | Mono | Multi | Mono |
| Windows XP/Vista/7/8/10/11 | Mono/Multi | Multi | Multi |
| UNIX, Linux, Windows NT, Windows Server | Multi | Multi | Multi |

4. SISTEMAS OPERATIVOS MÁS POPULARES



En la actualidad existen numerosos sistemas operativos, pero solo algunos han alcanzado una gran difusión. Podemos clasificarlos en dos grandes grupos:

- **Software libre:** el código fuente está disponible, lo que permite estudiarlo, modificarlo y redistribuirlo. Generalmente se pueden usar sin coste económico.
- **Software privativo:** el código fuente no está disponible y su uso suele requerir el pago de licencias.

Principales sistemas operativos

- **Linux:** muy utilizado en servidores, supercomputadoras y también en equipos de escritorio. Es software libre y existen muchas distribuciones (Ubuntu, Fedora, Debian, Arch, entre otras). Su seguridad, estabilidad y flexibilidad lo han convertido en la base de gran parte de la infraestructura de internet.
- **Microsoft Windows:** el sistema operativo más extendido en equipos de escritorio y portátiles. También tiene versiones para servidores (Windows Server). Es software privativo, aunque con gran compatibilidad de hardware y aplicaciones.
- **macOS (antes Mac OS X):** diseñado exclusivamente para computadoras Apple. Está basado en BSD (familia de Unix). Es software privativo y se caracteriza por su gran integración con el ecosistema de Apple.
- **Android:** sistema operativo libre basado en Linux, orientado a dispositivos móviles (teléfonos y tabletas). Aunque el núcleo es libre, la mayoría de fabricantes incluyen capas de software adicionales, a veces privativas. Es el sistema móvil más utilizado en el mundo.
- **iOS (Apple):** sistema operativo privativo para dispositivos móviles de Apple (iPhone, iPad). Destaca por su estabilidad, seguridad y optimización para el hardware de la compañía.

👉 Otros sistemas relevantes, aunque menos populares en uso general, son **FreeBSD**, **ChromeOS** (Google), o **sistemas embebidos en IoT**.

5. FUNCIONES DEL SISTEMA OPERATIVO

El sistema operativo cumple funciones esenciales para que un ordenador pueda trabajar de forma ordenada y eficiente. Entre ellas se encuentran:

- **Gestión de procesos:** decide qué procesos pueden usar la CPU y en qué momento.
- **Gestión de memoria:** organiza y controla el uso de la memoria principal, asignando espacio a los procesos y evitando que interfieran entre sí.
- **Gestión de dispositivos de entrada/salida (E/S):** coordina la comunicación entre el hardware (teclado, ratón, disco, impresora, etc.) y las aplicaciones.
- **Gestión del sistema de archivos:** organiza la información en discos y memorias mediante estructuras lógicas (archivos, directorios) que facilitan el acceso a los datos.

👉 Estas funciones se estudiarán en detalle en los siguientes apartados.

6. GESTIÓN DE PROCESOS

La gestión de procesos es especialmente importante en los sistemas multitarea, donde varios programas se ejecutan al mismo tiempo. En los sistemas monotarea no existe este problema, porque un único proceso monopoliza la CPU.

6.1. ¿Qué es un proceso?

Un **proceso** es un programa en ejecución. Esto significa que:

- Debe cargarse en la memoria principal.
- Utiliza tiempo de CPU para ejecutar sus instrucciones.
- Puede requerir otros recursos, como acceso a disco o dispositivos de entrada/salida.


📌 Una analogía clásica:

- **Programa:** la receta escrita en un libro.
- **Proceso:** el acto de cocinar siguiendo esa receta.

Diferencia entre programa y proceso

- Un **programa** es un conjunto estático de instrucciones, datos y recursos almacenados en un archivo ejecutable.
- Un **proceso** es la instancia activa de un programa en ejecución.

👉 Cuando ejecutamos un programa normalmente se crea **un proceso**, pero algunos programas (como navegadores web o bases de datos) pueden generar **múltiples procesos** para repartir el trabajo y mejorar el rendimiento.

 **Atención:** un proceso no es lo mismo que un programa. Confundirlos es un error común en los primeros cursos de informática.

6.2. Procesos e hilos (threads)

Hasta ahora hemos visto que un proceso es un programa en ejecución. Sin embargo, dentro de un proceso puede haber **uno o varios hilos de ejecución (threads)**.

- **Hilo:** es la unidad básica de ejecución dentro de un proceso.
- Cada hilo comparte con los demás el mismo espacio de memoria y recursos del proceso, pero tiene su propio contador de programa y pila de ejecución.


Ventajas de los hilos

- Permiten dividir el trabajo de un proceso en subtareas que se ejecutan en paralelo o de forma concurrente.
- Mejoran el rendimiento en sistemas multiprocesador, ya que diferentes hilos pueden ejecutarse en distintos núcleos al mismo tiempo.
- Reducen el coste de creación y cambio de contexto respecto a los procesos, ya que comparten gran parte de sus recursos.

Ejemplo práctico:

En un navegador web moderno:

- Un proceso puede representar una pestaña.
- Dentro de esa pestaña, un hilo puede encargarse de cargar el texto, otro de mostrar imágenes y otro de reproducir un vídeo, todo al mismo tiempo.

 **Atención:** trabajar con hilos también introduce complejidad, ya que al compartir memoria pueden aparecer problemas como condiciones de carrera o bloqueos mutuos (deadlocks) si no se gestionan correctamente.

6.3. Estados de los procesos

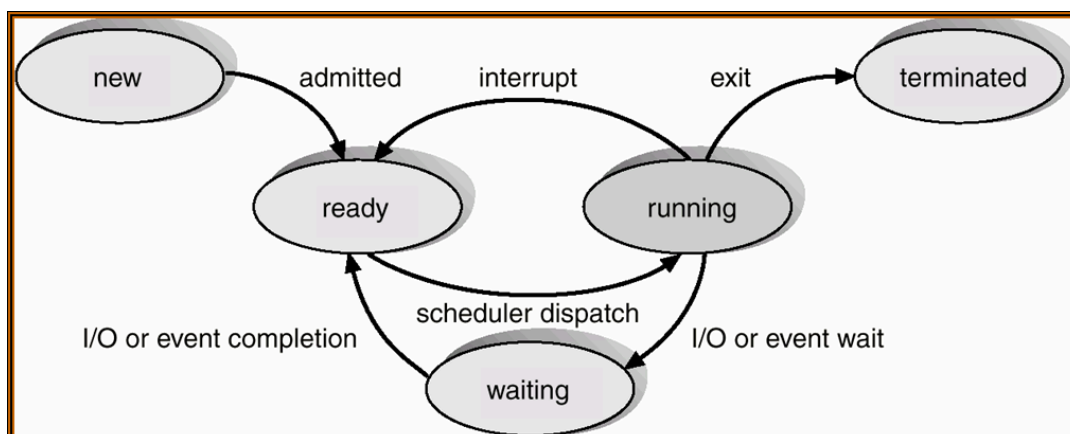
Un proceso no permanece siempre en la misma situación. A lo largo de su ciclo de vida pasa por diferentes **estados**, que representan su condición actual, y se producen **transiciones** entre dichos estados.

Principales estados de un proceso

- **Nuevo (New):** el programa acaba de iniciarse y el sistema crea un proceso. En este punto, un componente llamado **planificador a largo plazo (long-term scheduler)** decide si el proceso es admitido o no, en función de criterios como:
 - Cantidad de memoria disponible.
 - Número de procesos ya en ejecución.

- Políticas de carga del sistema.
- **Nuevo → Listo (New → Ready):** si el planificador a largo plazo acepta el proceso, este pasa al estado **Listo**.
- **Listo (Ready):** el proceso tiene todos los recursos necesarios (excepto la CPU) y espera ser seleccionado por el **planificador a corto plazo (short-term scheduler)**.
- **Listo → En ejecución (Ready → Running):** cuando el planificador a corto plazo escoge al proceso, este accede a la CPU y pasa a estado **En ejecución**.
- **En ejecución (Running):** el proceso está utilizando efectivamente la CPU para realizar sus operaciones.
- **En ejecución → Espera (Running → Waiting):** el proceso necesita realizar una operación de **entrada/salida (E/S)** (por ejemplo, leer datos de disco). Como no puede continuar hasta que termine, libera la CPU y pasa a estado **Esperando**.
- **En ejecución → Listo (Running → Ready):** el proceso aún puede ejecutarse, pero el planificador a corto plazo decide interrumpirlo para dar paso a otro proceso.
- **En ejecución → Terminado (Running → Terminated):** el proceso ha finalizado todas sus operaciones y libera la CPU definitivamente.
- **Terminado (Terminated):** el proceso ha concluido su ejecución y sus recursos son liberados por el sistema operativo.
- **Esperando (Waiting):** el proceso no puede continuar porque espera el resultado de una operación de E/S u otro recurso externo.
- **Esperando → Listo (Waiting → Ready):** cuando la operación de E/S ha concluido, el proceso está listo de nuevo para competir por la CPU.

👉 Este ciclo de vida se suele representar con un **diagrama de estados**, donde se visualizan los pasos y transiciones.



6.4. Planificador a corto plazo


El **planificador a corto plazo** es el componente encargado de decidir **qué proceso obtiene la CPU** en un momento dado.

- En un sistema con **un solo procesador**, únicamente un proceso puede estar en ejecución en un instante.
- En sistemas multiprocesador, el planificador debe distribuir los procesos entre diferentes núcleos.

Algoritmos de planificación

Existen diversos algoritmos que permiten decidir de manera eficiente y justa qué proceso usar en cada momento. Algunos de los más comunes son:

- **FIFO (First In, First Out) o FCFS (First Come, First Served)**: el primer proceso en llegar a la cola de listos es el primero en ejecutarse. Sencillo, pero puede provocar tiempos de espera largos para procesos cortos.
- **Round Robin (RR)**: cada proceso recibe la CPU por un intervalo de tiempo fijo (quantum). Si no termina, se interrumpe y pasa al final de la cola. Es muy utilizado en sistemas interactivos.
- **SJF (Shortest Job First)**: se da prioridad al proceso que requiere menos tiempo de CPU. Reduce el tiempo promedio de espera, pero puede provocar inanición (starvation) de procesos largos.
- **Prioridades**: cada proceso recibe una prioridad, y los de mayor prioridad se ejecutan antes. Puede combinarse con otros algoritmos.
- **Planificación por prioridades dinámicas**: el sistema ajusta las prioridades de los procesos para evitar que algunos queden indefinidamente sin ejecutar.

 **Importante:** la elección del algoritmo afecta directamente al rendimiento percibido por el usuario, la justicia entre procesos y el tiempo de respuesta de las aplicaciones.

6.5. Algoritmos de planificación de CPU

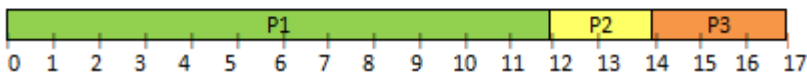
Cuando varios procesos compiten por la CPU, el sistema operativo debe decidir **qué proceso se ejecuta primero** y durante cuánto tiempo. Para ello utiliza diferentes **algoritmos de planificación**, cada uno con ventajas e inconvenientes.

1. FIFO (First In, First Out) o FCFS (First Come, First Served)

- **Idea básica:** funciona como una cola de supermercado: el primero que llega es el primero en ser atendido.
- **Características:**
 - Es simple de implementar.

- Justo en el sentido de que respeta el orden de llegada.
- Puede provocar tiempos de espera largos para procesos cortos si delante hay procesos muy largos (efecto conocido como **convoy effect**).
- **Ejemplo de uso:**
 - Procesamiento de trabajos por lotes en sistemas antiguos.
 - Impresoras compartidas: el primer documento enviado a imprimir es el primero en salir.

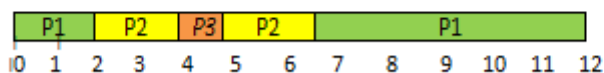
| Process | Arriving time | CPU |
|---------|---------------|-----|
| P1 | 0 | 12 |
| P2 | 2 | 2 |
| P3 | 5 | 3 |



2. Shortest Remaining Time First (SRTF)

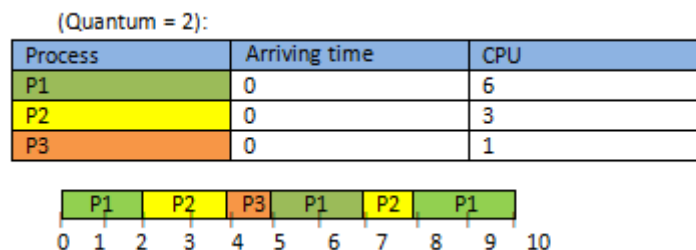
- **Idea básica:** el proceso con menor tiempo de CPU pendiente es el que obtiene la CPU.
- **Características:**
 - Es un algoritmo **apropiativo**: un proceso que está ejecutándose puede ser interrumpido si llega otro con menor tiempo estimado de ejecución.
 - Intenta minimizar el **tiempo medio de espera**.
- **Problemas:**
 - No podemos conocer con exactitud cuánto durará un proceso.
 - Puede producir **inanición (starvation)**: si siempre llegan procesos cortos, los procesos largos pueden quedar indefinidamente relegados.
- **Ejemplo de uso:**
 - Es más común en entornos teóricos o de simulación que en sistemas reales, aunque variantes aproximadas se utilizan en sistemas con planificación predictiva.

| Process | Arriving time | CPU |
|---------|---------------|-----|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |



3. Round Robin (RR)

- **Idea básica:** cada proceso recibe la CPU por un intervalo de tiempo fijo, llamado **quantum**.
- **Características:**
 - Justo y equitativo: todos los procesos obtienen tiempo de CPU de manera alternada.
 - Es el algoritmo más usado en sistemas **multitarea e interactivos**.
 - Si un proceso no termina durante su quantum, se interrumpe y pasa al final de la cola de listos.
 - Si no hay otros procesos esperando, el mismo proceso puede continuar.
- **Analogía para entenderlo:**
Imagina que quieres alquilar un campo de fútbol.
 - El quantum equivale a 2 horas.
 - Si hay más equipos esperando, al acabar tu turno debes dejar el campo y esperar tu turno de nuevo.
 - Si ningún otro equipo está esperando, puedes volver a jugar otras 2 horas seguidas.
 - Si un equipo se marcha antes de agotar su tiempo, el campo queda disponible inmediatamente.
- **Ejemplo de uso:**
 - Sistemas operativos modernos como **Linux**, **Windows** o **macOS** implementan variantes del algoritmo Round Robin en la planificación de procesos interactivos.



⚠ Atención: en la práctica, los procesos rara vez llegan exactamente al mismo tiempo. Sin embargo, en los **ejercicios teóricos** es común suponer que sí. En ese caso, el orden de llegada lo decide el enunciado o, si no se especifica, puede resolverse de forma arbitraria siempre que se sea consistente.

6.6. ¿Cómo resolver ejercicios de “algoritmos de planificación”?

En este módulo es habitual practicar con ejercicios que simulan la planificación de procesos. Para resolverlos correctamente se recomienda:

1. **Comprender bien el algoritmo:** leer sus reglas y restricciones antes de empezar.
2. **Construir una tabla o línea de tiempo (diagrama de Gantt):** anotar el instante en que cada proceso entra y sale de la CPU.
3. **Aplicar el algoritmo paso a paso:** siguiendo exactamente lo que haría el sistema operativo.
4. **Calcular métricas de rendimiento:**
 - Tiempo de espera (waiting time).
 - Tiempo de retorno o finalización (turnaround time).
 - Tiempo de respuesta (response time), si procede.
5. **Prestar especial atención al algoritmo Round Robin,** ya que suele ser el más complejo por los cambios frecuentes de proceso.

👉 En los exámenes, la dificultad no suele estar en las operaciones matemáticas, sino en **seguir rigurosamente la lógica del algoritmo**.

7. GESTIÓN DE MEMORIA

La **gestión de memoria** es un problema fundamental en los sistemas **multitarea**. En un sistema **monotarea**, este problema no existe, ya que el único proceso en ejecución puede utilizar toda la memoria disponible sin necesidad de compartirla.

Sin embargo, en un sistema multitarea:

- La **memoria es un recurso escaso y muy demandado**.
- Existen muchos procesos activos que requieren grandes cantidades de memoria.
- El sistema operativo debe decidir **cómo organizar, asignar y proteger** la memoria para garantizar estabilidad y eficiencia.

A lo largo de la historia de los sistemas operativos, las técnicas de gestión de memoria han ido evolucionando conforme aumentaba la cantidad de memoria disponible y el número de procesos concurrentes.

Paginación y páginas de memoria

La memoria RAM suele dividirse en unidades llamadas **páginas**.

- El tamaño de cada página depende de la implementación del sistema operativo (por ejemplo, 4 KB en muchos sistemas modernos).
- Los procesos no trabajan directamente con direcciones físicas, sino con un espacio de direcciones **virtual**.

7.1. Principales problemas de la gestión de memoria

1. Problema de protección

- Un proceso no debe poder acceder a la memoria de otro proceso.
- El sistema operativo establece **límites de memoria** para cada proceso y, si detecta un acceso indebido, genera un error (generalmente, una **violación de segmento o segmentation fault**).

2. Problema de reubicación (relocation)

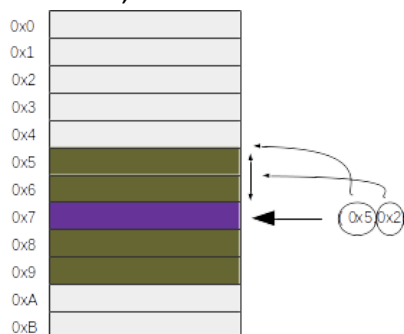
- Cuando un programa se compila, no sabe en qué posiciones de memoria se ejecutará. Dependerá de la memoria disponible y de qué procesos estén activos.
- Solución: cada proceso cree que tiene todo un espacio de memoria libre, empezando desde la dirección 0. Esto se logra gracias a la **memoria virtual**, donde el sistema operativo traduce direcciones **virtuales** a direcciones **físicas reales** mediante una tabla de páginas.

3. Problema de asignación y fragmentación

- Cuando un proceso termina, libera su espacio en memoria. Esto genera **huecos dispersos** que dificultan la asignación de memoria continua a nuevos procesos.
- Solución: el sistema operativo emplea la **traducción de direcciones virtuales**. Así, en lugar de exigir que la memoria sea continua, permite que un proceso ocupe páginas

físicas dispersas.

- Ejemplo: una dirección virtual (452, 45) puede traducirse a una dirección física real como 497, combinando un número de página con un desplazamiento (*offset*).

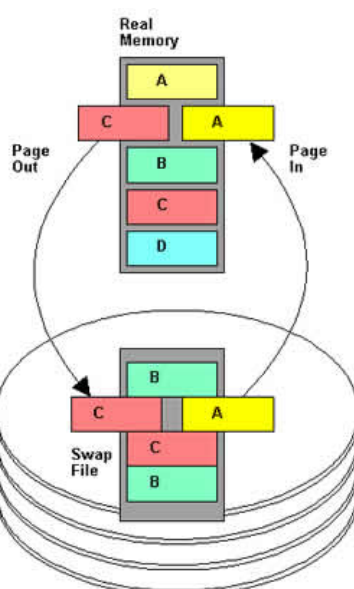


✿ Esto permite aprovechar mejor la memoria y evita problemas de fragmentación externa.

7.2. Problema de poca memoria: Swapping

Cuando los procesos requieren **más memoria de la que físicamente existe**, el sistema operativo recurre a una técnica llamada **swapping** o **intercambio**.

- El sistema utiliza un dispositivo de almacenamiento secundario (generalmente el disco duro o SSD) para guardar páginas de memoria menos usadas.
- Si un proceso necesita una página que está en disco, el sistema operativo intercambia una página de la RAM (normalmente la menos usada) por la página requerida.
- Es importante destacar:
 - Un proceso **solo puede ejecutarse si está en RAM**.
 - El disco duro **no se usa como RAM directa**, sino como un respaldo temporal.



👉 Este mecanismo permite que el sistema ejecute más procesos de los que cabrían en memoria física, pero con un coste: el acceso a disco es miles de veces más lento que a RAM.

Ejemplo práctico para entender swapping

Imagina que tienes una **mesa de trabajo pequeña (RAM)** y una **estantería al lado (disco duro)**:

- Solo puedes trabajar con lo que cabe en la mesa.
- Cuando necesitas un libro nuevo que está en la estantería, sacas uno de la mesa y lo cambias por el otro.
- Aunque la estantería guarda más información, el trabajo real siempre lo haces en la mesa.

Recursos recomendados

Para profundizar más en el uso de **swap en Linux**:

- Parte 1: <https://haydenjames.io/linux-performance-almost-always-add-swap-space/>
- Parte 2: <https://haydenjames.io/linux-performance-almost-always-add-swap-part2-zram/>

8. GESTIÓN DE ENTRADA / SALIDA

8.1. Formas de gestionar la entrada y salida

Principalmente, existen **tres formas** de gestionar las operaciones de Entrada/Salida en un sistema operativo:

1. Entrada/Salida Programada (Programmed I/O)

- Cada proceso de usuario debe comprobar periódicamente, consumiendo tiempo de CPU, si una operación de E/S se ha completado.
- Cuando detecta que la operación terminó, recoge los resultados.

- Es una técnica muy ineficiente porque obliga a la CPU a “preguntar constantemente” al dispositivo.
- Solo resulta viable en sistemas **monotarea** (un único proceso en ejecución).

Ejemplo: un programa antiguo esperando a que una impresora termine, revisando en bucle el estado de “¿ya terminó?”.

2. Interrupciones (Interrupts)

- El sistema operativo configura el dispositivo de E/S y **espera una señal**.
- Cuando la operación de E/S finaliza, el dispositivo interrumpe la CPU y notifica al proceso.
- De este modo, un proceso en espera de E/S no consume CPU (pasa al estado **Waiting**).
- La CPU solo interviene para procesar el resultado cuando la interrupción ocurre.

Ejemplo: cuando conectas un USB, el sistema no está comprobando continuamente, sino que el dispositivo envía una señal de interrupción para avisar que ya está disponible.

3. Acceso Directo a Memoria (DMA, Direct Memory Access)


- Es una técnica combinada con interrupciones.
- Permite que ciertos dispositivos (como discos duros o tarjetas de red) transfieran datos directamente a la RAM sin pasar por la CPU en cada operación.
- La CPU solo interviene al inicio y al final del proceso, reduciendo su carga de trabajo.

Ejemplo: copiar un archivo grande del disco a la memoria, donde el controlador DMA hace la transferencia, y la CPU queda libre para otras tareas.

8.2. Técnicas para incrementar el rendimiento de la E/S

En general, los dispositivos de E/S son **mucho más lentos** que la CPU. Para reducir este cuello de botella se aplican varias técnicas:

- **Uso de cachés:**
Cuando el sistema operativo quiere escribir datos en un disco duro, no espera a que la operación finalice físicamente.
Escribe la información en la caché y considera que la operación “ya está hecha”.
Posteriormente, el hardware se encarga de grabar la información realmente en el disco.

 **Ejemplo cotidiano:** este mecanismo explica por qué es peligroso apagar un ordenador de forma brusca o extraer un USB sin “expulsarlo con seguridad”. Puede que la información aún no se haya grabado físicamente en el dispositivo.

9. GESTIÓN DEL SISTEMA DE ARCHIVOS

9.1. Estructura de un sistema de archivos

El **sistema de archivos** es el componente del sistema operativo que organiza y gestiona cómo se almacena la información en un dispositivo de almacenamiento (como un disco duro).

- Externamente, la mayoría de sistemas de archivos utilizan una estructura de **árbol invertido** para organizar los contenidos.
- Existen dos tipos principales de objetos:
 - **Archivo:** almacena información (texto, imágenes, programas, etc.).
 - **Directorio:** es un archivo especial que contiene referencias a otros archivos o directorios.
- El directorio principal se llama **raíz (root)** y es el inicio del árbol.



9.2. Atributos de un sistema de archivos

- **Tamaño máximo de partición:** límite que puede alcanzar una partición.
- **Tamaño máximo de archivo:** tamaño máximo que puede tener un solo archivo.
- **Tamaño de clúster:** unidad mínima de almacenamiento usada por el disco.
 - Si el clúster es grande: mejor rendimiento, pero mayor desperdicio de espacio.
 - Si el clúster es pequeño: menor pérdida de espacio, pero peor rendimiento.



Ejemplo:

Si un archivo de 10 KB se guarda en un disco con clústeres de 8 KB, ocupará 2 clústeres (16 KB), desperdiciando 6 KB. Además, esos clústeres pueden no estar contiguos, obligando a realizar varios accesos físicos al disco, lo que ralentiza la operación.

9.3. Organización interna

Para localizar los archivos, los sistemas de archivos mantienen una **tabla de asignación** (como FAT, inodos, etc.), que guarda dónde está cada fragmento de archivo en el disco.

Ejemplos de sistemas de archivos: **FAT16, FAT32, NTFS, ext3, ext4, HPFS, HFS+**, etc.

Un mismo sistema operativo puede trabajar con varios, pero cada partición solo puede tener uno.

Cada sistema de archivos define sus características:

- Seguridad frente a fallos.
- Límite de tamaño de archivo.
- Límite de longitud de nombres.
- Número máximo de subdirectorios, etc.

Conceptos clave

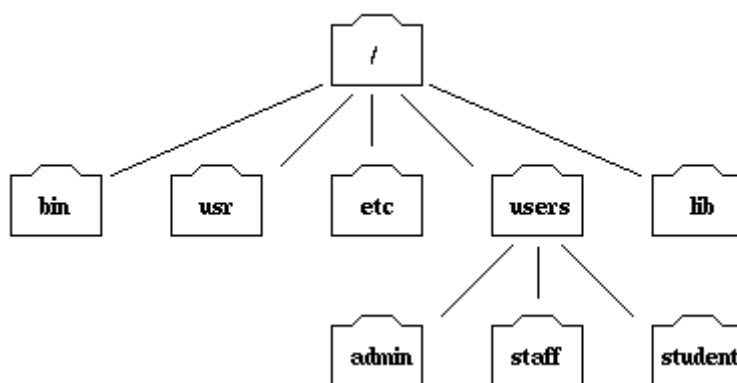
- **Partición:** división lógica de un disco duro. Sirve para organizar datos y mejorar la seguridad. Es posible formatear una partición sin afectar al resto del disco.
- **Formato (format):** no significa “borrar” un disco, aunque borra los datos como consecuencia. Lo que realmente hace es **reorganizar desde cero la tabla del sistema de archivos**, es decir, preparar el disco para usarse de nuevo.

9.4. Rutas absolutas y relativas

Cuando navegamos por un sistema de archivos y queremos referirnos a un directorio o un archivo, podemos hacerlo de dos formas principales: **usando rutas absolutas** o **usando rutas relativas**.

Ruta absoluta

- Es la ruta completa que parte siempre desde el **directorio raíz**.
- Indica de manera única y exacta dónde se encuentra un archivo o carpeta dentro del sistema, sin importar en qué directorio estemos situados.
- En sistemas **Unix/Linux** siempre comienza con **/**.
- En Windows, las rutas absolutas comienzan con la letra de la unidad seguida de ****, por ejemplo: **C:\Users\Admin\Documentos**.



Ejemplos:

- Unix/Linux: `/users/admin` → apunta a la carpeta *admin* dentro de *users*, que cuelga directamente del directorio raíz `/`.
- Windows: `C:\Users\Admin\Documentos` → apunta a la carpeta *Documentos* dentro de *Admin* en la unidad C:.

Ruta relativa

- Se define **en función del directorio actual en el que nos encontramos**.
- No comienza desde la raíz, sino que describe cómo llegar a un archivo o carpeta **a partir de nuestra posición actual** en el árbol de directorios.
- Es más corta y práctica cuando trabajamos en entornos donde no necesitamos escribir siempre la ruta completa.

Símbolos especiales en rutas relativas:

- `.` → representa el **directorio actual**.
- `..` → representa el **directorio padre** (un nivel superior).
- `~` → en Unix/Linux, representa el **directorio personal del usuario** (por ejemplo, `/home/usuario`).

Ejemplos prácticos en Windows:

1. Si estamos en `C:\Users\Juan\Documentos` y queremos ir a `C:\Users\Juan\Imágenes`:
 - Ruta relativa: `..\Imágenes`
2. Si estamos en `C:\Windows\System32` y queremos volver a la raíz de la unidad:

- Ruta relativa: `.. \ .`

3. Si estamos en `D:\Proyectos\2025\Codigo` y queremos acceder a `D:\Proyectos\2025\Documentos`:

- Ruta relativa: `..\Documentos`

Ejemplo práctico en Unix/Linux para comparación:

- Estamos en `/bin` y queremos acceder a `/users/admin`:
 - Ruta relativa: `../users/admin`
- Estamos en `/` y queremos acceder a `admin`:
 - Ruta relativa: `users/admin` o `./users/admin`

Notas importantes:

- En **Windows**, las rutas pueden usar `/` en algunos contextos (PowerShell o WSL), aunque tradicionalmente se usa `\`.
- Comprender la diferencia entre **rutas absolutas y relativas** es fundamental para:
 - Administrar archivos desde la terminal o el explorador de archivos.
 - Programación de scripts y automatización.
 - Navegación eficiente en proyectos con muchos directorios.

9.5. Características de los sistemas de ficheros más populares

| | FAT16 | FAT32 | NTFS | ext4 |
|-----------------------------------|----------------------------|---|---|------------------|
| Sistema operativo | MS-DOS 6.22, Windows 9X | Windows 9X, Windows Server, Windows XP/7/10/11 | Windows Server, Windows XP/7/10/11 | Linux |
| Máximo tamaño de fichero | 2 GiB | 4 GiB | Tamaño limitado por el tamaño del volumen | 16 GiB to 16 EiB |
| Máximo tamaño de partición | 2 GiB | 2 TiB | Tamaño limitado por el tamaño del volumen | 1 EiB |

10. MATERIAL ADICIONAL

[1] Tutorial de sistemas operativos (avanzado)

http://www.tutorialspoint.com/operating_system/

[2] Solucionador de problemas de planificación de procesos

<http://uhurulabs.com/PlanificadoresCPU/>

11. BIBLIOGRAFÍA

[1] Operating systems

https://en.wikipedia.org/wiki/Operating_system

[2] Process management

https://en.wikipedia.org/wiki/Process_management

[3] Memory management

https://en.wikipedia.org/wiki/Memory_management

[4] Device management

https://es.wikipedia.org/wiki/Device_Management

[5] File system

https://en.wikipedia.org/wiki/File_system

[6] Comparison of file systems

https://en.wikipedia.org/wiki/Comparison_of_file_systems