Fundamentos de los sistemas operativos

Funciones, procesos, planificación, memoria, archivos y entrada/salida

Josep Garcia

Sistemas Informáticos · 1 DAM

Curso 2026-2027Curso 2026-2027 — Sincronizado: 22/10/2026 11:20

Índice general

Introducción . . . 3
Objetivos . . . 3
Relación entre bloques, resultados de aprendizaje y criterios de evaluación . . . 3
Prácticas individuales de Aules . . . 4
Diapositivas y diagramas . . . 5
1.1. Introducción al software de base . . . 5
1.2. Arquitectura básica: kernel, espacio de usuario, shell e interfaz . . . 6
1.2.1. Kernel y espacio de usuario . . . 6
1.2.2. Controladores y servicios . . . 8
1.2.3. Shell, terminal e interfaz gráfica . . . 8
1.3. Funciones generales del sistema operativo . . . 9
1.4. Formas de organizar el núcleo . . . 9
1.5. Actividad 01.1. ¿Qué funciones del SO intervienen? . . . 10
1.6. Actividad 01.2. Corrige la explicación técnica . . . 11
1.7. Actividad 01.3. Compara las organizaciones del núcleo . . . 12
2.1. Evolución de los sistemas operativos . . . 13
2.2. Clasificación de los sistemas operativos . . . 14
2.2.1. Según el número de tareas . . . 14
2.2.2. Según el número de personas usuarias . . . 14
2.2.3. Según los procesadores o núcleos . . . 14
2.2.4. Según la respuesta temporal . . . 15
2.2.5. Según el ámbito de uso . . . 15
2.3. Aplicaciones, requisitos y licencias . . . 15
2.3.1. Software según su propósito . . . 15
2.3.2. Requisitos y compatibilidad . . . 16
2.3.3. Licencias de software . . . 17
2.4. Comparación de sistemas operativos según ámbito de uso . . . 19
2.4.1. Criterios de comparación . . . 19
2.4.2. Casos de uso profesionales . . . 20
2.5. Actividad 02.1. Propósito y licencia . . . 21
2.6. Actividad 02.2. Selección para dos entornos AULES . . . 21
2.6.1. Antes de entregar y criterios de corrección . . . 23
3.1. Qué es un proceso . . . 24
3.2. Procesos e hilos . . . 24
3.3. Estados de un proceso . . . 24
3.4. Colas, bloqueo y cambio de contexto . . . 25

3.5. Concurrencia y paralelismo . . . 26
3.6. Actividad 03.1. Un proceso espera datos . . . 26
3.7. Actividad 03.2. Programa, proceso e hilo . . . 26
4.1. Planificador a corto plazo . . . 27
4.2. Algoritmo FCFS/FIFO . . . 27
Ejemplo resuelto 1: FCFS . . . 28
4.3. Algoritmo SJF/SRTF . . . 28
Ejemplo resuelto: SJF no expropiativo . . . 29
Ampliación opcional. Ejemplo resuelto 2: SRTF . . . 29
4.4. Algoritmo Round Robin . . . 30
Ejemplo resuelto 3: Round Robin con quantum 2 . . . 30
4.5. Diagramas de Gantt y métricas básicas . . . 31
Ejemplo parcialmente resuelto: ¿qué algoritmo favorece más a P3? . . . 31
4.6. Comparación de algoritmos de planificación . . . 32
4.7. Planificación por prioridades . . . 33
Ejemplo resuelto: prioridades no expropiativas . . . 33
4.8. Actividad 04.1. Una misma carga, distintas decisiones . . . 34
4.9. Actividad 04.2. El tamaño del turno . . . 35
4.10. Actividad 04.3. Elegir una política . . . 35
5.1. Asignar, proteger y liberar memoria . . . 36
5.2. Direcciones virtuales y físicas . . . 36
5.3. Páginas, marcos y tabla de páginas . . . 37
Ejemplo resuelto: localizar un byte . . . 37
5.4. Cuando una página no está presente . . . 38
5.5. Intercambio y presión de memoria . . . 38
5.6. Actividad 05.1. Traducción con una tabla completa . . . 39
5.7. Actividad 05.2. Diagnosticar una idea equivocada . . . 40
6.1. Datos persistentes y sistema de archivos . . . 40
6.2. Árbol y rutas . . . 40
6.3. Sistemas de archivos, particiones y formato . . . 42
6.4. Entrada/salida: del servicio al dispositivo . . . 43
6.4.1. Consulta del estado e interrupciones . . . 44
6.4.2. Acceso directo a memoria . . . 44
6.5. Búferes, caché y cola de impresión . . . 44
6.6. Actividad 06.1. Leer un árbol, sin administrar discos . . . 45
6.7. Actividad 06.2. Una lectura que tarda . . . 45
6.8. Práctica integrada. Funcionamiento del SO AULES . . . 45
6.8.1. Procesos e hilos . . . 46
6.8.2. Decisiones de CPU . . . 46
6.8.3. Traducción de memoria . . . 46
6.8.4. Cooperación durante una lectura . . . 46
6.8.5. Antes de entregar y criterios de corrección . . . 46

Introducción

Partimos del hardware estudiado en UD01 y UD02 para comprender cómo el sistema operativo organiza CPU, memoria, archivos y dispositivos. Analizaremos sus funciones, tipos y licencias, y resolveremos casos básicos de procesos, planificación y paginación. Este recorrido prepara las decisiones y las instalaciones de UD04.

Las actividades de clase se trabajan y corrigen allí, **sin nota**. Hay dos prácticas individuales aprobadas para Aules que reutilizan productos de los bloques y alimentan el 30 % de actividades/prácticas de sus CE. Cada práctica recibe una única nota; las fechas se comunicarán en Aules.

Objetivos

Al finalizar la unidad, serás capaz de:

- Explicar las funciones y la arquitectura del SO, distinguiendo núcleo, modos de ejecución, espacio de usuario, llamadas al sistema, controladores, servicios e interfaces.
- Comparar sistemas operativos por requisitos, características, campos de aplicación, soporte y licencias.
- Diferenciar programa, proceso e hilo e interpretar los estados de un proceso.
- Resolver casos de planificación mediante FCFS, SJF, Round Robin y prioridades, justificando diagramas de Gantt y métricas.
- Explicar la protección y traducción de memoria y resolver un caso sencillo de paginación sin confundir memoria virtual, RAM y swap.
- Relacionar archivos, directorios y entrada/salida con los servicios que utiliza una aplicación.
- Seleccionar sistemas y aplicaciones para necesidades concretas de un entorno productivo, identificando su propósito y modelo de licencia y justificando una alternativa descartada.

Relación entre bloques, resultados de aprendizaje y criterios de evaluación

Bloque	Qué se trabaja y se comprueba	RA/CE directos
01. Funciones y arquitectura	Relación entre situaciones y funciones, corrección de explicaciones y comparación de organizaciones del núcleo (01.1–01.3).	**RA2.b**
02. Tipos, licencias y selección	Comparación de alternativas, clasificación por propósito y licencia y elección justificada para dos entornos.	**RA2.c**, **RA7.a**, **RA7.b**

Bloque	Qué se trabaja y se comprueba	RA/CE directos
03. Procesos, hilos y estados	Identificación del contexto de ejecución y explicación de estados y transiciones.	**RA2.b**
04. Planificación de CPU	Aplicación y comparación de algoritmos mediante Gantt y métricas.	**RA2.b**
05. Memoria y paginación	Traducción de direcciones y explicación de protección, residencia y fallo de página.	**RA2.b**
06. Archivos y E/S	Explicación de persistencia, organización conceptual y cooperación entre procesos y dispositivos.	**RA2.b**

La tabla identifica la aportación directa de los contenidos y actividades al aprendizaje; no certifica ejecución ni una calificación ya obtenida. RA2.c comienza aquí con la comparación de alternativas y continúa en UD04 al concretar versiones, requisitos y decisiones de instalación; conserva un único peso.

Los saberes de apoyo recuperan hardware y memorias de UD01/UD02. La introducción conceptual de archivos no acredita por sí sola administración de discos o copias, que corresponde a UD07.

Prácticas individuales de Aules

Entrega	Producto	CE vinculados
Selección para dos entornos, apartado 2.6 (02.2)	Una tabla y dos párrafos justificativos.	RA7.a y RA7.b principales; RA2.c relacionado.
Funcionamiento del SO, apartado 6.8	03.2; FCFS y RR de 04.1; 05.1; 06.2, reunidos en un solo documento.	RA2.b.

Las demás actividades siguen siendo de clase. Los productos de preparación de la segunda práctica no generan entregas ni notas separadas. Los criterios de corrección están al final de cada práctica.

Diapositivas y diagramas

Presentación	Diapositivas	Fuente
Funciones y arquitectura	Abrir	Markdown editable
Tipos, licencias y selección: síntesis	Abrir	Markdown editable
Procesos, hilos y estados	Abrir	Markdown editable
Planificación de CPU	Abrir	Markdown editable
Memoria y paginación	Abrir	Markdown editable
Archivos y entrada/salida	Abrir	Markdown editable

Colección de presentaciones · Galería de diagramas.

Como apoyo al bloque 04 puedes usar el simulador de planificación. Incluye FCFS, SJF y Round Robin; SRTF permite explorar la ampliación. La política por prioridades se comprueba con los datos y las reglas de los apuntes, ya que el simulador no la implementa.

Abrir las diapositivas de apoyo: Funciones y arquitectura del sistema operativo

1.1. Introducción al software de base

En UD01 seguimos cómo cooperan CPU, memoria y entrada/salida; en UD02 localizamos esas funciones en el equipo real y comprobamos su puesta en marcha. Ahora suponemos que el equipo ya dispone de un sistema operativo: queremos comprender cómo permite que varias aplicaciones compartan ese hardware sin tener que controlarlo por su cuenta.

Un sistema informático combina **hardware** y **software**. El hardware son los componentes físicos: CPU, memoria, placa base, disco, pantalla o red. El software es el conjunto de instrucciones y datos que hacen que ese hardware realice tareas útiles.

El software puede organizarse en tres grupos principales:

Tipo	Función	Ejemplos
Firmware	Inicia o controla un dispositivo concreto desde memoria no volátil.	UEFI, firmware de un router, firmware de una impresora.
Software de base	Gestiona recursos y proporciona servicios generales.	Sistema operativo, controladores, utilidades del sistema.
Software de aplicación	Resuelve una necesidad de la persona usuaria.	Navegador, entorno integrado de desarrollo (IDE), editor de imágenes, hoja de cálculo.

El **sistema operativo** (SO) es el componente central del software de base. Tiene dos papeles complementarios:

- Es un **administrador de recursos**: reparte CPU, memoria, almacenamiento y dispositivos entre los programas.
- Es una **máquina extendida**: oculta detalles físicos mediante abstracciones más cómodas, como archivos, ventanas, procesos, cuentas o impresoras.

Por ejemplo, al guardar un archivo, una aplicación no necesita decidir en qué sectores físicos del disco escribir. Solicita ese servicio al sistema operativo, que organiza el sistema de archivos y utiliza el controlador apropiado.



Un controlador o *driver* es software que permite al sistema operativo comunicarse con una familia concreta de dispositivos. No es una aplicación de uso cotidiano, sino una pieza de software de base.

Esta división permite entender la evolución de los sistemas operativos: cuanto más complejo es el hardware y más personas o programas deben usarlo, más necesaria resulta una capa que lo organice.

1.2. Arquitectura básica: kernel, espacio de usuario, shell e interfaz

Los sistemas operativos separan las funciones con mayor privilegio de las aplicaciones ordinarias. Esta separación protege el equipo: un error de un editor de texto no debería poder escribir directamente sobre el disco ni detener cualquier proceso del sistema.

1.2.1. Kernel y espacio de usuario

El **kernel** o núcleo es la parte central del sistema operativo. Se ejecuta con privilegios elevados y administra recursos críticos: CPU, memoria, dispositivos, comunicaciones y seguridad básica.

Las aplicaciones ordinarias utilizan un **espacio de usuario** separado del espacio reservado al núcleo. Estos espacios describen la organización y protección de la memoria. Los **modos de ejecución** describen, en cambio, qué operaciones permite realizar la CPU: el código de una aplicación se ejecuta normalmente en **modo usuario**; el núcleo atiende operaciones protegidas en **modo núcleo**. Son ideas relacionadas, pero espacio y modo no son sinónimos.

Tampoco equivalen a los permisos de una cuenta. Una aplicación abierta por una persona administradora sigue ejecutándose normalmente en modo usuario: disponer de autorización para modificar un archivo no permite ejecutar por ello cualquier instrucción privilegiada de la CPU.

Cuando una aplicación necesita una operación protegida, solicita una **llamada al sistema**. Por ejemplo, abrir un archivo, crear un proceso o reservar memoria requiere que el sistema gestione la petición;

algunas reservas se satisfacen desde memoria ya disponible en el proceso. Una **interfaz de programación de aplicaciones** o API ofrece funciones que un programa puede utilizar; una biblioteca puede apoyarse en llamadas al sistema para atenderlas.

```
Aplicación -> biblioteca o API -> llamada al sistema -> kernel -> hardware
```



Una aplicación de usuario, como un navegador, no puede escribir directamente en el disco ni controlar un dispositivo por su cuenta. Debe solicitarlo mediante una llamada al sistema para que el kernel, en modo privilegiado, compruebe permisos y actúe en su nombre. Esta separación limita el alcance de los fallos y protege los recursos compartidos.

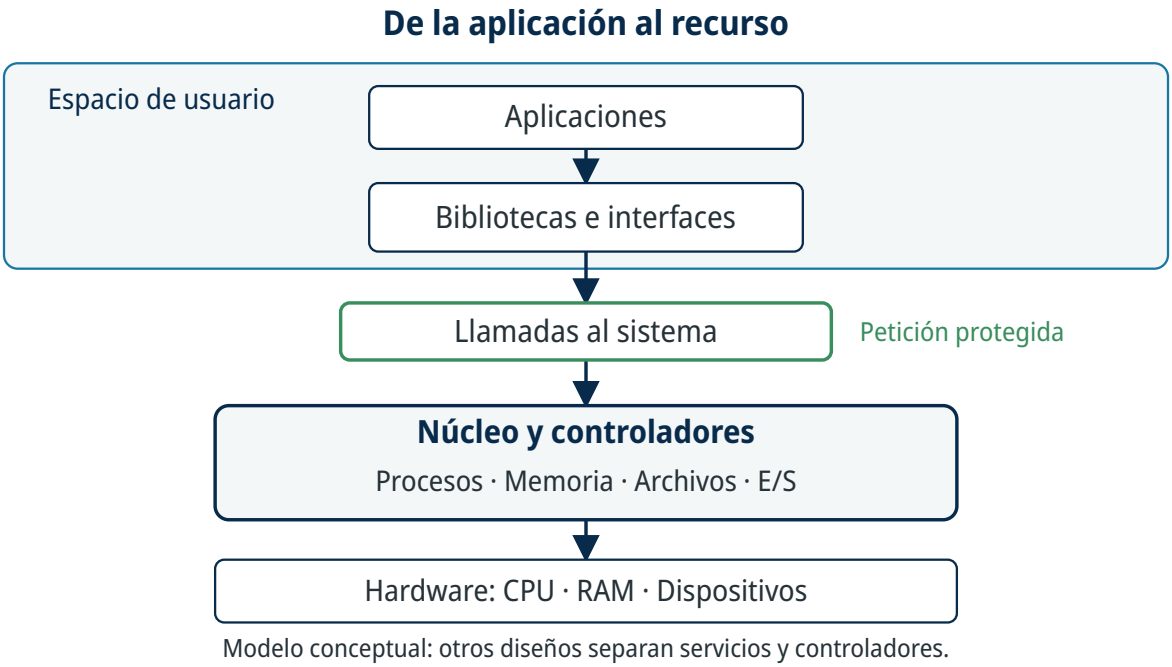


Figura 1: Las aplicaciones en espacio de usuario solicitan servicios mediante llamadas al sistema; núcleo y controladores gestionan procesos, memoria, archivos y dispositivos.

Figura 1.1. De la aplicación al recurso. Lee el esquema desde la petición hacia el hardware. Es un modelo conceptual próximo al Linux del laboratorio: agrupa núcleo y controladores para explicar su cooperación, sin afirmar que todos los SO sitúen sus componentes del mismo modo.

Por ejemplo, un visor ya abierto necesita **leer una imagen** del SSD para mostrarla:

1. El visor solicita abrir y leer el archivo mediante las funciones de una biblioteca; las peticiones necesarias llegan a la interfaz de llamadas al sistema.

2. El mecanismo de entrada al núcleo permite ejecutar el código del SO en modo núcleo. El SO comprueba la petición y si la identidad del proceso está autorizada para acceder al archivo.
3. El sistema de archivos localiza los datos. El SO puede conservar en RAM una copia de datos leídos anteriormente: es una **caché de datos del SO**, distinta de la caché de CPU estudiada en UD01. Si la copia está disponible, puede atender la lectura sin acceder de nuevo al SSD.
4. Si hace falta una lectura física, el SO coordina la operación con el controlador del dispositivo. La aplicación no elige sectores ni maneja directamente el SSD.
5. Cuando los datos están disponibles, el SO los hace accesibles al visor según la interfaz utilizada. La aplicación continúa en modo usuario y puede interpretar la imagen.

El ejemplo simplifica las operaciones internas y no supone que abrir, leer y mostrar sean una única llamada. Su idea central es que **la aplicación pide un servicio, el SO comprueba y organiza el acceso y el hardware realiza la operación necesaria**.

1.2.2. Controladores y servicios

Los **controladores** traducen las órdenes generales del sistema en operaciones entendibles por un dispositivo concreto. Los **servicios** son procesos que trabajan en segundo plano ofreciendo funciones, como impresión, red, actualización o registro de eventos.

Un servicio no tiene por qué mostrar una ventana. Puede iniciarse al arrancar el sistema y permanecer a la espera de peticiones.

1.2.3. Shell, terminal e interfaz gráfica

La persona usuaria necesita una forma de comunicarse con el sistema operativo:

- El **shell** interpreta órdenes y ejecuta programas. Puede ser textual o formar parte de un entorno más amplio.
- La **terminal** es la aplicación que proporciona una ventana para usar un shell de texto.
- La **interfaz gráfica de usuario** (GUI) permite trabajar con ventanas, iconos, menús y puntero.

El shell no es el kernel. Ambos pueden trabajar juntos, pero cumplen funciones distintas: el shell recibe órdenes de la persona usuaria; el kernel administra los recursos necesarios para ejecutarlas.



En unidades posteriores se usarán comandos para administrar sistemas. Aquí interesa comprender la relación entre la orden, el proceso que crea y el servicio o recurso que solicita.

1.3. Funciones generales del sistema operativo

La arquitectura anterior explica quién puede hacer qué. Ahora podemos agrupar las funciones del sistema operativo según el recurso que administra.

Función	Recurso o problema	Ejemplo visible
Gestión de procesos	CPU y ejecución concurrente.	Alternar navegador, IDE y reproductor.
Gestión de memoria	RAM y memoria virtual.	Evitar que un programa invada la memoria de otro.
Gestión de E/S	Teclado, red, disco, impresora, pantalla.	Recibir una pulsación o guardar un archivo.
Sistema de archivos	Organización persistente de datos.	Carpetas, nombres, permisos y directorios.
Gestión de usuarios y seguridad	Identidad, permisos y aislamiento.	Iniciar sesión con una cuenta y limitar accesos.
Interfaz y servicios	Comunicación con la persona usuaria y aplicaciones.	Escritorio, terminal, impresión o actualización.

Estas funciones no son compartimentos estancos. Abrir un documento puede crear un proceso, reservar memoria, leer datos del almacenamiento, mostrar una ventana y comprobar permisos. En los siguientes apartados veremos primero el reparto de CPU y después el resto de recursos de forma introductoria.

1.4. Formas de organizar el núcleo

La separación entre usuario y núcleo explica los privilegios, pero no determina por sí sola cómo se distribuye el código interno del SO. En un **núcleo monolítico**, muchos servicios de gestión y controladores funcionan en el espacio del núcleo. Puede admitir módulos cargables: monolítico no significa que todo sea un único archivo fijo. Linux es un ejemplo de núcleo monolítico modular.

Un **micronúcleo** conserva en modo privilegiado un conjunto reducido de funciones y sitúa otros servicios en procesos separados. Estos se comunican mediante mensajes. La separación puede facilitar el aislamiento de fallos, aunque introduce comunicación adicional. Los diseños **híbridos** combinan decisiones de ambos modelos. Estas categorías describen organización; no permiten concluir que un sistema sea siempre más rápido o más seguro solo por su etiqueta.

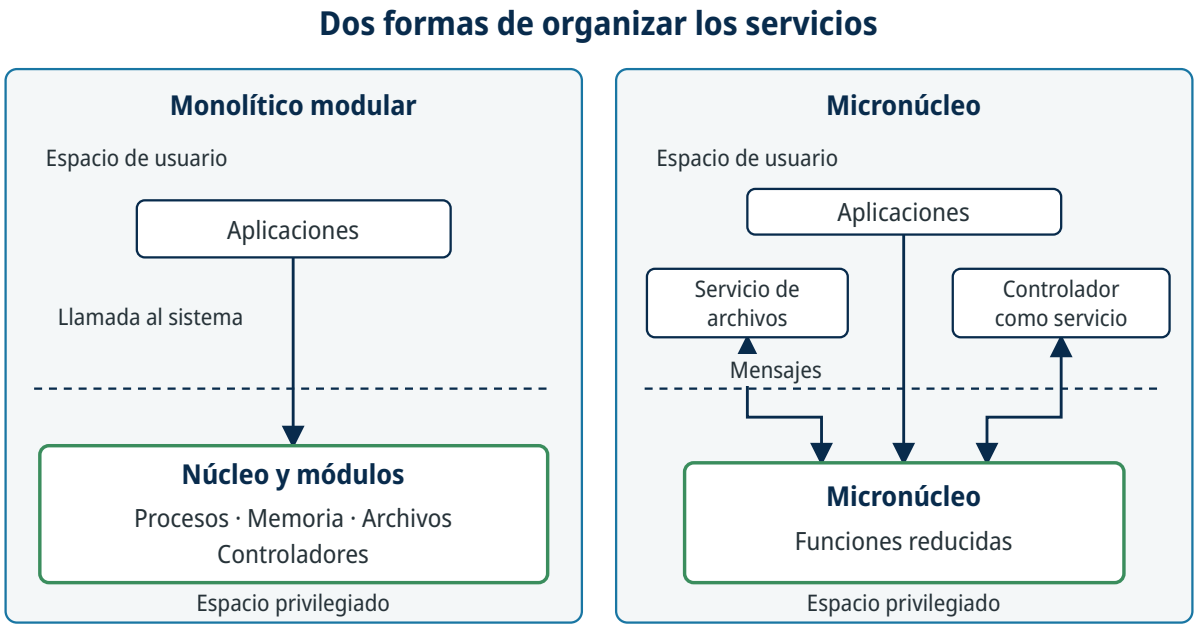


Figura 2: El núcleo monolítico modular reúne numerosos servicios y controladores en espacio privilegiado; el micronúcleo mantiene funciones reducidas y comunica mediante mensajes servicios separados.

Figura 1.2. Dos organizaciones del núcleo. Compara dónde se sitúan los servicios y cómo se comunican, manteniendo la separación entre código de usuario y código privilegiado. El esquema es funcional: no representa todos los componentes ni todas las variantes de cada familia.

En el panel izquierdo, la aplicación solicita una operación al núcleo, que reúne gestión de procesos, memoria, archivos y controladores; los módulos amplían ese núcleo sin dejar por ello de ejecutarse con sus privilegios. En el derecho, el servicio de archivos y el controlador aparecen separados en espacio de usuario: cooperan mediante mensajes con el micronúcleo. La diferencia que interesa observar es **dónde se ejecutan las funciones y cómo se comunican**, no el número de cajas ni una supuesta superioridad de un modelo.

1.5. Actividad 01.1. ¿Qué funciones del SO intervienen?

Actividad de clase, sin nota · RA2.b.

Relaciona las situaciones con las funciones del apartado 1.3: gestión de procesos, gestión de memoria, gestión de E/S, sistema de archivos, gestión de usuarios y seguridad, e interfaz y servicios. En una situación puede intervenir más de una función; indica las que consideres necesarias y explica su papel.

Situación	Funciones del SO que intervienen	¿Qué aporta cada una?
Mantienes abiertos un navegador y un editor, y ambos necesitan ejecutar instrucciones.		
Una aplicación solicita espacio de RAM para trabajar con sus datos.		
Abres un documento guardado en el SSD.		
Envías un documento a una impresora.		
Inicias sesión con tu cuenta y accedes a tus archivos.		

Completa la tabla. Después elige dos situaciones en las que cooperen varias funciones y explica esa cooperación en una o dos frases por caso. No basta con enumerar nombres: relaciona cada función con una necesidad de la situación.

1.6. Actividad 01.2. Corrige la explicación técnica

Actividad de clase, sin nota · RA2.b.

Un compañero ha escrito estas afirmaciones. Las cuatro contienen un error: reescribe cada una de forma correcta y explica brevemente qué conceptos estaba confundiendo.

Afirmación que debes corregir	Explicación corregida	¿Cuál era la confusión?
«El shell es el núcleo del sistema operativo: interpreta órdenes y administra directamente todos los recursos críticos».		
«Una aplicación abierta con una cuenta administradora ejecuta siempre sus instrucciones en modo núcleo».		

Afirmación que debes corregir	Explicación corregida	¿Cuál era la confusión?
«Cada aplicación controla directamente sus dispositivos, sin solicitar servicios al sistema operativo».		
«Todos los servicios del sistema muestran una ventana para poder funcionar».		

Usa las distinciones entre shell y núcleo, permisos de cuenta y modos de ejecución, aplicación y controlador, y servicio e interfaz. Completa las dos columnas; señalar únicamente que una frase es falsa no explica el error.

1.7. Actividad 01.3. Compara las organizaciones del núcleo

Actividad de clase, sin nota · RA2.b.

Observa la **figura 1.2 del apartado 1.4**, que compara un núcleo monolítico modular y un micronúcleo. Responde tomando como referencia lo que representa el dibujo:

1. ¿Dónde se sitúan los controladores y el servicio de archivos en cada panel? Distingue espacio de usuario y espacio privilegiado.
2. ¿Cómo se comunican la aplicación, el núcleo y los servicios en cada organización? Apóyate en las flechas y etiquetas de la figura.
3. ¿Por qué permitir cargar módulos no convierte un núcleo monolítico en micronúcleo? Explica qué diferencia organizativa seguiría existiendo.

Escribe tres respuestas breves y razonadas. No necesitas buscar productos ni decidir qué modelo es «mejor»: compara la ubicación de las funciones y su forma de comunicación.

Ahora que conocemos los servicios del SO, el siguiente bloque permitirá elegir qué sistema y qué aplicaciones responden a un escenario concreto.

Abrir las diapositivas de apoyo: Tipos, licencias y selección de software

En el bloque anterior vimos los servicios y la arquitectura del SO. Su evolución ayuda a entender por qué existen sistemas con prioridades distintas; sus licencias y requisitos permiten elegir una solución para cada entorno.

2.1. Evolución de los sistemas operativos

Los sistemas operativos no aparecieron de golpe. Respondieron a problemas concretos de cada época: aprovechar equipos muy caros, permitir varios trabajos, facilitar el uso personal y conectar dispositivos muy distintos.

Etapa	Necesidad principal	Aportación
Equipos sin SO	Ejecutar un único programa preparado manualmente.	La persona programadora controlaba directamente el hardware.
Procesamiento por lotes	Reducir tiempos muertos entre trabajos.	Los trabajos se agrupaban y ejecutaban uno tras otro.
Multiprogramación	Mantener la CPU ocupada mientras un programa esperaba E/S.	Varios programas podían residir en memoria y alternarse.
Tiempo compartido	Dar respuesta a varias personas usuarias.	Reparto rápido de CPU y terminales interactivos.
Sistemas personales e interfaz gráfica	Hacer el ordenador accesible a público no especializado.	Escritorio, ventanas, ratón y aplicaciones de propósito general.
Redes, móviles y nube	Conectar equipos, servicios y dispositivos variados.	Administración remota, movilidad, virtualización y servicios distribuidos.

En los primeros ordenadores, preparar una ejecución podía exigir cablear paneles o cargar manualmente soportes físicos. El tiempo de la CPU era valioso y no debía quedar inactivo mientras se cambiaba de programa. Los sistemas por lotes automatizaron parte de esa secuencia.

La multiprogramación introdujo una idea fundamental para esta unidad: si un proceso espera la lectura de un disco o la llegada de datos, la CPU puede atender a otro. Esto hace necesario saber qué procesos existen, en qué estado están y cuál debe ejecutarse. La planificación de CPU nace precisamente de ese problema.

Unix consolidó una arquitectura multitarea y multiusuario que influyó en muchos sistemas posteriores. GNU aportó numerosas herramientas y utilidades libres; el kernel Linux es el núcleo que, combinado habitualmente con componentes GNU y otros proyectos, forma distribuciones GNU/Linux. En paralelo, los sistemas personales popularizaron las interfaces gráficas y los sistemas móviles adaptaron el SO a pantallas táctiles, sensores y consumo energético limitado.



Llamar “Linux” a todo el sistema puede ser útil en una conversación informal, pero conviene distinguir entre el **kernel Linux** y una distribución completa, que incluye herramientas, bibliotecas, instalador, gestor de paquetes e interfaz.

La historia explica por qué hoy hay sistemas operativos tan distintos. El siguiente paso es clasificarlos según cómo administran los recursos y para qué contexto se han diseñado.

2.2. Clasificación de los sistemas operativos

Un mismo sistema operativo puede pertenecer a varias categorías. Por ejemplo, un servidor GNU/Linux puede ser multitarea, multiusuario, multiprocesador y orientado a red al mismo tiempo.

2.2.1. Según el número de tareas

- **Monotarea:** ejecuta una tarea principal cada vez. Su diseño puede ser sencillo y predecible, pero limita la interacción.
- **Multitarea:** mantiene varios procesos y alterna la CPU entre ellos. En un equipo actual, navegador, editor, antivirus y servicios pueden ejecutarse de forma concurrente.

La multitarea no significa que todos los programas se ejecuten literalmente a la vez en un único núcleo. El sistema alterna tan rápido entre ellos que la persona usuaria percibe simultaneidad.

2.2.2. Según el número de personas usuarias

- **Monousuario:** diseñado principalmente para una persona usuaria activa en un momento dado.
- **Multiusuario:** permite cuentas separadas, sesiones y recursos compartidos con aislamiento y permisos.

Un equipo personal puede tener varias cuentas locales, pero un sistema multiusuario adquiere especial importancia cuando varias personas acceden simultáneamente a un servidor.

2.2.3. Según los procesadores o núcleos

- **Monoprocesador:** diseñado para trabajar con una CPU o un núcleo de ejecución.
- **Multiprocesador:** aprovecha varios procesadores o núcleos para ejecutar tareas en paralelo.

La planificación de procesos sigue siendo necesaria en ambos casos. Con varios núcleos, el sistema debe decidir además qué proceso asignar a cada núcleo y equilibrar la carga.

2.2.4. Según la respuesta temporal

- **Sistemas de propósito general:** buscan buen rendimiento y buena experiencia global, sin garantizar un plazo exacto para cada tarea.
- **Sistemas de tiempo real:** deben responder dentro de un límite temporal definido. Se usan, por ejemplo, en control industrial, automoción o dispositivos médicos.



Tiempo real no significa simplemente "rápido". Significa que cumplir el plazo es parte del funcionamiento correcto; una respuesta tardía puede equivaler a un fallo.

2.2.5. Según el ámbito de uso

Ámbito	Prioridades habituales	Ejemplos de familias
Escritorio	Interfaz gráfica, periféricos, ofimática, aplicaciones de usuario.	Windows, macOS, distribuciones GNU/Linux.
Servidor	Estabilidad, red, servicios, administración remota y seguridad.	GNU/Linux, Windows Server, BSD.
Móvil	Batería, sensores, conectividad móvil y tienda de aplicaciones.	Android, iOS.
Embebido	Tamaño reducido, consumo y función específica.	Linux embebido, FreeRTOS, sistemas propietarios.
Tiempo real	Respuesta determinista ante eventos.	QNX, VxWorks, variantes especializadas.

Esta clasificación permite pasar de una pregunta genérica, "¿qué sistema es mejor?", a una pregunta profesional: "¿qué sistema responde mejor a estos requisitos?".

2.3. Aplicaciones, requisitos y licencias

Una vez que el sistema operativo ofrece servicios, pueden instalarse aplicaciones para necesidades concretas. Elegir software exige valorar su finalidad, sus requisitos y las condiciones legales de uso.

2.3.1. Software según su propósito

Tipo	Finalidad	Ejemplos
Software de base	Hacer utilizable el hardware y ofrecer servicios comunes.	SO, drivers, utilidades de diagnóstico.
Software de aplicación	Resolver una tarea de usuario o de empresa.	Ofimática, diseño asistido por ordenador (CAD), navegador, gestión de recursos empresariales (ERP), editor de vídeo.
Software de desarrollo	Crear, probar y mantener software.	Entorno integrado de desarrollo (IDE), compilador, intérprete, depurador, control de versiones.

Una misma herramienta puede cubrir más de una necesidad. Por ejemplo, un IDE es software de desarrollo por su propósito, pero también es una aplicación que utiliza servicios del SO. La separación base/aplicación del bloque 01 describe su papel respecto al sistema; aquí destacamos qué tarea resuelve. Por eso clasificar un IDE como aplicación y precisar que se utiliza para desarrollo es compatible, siempre que expliques el eje utilizado.

2.3.2. Requisitos y compatibilidad

Una **máquina virtual (VM)** ofrece recursos virtuales para ejecutar un sistema invitado sobre un equipo anfitrión. La utilizaremos como laboratorio de instalación en UD04, a partir del bloque 07; por ahora interesa tener en cuenta que sus recursos proceden del equipo real.

Antes de instalar un sistema o una aplicación, hay que comprobar al menos:

- Arquitectura compatible: procesador y versión de 32 o 64 bits cuando aplique.
- Memoria RAM, almacenamiento disponible y requisitos gráficos.
- Versión del sistema operativo y actualizaciones necesarias.
- Controladores o periféricos compatibles.
- Dependencias: bibliotecas, entornos de ejecución, navegador o servicios requeridos.
- Necesidades de red, cuentas, permisos y licencia.

Los **requisitos mínimos** son las condiciones que declara el proveedor para una versión y un uso determinados. Cumplirlos no garantiza que el programa responda bien con cualquier carga; los **recomendados** orientan una experiencia de uso más adecuada. Antes de convertir una cifra en requisito, identifica producto, versión y documentación de origen. En un entorno profesional también importan soporte, mantenimiento, copias, compatibilidad con formatos y continuidad del proveedor o comunidad.

2.3.3. Licencias de software

La licencia define qué se permite hacer con un programa: usarlo, copiarlo, modificarlo, redistribuirlo o integrarlo con otros componentes. El precio no define por sí solo la licencia: un programa gratuito puede seguir siendo propietario, y uno libre puede ofrecer servicios de pago.

Modelo o licencia	Idea principal	Ejemplo de implicación
Propietario	El titular conserva el control del código y fija condiciones de uso.	Normalmente se acepta un contrato de licencia de usuario final.
Software libre	Garantiza libertades de uso, estudio, modificación y distribución bajo sus condiciones.	Puede auditarse y adaptarse, cumpliendo la licencia.
Código abierto	El código fuente está disponible bajo una licencia aprobada como abierta.	Permite revisar e integrar según las condiciones concretas.
Freeware	Puede utilizarse sin coste, pero no implica acceso al código fuente.	Gratuito no equivale a libre.
Shareware o prueba	Uso limitado en tiempo o funciones.	Puede requerir compra tras el periodo de prueba.
GPL	Licencia libre de tipo copyleft.	Las redistribuciones de obras derivadas deben conservar condiciones compatibles.
BSD o Apache	Licencias permisivas.	Permiten reutilización amplia manteniendo avisos y condiciones.
EULA	Contrato de licencia de usuario final.	Establece condiciones de uso de software propietario, y a veces de otros modelos.



> “Software libre” no significa “sin coste” y “código abierto” no significa automáticamente que cualquier uso esté permitido. Hay que leer las condiciones de la licencia concreta.

Ampliación: GPLv3, MIT y Apache 2.0 en un proyecto de desarrollo La selección básica exige reconocer propósito, libertades y condiciones de uso. La siguiente lectura amplía las obligaciones al

distribuir proyectos de desarrollo; no necesitas resolver un análisis jurídico de dependencias para completar las actividades de esta unidad.

Cuando se incorpora una biblioteca de terceros, no basta con comprobar que es gratuita o que su código está disponible. Hay que identificar su licencia, conservar los avisos que exige y revisar qué ocurrirá si el proyecto se distribuye a otras personas o empresas.

Licencia	Tipo y obligación práctica al redistribuir	Qué debe revisar un equipo de desarrollo	Ejemplo real
GPLv3	Copyleft. Al distribuir una obra derivada, deben mantenerse las condiciones de la GPLv3 y ponerse a disposición el código fuente correspondiente.	Si la dependencia se integra de un modo que forme una obra derivada y el producto se va a distribuir, hay que comprobar la compatibilidad con la licencia del proyecto.	El kernel Linux ilustra el copyleft, pero usa **GPLv2-only**, no GPLv3. No debe citarse como ejemplo de GPLv3.
MIT	Permisiva. Permite usar, modificar, sublicenciar y redistribuir, incluso en software propietario, conservando el aviso de copyright y la licencia.	Incluir los avisos de licencia de las dependencias en el repositorio o en la documentación de distribución.	React se distribuye bajo licencia MIT.
Apache 2.0	Permisiva. Al redistribuir exige entregar la licencia, señalar los archivos modificados y conservar los avisos aplicables; si la obra incluye `NOTICE`, hay que reproducir sus atribuciones pertinentes en alguna de las formas permitidas. Incluye una concesión expresa de patentes de quienes contribuyen.	Revisar las condiciones del apartado 4 y los avisos de la distribución concreta.	Kubernetes usa Apache 2.0. Android Open Source Project (AOSP) usa Apache 2.0 para la mayor parte de su software de espacio de usuario; el kernel incluido sigue GPLv2.

En Apache 2.0, apartado 4, la copia legible de las atribuciones de `NOTICE` puede acompañar a la distribución, figurar en su documentación o mostrarse donde aparezcan normalmente avisos de

terceros. No se exige conservar siempre un archivo con ese nombre ni presentar `NOTICE` como si cambiara la licencia.

La diferencia más visible es que MIT y Apache 2.0 permiten normalmente integrar componentes en una aplicación con otra licencia, siempre que se respeten sus avisos y condiciones. La GPLv3 busca que las libertades se mantengan cuando se distribuye una obra derivada. El resultado exacto depende de cómo se integre el componente y de si se distribuye el producto; por eso no se deben extraer conclusiones legales solo a partir del nombre de una licencia.

Caso de desarrollo: añadir una dependencia Un equipo crea una aplicación para gestionar reservas y quiere incorporar una biblioteca externa. Antes de añadirla al proyecto, debería seguir este proceso:

1. Localizar el archivo `LICENSE` y, cuando exista, el archivo `NOTICE` del repositorio de la dependencia.
2. Anotar el nombre, versión, licencia, origen y uso de la dependencia en un inventario del proyecto.
3. Comprobar si el producto se distribuirá como aplicación, biblioteca, imagen de contenedor o servicio instalado en equipos de clientes.
4. Mantener los avisos requeridos y consultar la política de licencias de la organización antes de combinar componentes con requisitos distintos.



Esta comparación sirve para reconocer obligaciones básicas de las licencias más habituales. No sustituye la lectura de la licencia concreta, la documentación del proyecto ni el asesoramiento jurídico cuando una decisión de distribución tenga impacto profesional.

El propósito y la licencia son dos clasificaciones distintas. Una base de datos puede ser software de aplicación y, a la vez, distribuirse bajo una licencia libre o propietaria.

2.4. Comparación de sistemas operativos según ámbito de uso

Comparar sistemas operativos no consiste en declarar uno ganador. Consiste en relacionar requisitos, aplicaciones necesarias, experiencia del equipo técnico, coste, soporte y restricciones legales con un caso de uso.

2.4.1. Criterios de comparación

Para justificar una elección, conviene valorar:

- Hardware disponible y consumo de recursos.
- Aplicaciones imprescindibles y compatibilidad de formatos.
- Tipo de usuario: oficina, desarrollo, administración, diseño o atención al público.
- Necesidades de red, seguridad, cuentas y gestión centralizada.
- Licencia, coste de adquisición, soporte y mantenimiento.

- Formación previa del equipo y documentación disponible.
- Estabilidad, actualizaciones y ciclo de vida.

2.4.2. Casos de uso profesionales

Entorno	Necesidades principales	Elección razonada posible
Aula de programación	IDE, compiladores, terminal, herramientas de red y facilidad de recuperación.	Distribución GNU/Linux en equipos o máquinas virtuales, complementada si se requiere software específico.
Oficina administrativa	Suite ofimática, correo, navegador, impresoras, formatos compartidos y soporte cercano.	Sistema de escritorio compatible con las aplicaciones y periféricos de la organización.
Servidor web	Estabilidad, red, actualizaciones controladas, administración remota, logs y servicios web.	Sistema orientado a servidor, habitualmente GNU/Linux o Windows Server según la pila tecnológica y la experiencia disponible.
Estación de diseño	Rendimiento gráfico, almacenamiento rápido, aplicaciones creativas y controladores certificados.	Sistema compatible con la suite de diseño elegida y el hardware gráfico.
Dispositivo industrial	Arranque rápido, función limitada, consumo reducido y respuesta predecible.	Sistema embebido o de tiempo real, según los plazos exigidos.
Desarrollo móvil	Kit de desarrollo de software (SDK), emuladores, control de versiones y compilación.	Sistema compatible con el SDK y las herramientas oficiales del ecosistema objetivo.

Caso guiado: pequeña empresa de desarrollo web Una empresa tiene diez puestos para desarrollar aplicaciones web, un servidor que publicará una intranet y personal administrativo que usa documentos compartidos.

Una propuesta razonada puede combinar:

- Equipos de desarrollo con un SO que ejecute cómodamente editor, terminal, navegador, contenedores o máquinas virtuales y herramientas de control de versiones.
- Un servidor con un SO estable, administración remota, actualizaciones controladas y soporte para el servidor web y la base de datos seleccionados.

- Puestos administrativos con aplicaciones ofimáticas compatibles con los formatos y flujos de la empresa.

La elección final no depende solo de una marca o de la preferencia personal. Debe justificar aplicaciones, requisitos, licencias, personal disponible y mantenimiento.



> Una justificación completa nombra el entorno, dos o más requisitos, el sistema elegido, las aplicaciones necesarias y una razón relacionada con licencia o soporte.

2.5. Actividad 02.1. Propósito y licencia

Actividad de clase, sin nota · RA7.a.

Relaciona las afirmaciones con los conceptos freeware, copyleft, EULA y licencia permisiva. Explica por qué precio y libertad son ejes distintos.

Afirmación	Concepto	Justificación
Se permite el uso sin pagar, sin conceder derecho de modificar el código.		
Redistribuir una obra derivada exige mantener determinadas libertades.		
El contrato define las condiciones de uso para la persona usuaria final.		
Se permite reutilización amplia, conservando avisos y condiciones.		

Clasifica además un IDE y una hoja de cálculo por propósito. ¿Podría una empresa usar software libre y contratar soporte? Entrega la tabla y una explicación breve; no debes deducir obligaciones de una licencia concreta solo por su precio.

2.6. Actividad 02.2. Selección para dos entornos AULES

Práctica individual calificable · RA principal: RA7.a y RA7.b · CE relacionado: RA2.c.

Esta es una única entrega: la tabla y los dos párrafos que se indican al final. La selección de aplicaciones y sus licencias condiciona el SO y su compatibilidad; por eso se integran estos tres CE. Se trabaja en el

aula y recibe una única nota para el 30 % de actividades/prácticas de sus CE, siempre que el producto los evidencie suficientemente. La fecha se comunicará en Aules.

Tu centro prepara ocho puestos para programación: IDE, navegador, terminal y máquinas virtuales. Dispone de equipos de 64 bits con 16 GiB de RAM y SSD de 512 GB; busca costes contenidos y recuperación de prácticas. Una empresa necesita además diez puestos administrativos: documentos compartidos, correo, impresora y una aplicación de gestión que su proveedor solo certifica para Windows. No se ha decidido aún qué edición ni qué licencias contratar.

Propón SO y aplicaciones para ambos entornos. En el aula prioriza GNU/Linux y justifica el uso de una VM cuando sea necesario otro sistema. En la empresa considera la compatibilidad certificada, los formatos y el soporte; no presupongas que el software gratuito cubre toda necesidad.

Para resolver esta selección sin una búsqueda abierta, usa los datos del escenario y estas fichas acotadas. **No son fichas de requisitos de instalación**: las versiones, ediciones y recursos necesarios se comprobarán en UD04.

Candidato o componente	Propósito y condición conocida	Qué queda pendiente antes de instalar
Distribución GNU/Linux	Software de base; reúne componentes con distintas licencias. El kernel Linux utiliza GPL-2.0-only con la excepción de llamadas al sistema documentada por el proyecto; no asignamos esa licencia a toda la distribución.	Distribución y versión, soporte vigente, aplicaciones disponibles y recursos para las VMs.
Windows	Software de base bajo condiciones propietarias de Microsoft; su elección debe respetar el contrato aplicable.	Edición y versión certificadas por el proveedor de gestión, licencias de los diez puestos y requisitos.
Code - OSS	Editor de código para desarrollo; el código del repositorio de Microsoft se publica bajo MIT. La distribución de marca Visual Studio Code tiene su propia licencia y debe distinguirse del proyecto de código abierto.	Elegir distribución concreta y comprobar condiciones de sus extensiones y herramientas de desarrollo.
LibreOffice	Aplicación ofimática; se distribuye bajo MPL 2.0 e incluye componentes con otras licencias.	Compatibilidad real con los documentos de la empresa, versión y soporte necesario.

Las condiciones anteriores están resumidas de las fuentes oficiales: licencias del kernel Linux, términos de Microsoft, MIT en el repositorio de Code - OSS, distinción entre Code - OSS y Visual Studio Code y licencias de LibreOffice. No necesitas consultar otros productos: puedes dejar navegador, correo e hipervisor como necesidades con selección pendiente y explicar qué comprobarías.

Entorno	Necesidades	SO propuesto y alternativa descartada	Aplicaciones y propósito	Licencia/modelo conocido	Riesgos y verificación pendiente
Aula					
Empresa					

Entrega la tabla y dos párrafos justificativos con al menos cinco criterios: recursos, compatibilidad, uso, licencia/coste y mantenimiento. En cada entorno contrasta la elección con una alternativa descartada usando al menos una restricción del escenario. Clasifica por propósito y licencia/modelo los componentes concretos que selecciones de las fichas; distingue esas condiciones conocidas de lo que siga pendiente. No inventes requisitos mínimos oficiales ni elimines la aplicación de gestión para evitar su restricción de Windows. Esta evidencia compara sistemas y analiza necesidades de software en entornos productivos; se reutilizará al planificar la instalación, sin otro proyecto independiente.

2.6.1. Antes de entregar y criterios de corrección

Revisa que la tabla recoge ambos entornos y que cada párrafo justifica la elección y la alternativa descartada. Se valorará:

- La respuesta a las necesidades y restricciones de cada entorno.
- La clasificación correcta de los componentes elegidos por propósito y licencia/modelo.
- La comparación de SO y la justificación de recursos, compatibilidad, uso, licencia/coste y mantenimiento.
- La distinción entre condiciones conocidas y comprobaciones pendientes, sin requisitos inventados.

Entrega estos productos juntos, una sola vez. 02.1 es preparación de clase, sin nota ni entrega adicional en Aules. En UD04 se reutilizará esta selección para concretar versiones y requisitos antes de instalar.

Elegir un SO prepara su instalación, pero antes estudiaremos qué ocurre cuando los programas elegidos se ejecutan y compiten por recursos.

Abrir las diapositivas de apoyo: Procesos, hilos y estados

Un sistema multitarea necesita decidir qué programa usa la CPU y cuándo. Para hacerlo no administra “aplicaciones” de forma abstracta, sino **procesos** concretos.

3.1. Qué es un proceso

Un **programa** es un conjunto de instrucciones almacenado, por ejemplo un ejecutable en el disco. Un **proceso** es ese programa cuando está siendo ejecutado, junto con su contexto: memoria asignada, registros, estado, archivos abiertos y recursos asociados.

Abrir dos veces el mismo editor puede crear dos procesos diferentes. Comparten el mismo programa de origen, pero cada uno mantiene su propio estado.

3.2. Procesos e hilos

Un proceso puede contener uno o varios **hilos**. Los hilos de un mismo proceso comparten recursos como la memoria del proceso, pero cada hilo tiene su propia secuencia de ejecución.

Un navegador puede usar hilos para atender la interfaz, reproducir contenido y realizar operaciones de red. El objetivo es mejorar la respuesta o aprovechar varios núcleos, aunque compartir datos obliga a coordinarse correctamente.

Elemento	¿Tiene un espacio de direcciones independiente?	Recursos compartidos	Uso típico
Proceso	Sí, normalmente.	Puede compartir recursos autorizados con otros procesos.	Ejecutar aplicaciones aisladas.
Hilo	No; utiliza el del proceso, aunque mantiene pila y contexto propios.	Comparte memoria y otros recursos con los hilos del mismo proceso.	Dividir tareas de una aplicación.

3.3. Estados de un proceso

Para estudiar este modelo supondremos procesos con un único hilo. En un proceso multihilo, cada hilo puede estar listo, ejecutándose o bloqueado de forma independiente: bloquear uno no bloquea necesariamente los demás.

Con esta simplificación, distinguimos cinco estados:

- **Nuevo:** se está creando.
- **Listo:** puede usar la CPU, pero espera turno.
- **En ejecución:** está utilizando una CPU.
- **Bloqueado:** espera un evento, normalmente una operación de entrada/salida.
- **Terminado:** ha finalizado o ha sido cancelado.

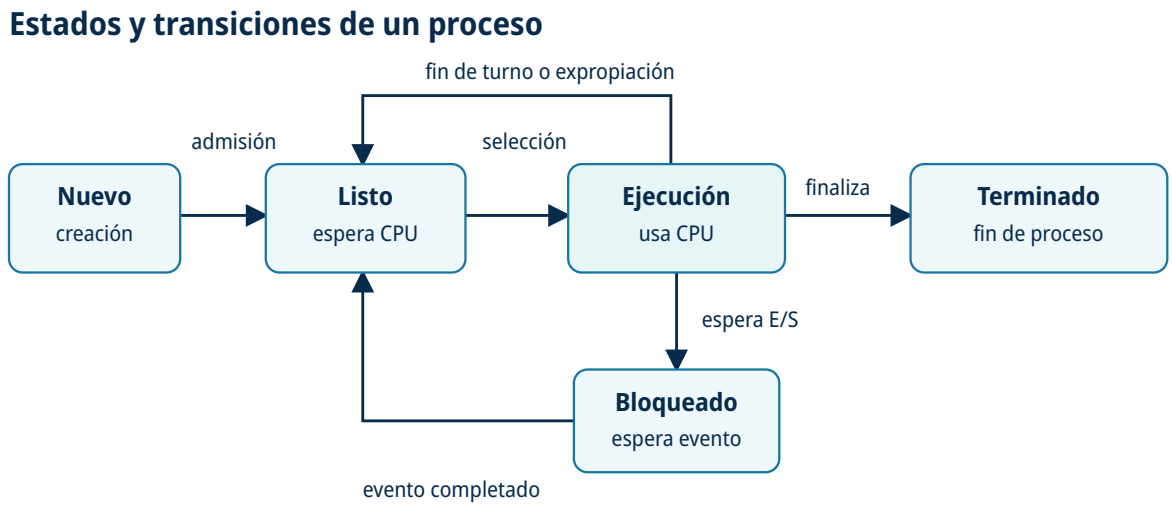


Figura 3: El proceso pasa de nuevo a listo y recibe CPU al ser seleccionado; la espera de E/S lo bloquea y el evento completado lo devuelve a listo.

Un proceso bloqueado no está esperando CPU: está esperando que ocurra algo externo, como que termine una lectura de disco. Si el sistema ejecutara solo ese proceso, la CPU quedaría desaprovechada; por eso el planificador lo sustituye temporalmente por otro listo.

3.4. Colas, bloqueo y cambio de contexto

Los procesos listos se sitúan en una **cola de preparados**. El planificador elige uno para la CPU. Cuando el proceso agota su turno, se bloquea o termina, el sistema guarda la información necesaria para retomar su ejecución más tarde y carga la del siguiente: es el **cambio de contexto**.

Podemos seguir el mecanismo con una lectura de teclado que todavía no dispone de entrada. Suponemos una lectura bloqueante y otro proceso listo:

1. El proceso lector solicita el dato y no puede continuar. El sistema conserva el contexto necesario para reanudarlo, como su contador de programa y registros.
2. El sistema lo sitúa en estado bloqueado, a la espera de entrada. Ya no es candidato a recibir CPU.
3. El planificador selecciona otro proceso de la cola de preparados.
4. El sistema recupera el contexto del seleccionado y este continúa desde donde había quedado.

Cuando llega una pulsación, el lector pasa a listo. Para volver a ejecutar, tendrá que ser seleccionado y recuperar su contexto. El mismo cambio de contexto puede deberse a otra causa: si vence un turno y queda trabajo pendiente, el proceso saliente vuelve a listo, en vez de quedar bloqueado. Si termina, se liberan sus recursos y no habrá que reanudarlo.

Un cambio de contexto tiene coste. Durante ese tiempo la CPU no avanza trabajo útil de la aplicación, sino que el sistema prepara el cambio. Por ello, los turnos demasiado cortos pueden perjudicar el rendimiento global.

3.5. Concurrencia y paralelismo

Concurrencia significa que varias tareas progresan durante un mismo período; pueden alternarse en un solo núcleo. **Paralelismo** significa que se ejecutan al mismo tiempo en distintos recursos de ejecución. Añadir núcleos no exige inventar estados nuevos en el modelo: permite que más de un hilo esté en ejecución simultáneamente.

Cada hilo mantiene su contador de programa, registros y pila. Comparte con los demás hilos del proceso el espacio de direcciones y otros recursos. Compartir facilita cooperar, pero también obliga a coordinar el acceso a datos; estudiar sincronización avanzada pertenece a otras materias.

3.6. Actividad 03.1. Un proceso espera datos

Actividad de clase, sin nota · RA2.b.

P1 está ejecutándose y solicita una lectura de disco. Mientras la lectura se completa, P2 obtiene CPU. El dispositivo termina la lectura; P1 queda disponible, pero P2 continúa hasta agotar su turno. Después el planificador selecciona P1.

Momento	Estado de P1	Estado de P2	Motivo
P1 ejecuta; P2 espera CPU			
P1 solicita lectura; P2 recibe CPU			
Termina la lectura; P2 sigue ejecutando			
P2 agota turno; se selecciona P1			

Entrega la tabla, dibuja la secuencia de estados de P1 y explica por qué acabar la E/S no garantiza recibir CPU inmediatamente. Usa flechas dirigidas con la causa de cada transición. Se comprobará que distingues listo de bloqueado.

3.7. Actividad 03.2. Programa, proceso e hilo

Actividad de clase, sin nota · RA2.b.

Reutilizarás este producto en la práctica integrada de funcionamiento del SO, apartado 6.8; no tiene una nota ni una entrega separadas.

Un mismo editor se abre en dos instancias independientes. Una instancia utiliza un hilo para la interfaz y otro para una tarea de cálculo. Explica cuántos procesos describe el escenario, qué comparten los dos hilos y qué mantienen por separado. Razona si estos hilos pueden progresar en un solo núcleo y qué cambia si hay dos. Entrega un esquema y un párrafo; no necesitas administrar procesos ni usar comandos.

Los estados permiten saber quién puede usar CPU. En el siguiente bloque aplicaremos reglas para decidir a quién dar el turno.

Abrir las diapositivas de apoyo: Planificación de CPU

El bloque anterior ha distinguido los procesos listos de los bloqueados. El planificador solo puede asignar CPU a quienes están preparados. Resolveremos modelos simplificados para entender las decisiones, sin suponer que un SO actual use exclusivamente uno de estos algoritmos para todas sus tareas.

4.1. Planificador a corto plazo

El **planificador a corto plazo** decide cuál de los procesos listos obtiene la CPU a continuación. Los algoritmos que veremos usan una simplificación útil para aprender:

- Una única unidad de ejecución: solo un proceso ejecuta en cada instante del modelo.
- Todos los tiempos están expresados en unidades enteras.
- No hay coste de cambio de contexto, salvo que se indique.
- La llegada de un proceso se anota como `llegada` y el tiempo total de CPU que necesita como `ráfaga`.

Para cada proceso calcularemos estas métricas:

Métrica	Cálculo	Significado
Tiempo de finalización	Instante en que termina.	Cuándo acaba el proceso.
Tiempo de retorno	`finalización - llegada`	Tiempo total dentro del sistema.
Tiempo de espera	`retorno - ráfaga`	Tiempo que pasó listo sin usar CPU.
Tiempo de respuesta	Primer uso de CPU `- llegada`	Espera hasta recibir CPU por primera vez.



En ejercicios sin bloqueos de E/S, el tiempo de espera es exactamente el tiempo que el proceso permanece en la cola de preparados.

4.2. Algoritmo FCFS/FIFO

FCFS (*First Come, First Served*), también llamado **FIFO** (*First In, First Out*), ejecuta los procesos por orden de llegada. Una vez que un proceso empieza, no se interrumpe hasta terminar su ráfaga: es un algoritmo **no expropiativo**.

Ejemplo resuelto 1: FCFS

Proceso	Llegada	Ráfaga de CPU
P1	0	5
P2	1	3
P3	2	1
P4	3	2

P1 llega primero y ejecuta de `0` a 5. Mientras tanto llegan P2, P3 y P4, que esperan por orden de llegada. Después se ejecutan P2, P3 y P4.

```
0       5       8   9     11
|--- P1 ---|-- P2 --|-P3-|--P4--|
```

Proceso	Finalización	Retorno	Espera
P1	5	5 - 0 = 5	5 - 5 = 0
P2	8	8 - 1 = 7	7 - 3 = 4
P3	9	9 - 2 = 7	7 - 1 = 6
P4	11	11 - 3 = 8	8 - 2 = 6
Media		**6,75**	**4,00**

FCFS es sencillo y previsible, pero puede provocar el **efecto convoy**: un proceso largo al principio obliga a esperar a muchos procesos cortos que llegan detrás.

4.3. Algoritmo SJF/SRTF

SJF (*Shortest Job First*) selecciona, entre los procesos listos, el que tiene la menor ráfaga de CPU y lo deja terminar: aquí estudiaremos su versión no expropiativa. Su versión expropiativa es **SRTF** (*Shortest Remaining Time First*): si llega un proceso cuya ráfaga restante es menor que la del proceso actual, lo sustituye.

SJF necesita conocer o estimar la duración de la ráfaga. En la práctica no se conoce con certeza, pero el sistema puede estimarla a partir del comportamiento anterior.

Ejemplo resuelto: SJF no expropiativo

Usamos P1 (llegada 0, ráfaga 5), P2 (1, 3), P3 (2, 1) y P4 (3, 2). En el instante 0 solo está P1; SJF lo deja terminar en 5 aunque lleguen procesos cortos. Entonces elige P3, después P4 y por último P2.

```
0–5 P1 | 5–6 P3 | 6–8 P4 | 8–11 P2
```

Proceso	Finalización	Retorno	Espera
P1	5	5	0
P2	11	10	7
P3	6	4	3
P4	8	5	3
Media		6,00	3,25

Este ejemplo muestra la diferencia con SRTF: llegar con una ráfaga corta no interrumpe al proceso que ya ejecuta en SJF no expropiativo.

Ampliación opcional. Ejemplo resuelto 2: SRTF

Este ejemplo desarrolla el contraste con SJF; no añade un quinto algoritmo obligatorio a la actividad 04.1. Usamos los mismos procesos del ejemplo anterior.

Proceso	Llegada	Ráfaga
P1	0	5
P2	1	3
P3	2	1
P4	3	2

1. En `t=0`, solo existe P1: ejecuta durante una unidad. Le quedan 4.
2. En `t=1`, llega P2 con ráfaga 3. Como `3 < 4`, P2 sustituye a P1.
3. En `t=2`, llega P3 con ráfaga 1. Como `1 < 2` que le queda a P2, P3 ejecuta y termina.
4. En `t=3`, llega P4. P2 y P4 tienen ambos 2 unidades restantes; mantenemos el que ya estaba listo antes, P2.
5. Al terminar P2, se ejecuta P4 y finalmente P1.

```
0   1     2   3     5     7       11
|-P1-|--P2--|-P3-|--P2--|--P4--|----P1----|
```

Proceso	Finalización	Retorno	Espera
P1	11	11 - 0 = 11	11 - 5 = 6
P2	5	5 - 1 = 4	4 - 3 = 1
P3	3	3 - 2 = 1	1 - 1 = 0
P4	7	7 - 3 = 4	4 - 2 = 2
Media		**5,00**	**2,25**

SRTF mejora la espera media en este caso porque termina antes los trabajos cortos. Sin embargo, si siguen llegando procesos breves, uno largo puede retrasarse mucho: este riesgo se denomina **inanición**.



No hay que reordenar los procesos en SRTF solo al terminar uno. Hay que revisar la decisión cada vez que llega un proceso nuevo, porque puede ser necesario desalojar al que estaba ejecutándose.

4.4. Algoritmo Round Robin

Round Robin reparte la CPU en turnos de duración máxima fija, llamada **quantum**. Cuando el quantum termina, el proceso vuelve al final de la cola de preparados si todavía necesita CPU. Es un algoritmo expropiativo y adecuado para sistemas interactivos, porque evita que un proceso largo monopolice la CPU.

Ejemplo resuelto 3: Round Robin con quantum 2

Para aislar el funcionamiento de la cola, suponemos que todos los procesos llegan en `t=0`.

Proceso	Ráfaga
P1	5
P2	3
P3	1

1. P1 usa dos unidades y le quedan 3: vuelve al final de la cola.
2. P2 usa dos unidades y le queda 1: vuelve al final.

3. P3 necesita solo una unidad y termina.
4. P1 usa otras dos unidades y le queda 1.
5. P2 termina en su siguiente turno y P1 termina al final.

```
0     2     4   5     7   8   9
|--P1--|--P2--|-P3-|--P1--|-P2|-P1|
```

Proceso	Finalización	Retorno	Espera
P1	9	9	9 - 5 = 4
P2	8	8	8 - 3 = 5
P3	5	5	5 - 1 = 4
Media		**7,33**	**4,33**

Round Robin ofrece una respuesta inicial equilibrada: todos los procesos reciben CPU pronto. Su resultado depende mucho del quantum:

- Con un quantum muy grande, se comporta de forma parecida a FCFS.
- Con un quantum pequeño, los procesos pueden recibir antes su primera atención, pero pueden aumentar los cambios de contexto. El efecto depende de la carga y del tamaño de las ráfagas.

4.5. Diagramas de Gantt y métricas básicas

Un diagrama de Gantt representa qué proceso usa la CPU en cada intervalo. En los esquemas de texto, la anchura gráfica de los tramos es orientativa: los límites numéricos determinan la duración exacta. Para resolver cualquier ejercicio de planificación conviene seguir siempre el mismo método:

1. Ordena los procesos por instante de llegada y crea una tabla con llegadas y ráfagas.
2. Sitúate en el instante inicial y anota qué procesos están listos.
3. Aplica estrictamente la regla del algoritmo.
4. Añade al diagrama cada intervalo de ejecución y actualiza las ráfagas restantes.
5. Cuando todos terminen, calcula finalización, retorno y espera.
6. Comprueba que la suma de las ráfagas coincide con el tiempo total de CPU ocupado, si no hay periodos inactivos.

Ejemplo parcialmente resuelto: ¿qué algoritmo favorece más a P3?

Proceso	Llegada	Ráfaga
P1	0	8
P2	1	4
P3	2	1

En FCFS, P3 solo puede empezar cuando terminen P1 y P2: espera `8 + 4 - 2 = 10` unidades. En SRTF, P3 puede desalojar al proceso actual al llegar porque su ráfaga es la menor: su espera es `0`.

La comparación muestra una idea importante: una métrica media puede mejorar mientras algún proceso concreto empeora. Por eso la elección de algoritmo depende del objetivo del sistema.

En este ejemplo visual todos llegan en 0: A necesita 5 unidades, B 2 y C 1. Se omite coste de contexto; el orden inicial es A, B, C. El Gantt compara FCFS, SJF y RR con quantum 2.

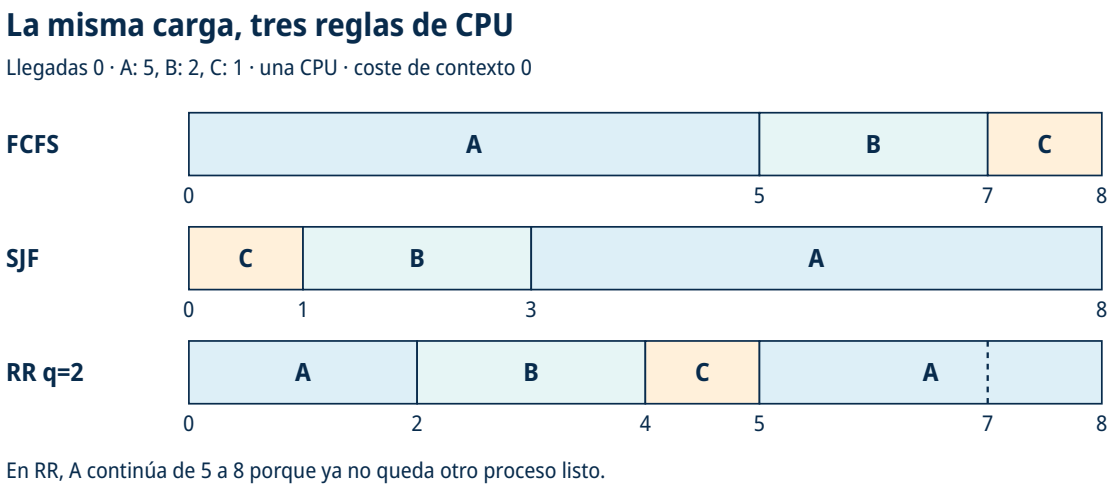


Figura 4: Gantt proporcional de una carga común: FCFS A-B-C; SJF C-B-A; RR A-B-C-A, con final total en 8.

4.6. Comparación de algoritmos de planificación

Algoritmo	Tipo	Ventaja principal	Limitación principal	Contexto apropiado
FCFS/FIFO	No expropiativo	Sencillo y previsible.	Efecto convoy; respuesta pobre ante trabajos largos.	Procesos por lotes simples.

Algoritmo	Tipo	Ventaja principal	Limitación principal	Contexto apropiado
SJF/SRTF	SJF no expropiativo; SRTF expropiativo	Reduce la espera media cuando se estiman bien las ráfagas.	Requiere estimaciones; puede causar inanición.	Lotes con trabajos de duración previsible.
Prioridades	Según la variante; aquí no expropiativo	Atiende trabajos con mayor prioridad.	Puede causar inanición sin medidas como envejecimiento.	Cargas con importancia o urgencia diferenciadas; véase 4.7.
Round Robin	Expropiativo	Reparte respuesta entre procesos y evita monopolios.	Depende del quantum y puede generar más cambios de contexto.	Sistemas interactivos y uso compartido.

No existe un algoritmo universalmente mejor. Un servidor de cálculo por lotes puede priorizar el rendimiento medio; un escritorio interactivo debe evitar que una aplicación larga deje sin respuesta a las demás; un sistema de tiempo real necesita además cumplir plazos definidos.



Antes de entregar un ejercicio, verifica que cada proceso ha recibido exactamente tantas unidades de CPU como su ráfaga y que sus tiempos de espera nunca son negativos.

4.7. Planificación por prioridades

Cada proceso recibe un nivel de prioridad. El planificador selecciona el más prioritario de la cola de listos. Debe indicarse si un número pequeño o grande significa mayor prioridad y si el algoritmo permite expropiación. En el modelo **no expropiativo**, el proceso elegido conserva CPU hasta terminar su ráfaga; en el expropiativo, una llegada de mayor prioridad puede interrumpirlo.

Una prioridad fija puede retrasar indefinidamente trabajos poco prioritarios. El **envejecimiento** aumenta progresivamente la prioridad de quienes llevan mucho tiempo esperando para reducir ese riesgo. Prioridad no equivale a duración: un trabajo largo puede ser urgente.

Ejemplo resuelto: prioridades no expropiativas

Todos llegan en 0. A necesita 4 unidades y tiene prioridad 2; B necesita 2 y tiene prioridad 1; C necesita 3 y tiene prioridad 3. Un número menor indica prioridad mayor.

```
0-2 B | 2-6 A | 6-9 C
```

B no se elige por ser corto, sino por prioridad. Las esperas son A=2, B=0 y C=6; la espera media es $8/3 \approx 2,67$. Los retornos son A=6, B=2 y C=9; media $17/3 \approx 5,67$.

4.8. Actividad 04.1. Una misma carga, distintas decisiones

Actividad de clase, sin nota · RA2.b.

Reutilizarás este producto en la práctica integrada de funcionamiento del SO, apartado 6.8; no tiene una nota ni una entrega separadas. En 04.1, solo se incorporan FCFS y RR.

Resuelve esta carga con FCFS, SJF no expropiativo, Round Robin con quantum 2 y prioridades no expropiativas. Como ampliación opcional, compara también SRTF con los cuatro algoritmos del recorrido principal.

Proceso	Llegada	Ráfaga	Prioridad
A	0	6	3
B	1	2	1
C	2	4	2
D	4	1	1

Reglas comunes: una única unidad de ejecución, sin E/S ni coste de contexto; solo un proceso ejecuta en cada instante. Desempata por llegada y después orden A, B, C, D. En prioridades, menor número significa mayor prioridad. En SRTF conserva al que ejecuta si su tiempo restante empata con una nueva llegada. En RR, las llegadas que coinciden con el fin de quantum se incorporan a la cola antes de reencolar el proceso que agota su turno; las que llegaron durante el turno ya están esperando.

Para cada uno de los cuatro algoritmos principales dibuja Gantt y completa una copia de esta tabla:

Proceso	Primer inicio	Finalización	Retorno	Espera	Respuesta
A					
B					
C					
D					
Media					

Comprueba que se ejecutan exactamente 13 unidades de CPU y que ninguna espera es negativa. Explica qué decisión favorece a D y por qué una mejora de la media no garantiza mejorar cada proceso. Entrega los diagramas, tablas y comparación razonada.



Puedes contrastar tus cálculos con el simulador de planificación. Para reproducir 04.1, identifica P1, P2, P3 y P4 con A, B, C y D, respectivamente, y conserva sus llegadas y ráfagas. En RR selecciona la regla «el que llega» para incorporar las llegadas antes de reencolar al proceso que agota el turno. El recurso ofrece FCFS, SJF, SRTF y RR; no incluye prioridades. Úsalo como comprobación opcional y justifica también las colas, intervalos y métricas por tu cuenta. SRTF sigue siendo ampliación.

4.9. Actividad 04.2. El tamaño del turno

Actividad de clase, sin nota · RA2.b.

Todos llegan en 0 y se encolan A, B, C. Sus ráfagas son A=5, B=4 y C=2. Resuelve RR con quantum 2 y quantum 4, sin coste de contexto ni E/S. Compara primer inicio, finalización y respuesta de cada proceso. Cuenta cambios de proceso entre intervalos consecutivos, sin contar el arranque inicial ni la salida final.

Completa esta tabla y entrega ambos Gantt:

Proceso	Primer inicio q2	Finalización q2	Respuesta q2	Primer inicio q4	Finalización q4	Respuesta q4
A						
B						
C						

Añade el número de cambios entre procesos para cada quantum. Explica qué quantum reparte antes la primera atención y por qué turnos más cortos pueden aumentar el coste real aunque nuestro cálculo lo omita.

4.10. Actividad 04.3. Elegir una política

Actividad de clase, sin nota · RA2.b.

Un equipo comparte CPU entre tareas de cálculo largas y una interfaz que debe responder a pulsaciones. Otro equipo procesa por lotes trabajos cuyas ráfagas se estiman razonablemente. Propón una política para cada escenario entre FCFS, SJF, RR y prioridades, explica una ventaja y un riesgo, y describe cómo el envejecimiento ayuda cuando se usan prioridades. Se aceptan alternativas justificadas; no se pide identificar el planificador real de un producto.

La CPU no puede trabajar con cualquier dato directamente desde el disco. El siguiente bloque estudia cómo el SO organiza el espacio de memoria de cada proceso.

Abrir las diapositivas de apoyo: Gestión de memoria y paginación

En UD01 distinguimos registros, caché, RAM y almacenamiento. Ahora cambiamos de pregunta: ¿cómo usa el SO esa memoria para que varios procesos trabajen sin invadir los datos de los demás? El planificador reparte CPU, pero cada proceso necesita también un espacio donde mantener instrucciones y datos.

5.1. Asignar, proteger y liberar memoria

La **RAM** guarda temporalmente instrucciones y datos que están en uso. El SO asigna memoria a los procesos, controla los accesos y recupera recursos cuando dejan de necesitarse. El problema no se reduce a tener capacidad: hay que proteger el núcleo y los procesos, permitir compartir zonas cuando se autorice y traducir las direcciones usadas por cada programa.

Un editor y un navegador pueden usar internamente la misma dirección virtual sin referirse al mismo dato físico. El aislamiento permite que un error de uno no modifique sin permiso la memoria del otro. Tampoco es exclusivo de multitarea: incluso un sistema con una sola aplicación necesita organizar memoria y controlar sus recursos.

5.2. Direcciones virtuales y físicas

Una **dirección virtual** pertenece al espacio de direcciones de un proceso. Una **dirección física** identifica una posición de RAM. La **memoria virtual** proporciona a los procesos espacios de direcciones y mecanismos de protección/traducción; no es simplemente “RAM más disco”. Puede funcionar aunque no se haya configurado un área de intercambio.

El SO configura las estructuras de traducción y el hardware, mediante la **unidad de gestión de memoria** o MMU, las utiliza durante los accesos. Un proceso no decide libremente qué marco físico emplea. Algunas direcciones pueden estar autorizadas para leer pero no escribir; otras pueden no estar asignadas. Un acceso inválido no equivale a un fallo de página recuperable.



Memoria virtual, RAM física y área de intercambio son conceptos distintos. El espacio virtual describe direcciones; la RAM mantiene datos residentes; el intercambio puede conservar temporalmente determinados datos fuera de RAM.

5.3. Páginas, marcos y tabla de páginas

La **paginación** divide el espacio virtual en unidades de tamaño fijo llamadas **páginas** y la RAM en unidades del mismo tamaño llamadas **marcos**. La tabla de páginas relaciona cada página virtual con el marco que la aloja e incluye información como presencia y permisos.

Esto permite que páginas consecutivas de un proceso ocupen marcos físicos separados. No hace falta encontrar un único hueco continuo para todo su espacio. Puede quedar espacio sin aprovechar dentro de la última página asignada: **fragmentación interna**. No confundas una página de memoria con un bloque del sistema de archivos.

Usaremos páginas de **4 KiB = 4096 bytes** en nuestros ejemplos. Es un dato del ejercicio, no un tamaño único obligatorio de todos los equipos. Una dirección virtual se divide en número de página y desplazamiento dentro de esa página.

```
página = cociente entero(dirección virtual / tamaño de página)
desplazamiento = resto(dirección virtual / tamaño de página)
dirección física = marco × tamaño de página + desplazamiento
```

Ejemplo resuelto: localizar un byte

Un proceso accede a la dirección virtual 8205, con páginas de 4096 bytes. La página 2 está presente en el marco 7.

1. `8205 = 2 × 4096 + 13`: página virtual 2, desplazamiento 13.
2. La tabla indica marco 7.
3. La dirección física es `7 × 4096 + 13 = 28685`.

El desplazamiento se conserva: cambia el lugar donde empieza la página, no la posición del byte dentro de ella. Sumar número de página y desplazamiento sería incorrecto.

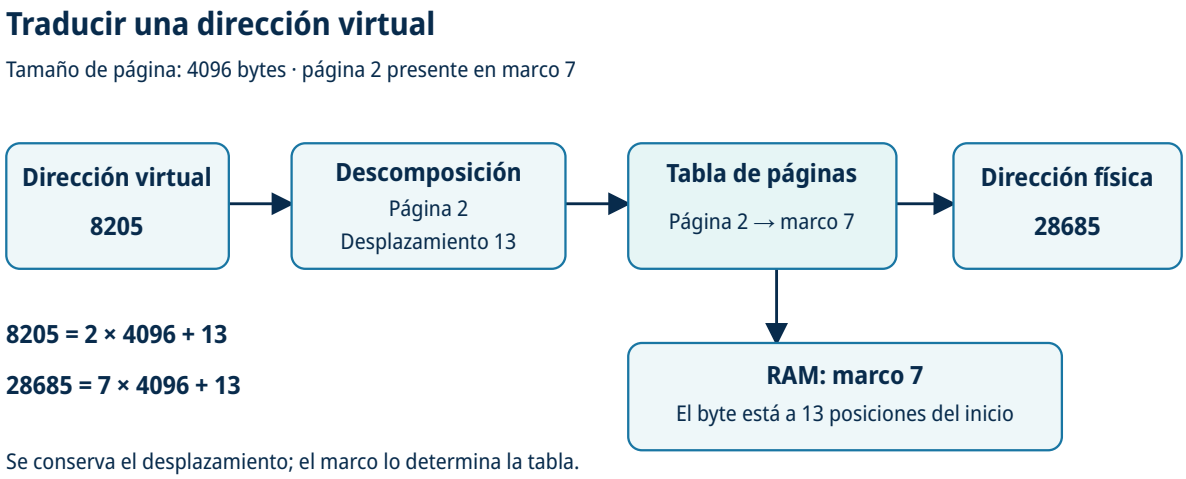


Figura 5: La dirección virtual 8205 se divide en página 2 y desplazamiento 13; la tabla indica marco 7 y la dirección física resulta 28685.

5.4. Cuando una página no está presente

Un acceso a una página no residente puede producir un **fallo de página**. El hardware informa al SO; este comprueba si el acceso es válido. Si lo es, prepara la página en RAM y actualiza la traducción para que el proceso pueda continuar. Si necesita leerla de almacenamiento, el proceso puede esperar mientras otro usa CPU.

No todo fallo de página implica leer swap. La página puede proceder de un archivo ejecutable, requerir una página inicializada a cero o responder a otras situaciones gestionadas por el sistema. Si no existe autorización para la dirección, el SO no inventa una página válida: puede terminar el proceso por acceso indebido.

Cuando faltan marcos libres, puede ser necesario elegir una página para retirar. Si contiene cambios que deben conservarse, habrá que escribirlos antes; si puede reconstruirse desde su archivo y no ha cambiado, puede descartarse y cargarse después. Aquí interesa entender la decisión, sin implementar algoritmos de reemplazo avanzados.

5.5. Intercambio y presión de memoria

El **área de intercambio** o *swap* puede ser una partición o un archivo destinado a alojar temporalmente ciertas páginas. En textos históricos, *swapping* también describe trasladar procesos completos; al hablar de sistemas actuales debemos precisar si se intercambian páginas.

Una analogía útil es la mesa y la estantería. La RAM es la mesa donde están los documentos en uso; el almacenamiento guarda más información, pero buscarla introduce espera. La analogía ayuda a

entender el movimiento, aunque la traducción de direcciones y los permisos requieren el modelo de páginas.

Si el sistema mueve páginas continuamente, puede dedicar gran parte del tiempo a transferencias en vez de avanzar trabajo: **hiperpaginación** o *thrashing*. Más almacenamiento de intercambio no garantiza una buena respuesta ni evita toda situación de falta de memoria.



La memoria virtual no crea RAM física. No apliques una regla universal de swap “doble de RAM”: influyen carga, memoria disponible, distribución y necesidades como hibernación. En el laboratorio usaremos una configuración elegida y justificada.

5.6. Actividad 05.1. Traducción con una tabla completa

Actividad de clase, sin nota · RA2.b.

Reutilizarás este producto en la práctica integrada de funcionamiento del SO, apartado 6.8; no tiene una nota ni una entrega separadas.

Un proceso usa páginas de 4096 bytes y esta tabla:

Página virtual	Presente	Marco físico
0	Sí	3
1	No	—
2	Sí	7
3	Sí	1

Todos los accesos del ejercicio están autorizados. Traduce 15, 8205 y 12300 a direcciones físicas. Analiza además el acceso a 4100: indica página/desplazamiento y explica por qué no puedes obtener todavía una dirección física con la tabla dada.

Dirección virtual	Página	Desplazamiento	Dirección física o situación
15			
8205			
12300			
4100			

Entrega tabla y explicación. Comprueba que todo desplazamiento está entre 0 y 4095. No asignes un marco inventado a la página ausente.

5.7. Actividad 05.2. Diagnosticar una idea equivocada

Actividad de clase, sin nota · RA2.b.

Un compañero dice: “Mi VM tiene 2 GiB de RAM y 8 GiB de swap; por tanto trabaja igual que una VM con 10 GiB de RAM”. Corrige la afirmación usando espacio virtual, RAM, intercambio y rendimiento. Explica también si un fallo de página siempre es un error del programa y si siempre exige leer disco. Entrega un párrafo de ocho a diez líneas con un ejemplo concreto.

La memoria contiene el trabajo en curso. El siguiente bloque distingue esos datos de los archivos persistentes y conecta su acceso con los dispositivos de E/S.

Abrir las diapositivas de apoyo: Archivos, directorios y entrada/salida

Un proceso recibe CPU y usa memoria, pero necesita conservar resultados y comunicarse con dispositivos. En este bloque introduciremos archivos, directorios y E/S desde las funciones del SO. Su administración mediante comandos, permisos y herramientas se desarrollará en UD05 Linux y UD06 Windows; la gestión de discos, copias y automatización se consolidará en UD07.

6.1. Datos persistentes y sistema de archivos

El **sistema de archivos** organiza datos persistentes y mantiene información para localizarlos y controlar el acceso. Un **archivo** reúne datos; un **directorio** organiza nombres y referencias a archivos o subdirectorios. El tamaño, fechas y permisos son **metadatos**: describen el objeto, no son todo su contenido.

Abrir, leer, escribir y cerrar un archivo son operaciones solicitadas por aplicaciones al SO. El sistema comprueba permisos, localiza la información y coordina el dispositivo. La aplicación trabaja con la abstracción de archivo; no necesita conocer cada sector físico.

No confundas el árbol de directorios con el disco. Un mismo árbol puede reunir datos de varios dispositivos montados. Tampoco todos los directorios necesitan una partición independiente. Ver una carpeta llamada home no demuestra que exista una partición separada para ella.

6.2. Árbol y rutas

En GNU/Linux, / es la raíz del árbol. Este ejemplo tiene una carpeta personal, un proyecto y un registro:

```
/
├── home/
│   └── alumno/
│       ├── documentos/
│       │   └── informe.txt
│       └── proyecto/
│           └── main.py
```

```
└── var/
    └── log/
        └── practica.log
```

Una **ruta absoluta** indica el recorrido desde la raíz: `/home/alumno/proyecto/main.py`. Una **ruta relativa** se interpreta desde el directorio actual. Desde `/home/alumno/documentos`, `../proyecto/main.py` llega al mismo archivo: `..` representa el padre y `.` el directorio actual. No estamos ejecutando una orden; estamos interpretando referencias sobre un árbol conocido.

En Windows es habitual encontrar rutas absolutas como `C:\Users\Alumno\Documents\informe.txt`; también existen otros tipos de rutas, como las de recursos de red. La letra de unidad y las barras requieren atención al comparar sistemas. Aquí importa entender raíz, directorio actual y recorrido, no memorizar toda la jerarquía del SO.



El shell puede expandir ~ al directorio personal en ciertos contextos. No es un directorio llamado literalmente ~ ni una regla universal de todos los programas que reciben rutas.

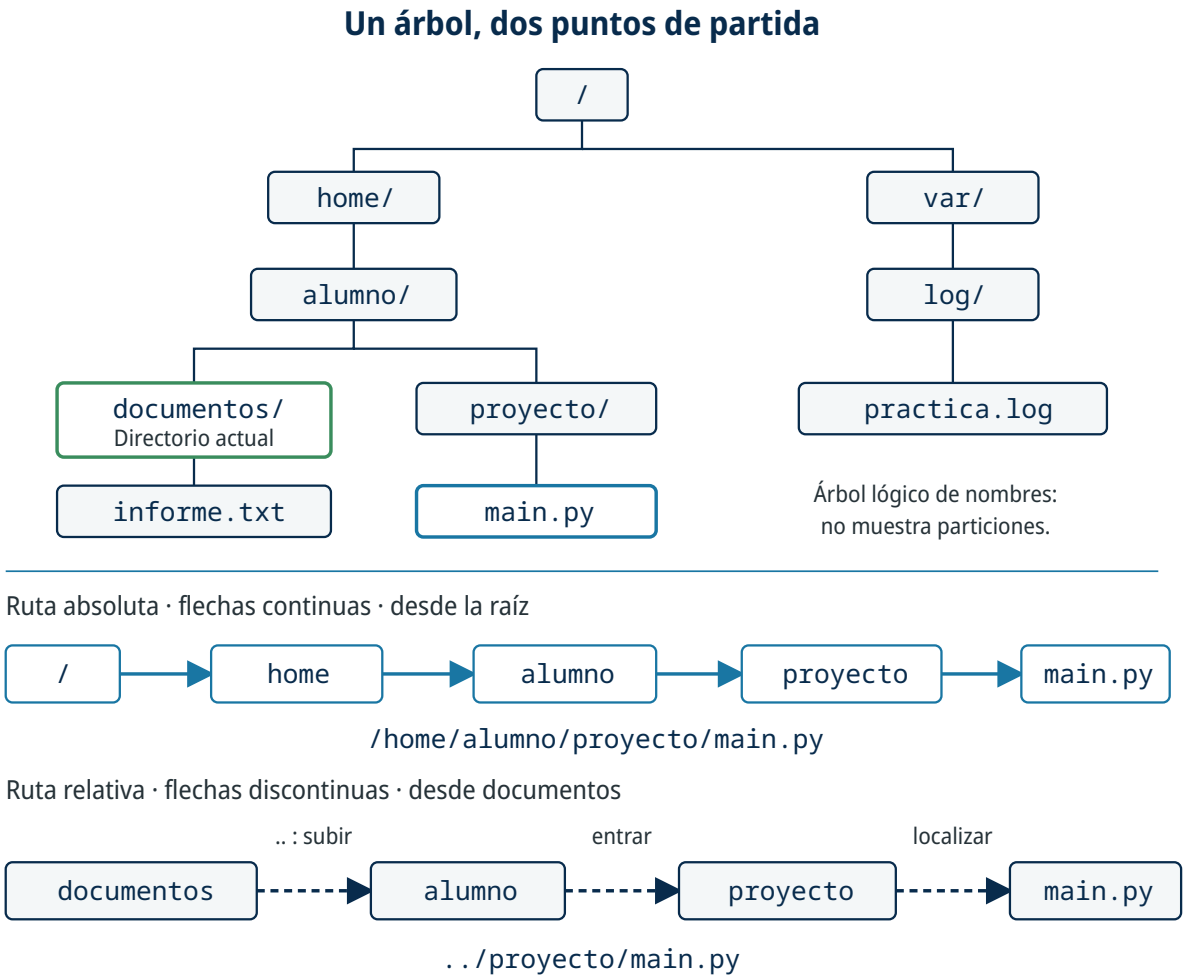


Figura 6: Desde la raíz, /home/alumno/proyecto/main.py identifica main.py; desde /home/alumno/documentos, ../proyecto/main.py sube al padre y entra en proyecto.

Figura 6.1. Dos recorridos hacia el mismo archivo. La ruta absoluta comienza en `/`; la relativa comienza en `documentos`, sube a `alumno` con `..` y entra en `proyecto`. Las ramas representan directorios y archivos, no particiones.

6.3. Sistemas de archivos, particiones y formato

Una **partición** es una división lógica de un disco; un **sistema de archivos** proporciona estructuras para organizar datos. **Formatear** crea las estructuras del sistema de archivos en el destino seleccionado y puede destruir el acceso a los datos que había. No equivale a un borrado seguro de cada byte.

El SO puede trabajar con diferentes sistemas de archivos. ext4 es habitual en Linux; NTFS, en Windows; FAT32 puede aparecer en una partición de sistema EFI. Cada uno tiene características y límites que deben

comprobarse para la configuración utilizada. La comparación técnica y la administración detallada de formatos, volúmenes y discos se estudian en UD07.

Durante la instalación será necesario elegir un destino y un diseño básico, pero todavía no haremos una práctica de administración de discos. La diferencia entre directorio, partición y sistema de archivos permitirá interpretar el instalador sin confundir las tres ideas.

6.4. Entrada/salida: del servicio al dispositivo

La **entrada/salida** (E/S) incluye operaciones con teclado, pantalla, red, discos e impresoras. El SO las coordina mediante controladores y mecanismos de hardware, retomando las interrupciones de UD01.

En una operación síncrona, si un proceso solicita datos que aún no están disponibles, puede quedar bloqueado mientras se completa la E/S. La CPU puede ejecutar otro proceso listo. Cuando llega el evento esperado, el proceso pasa a listo, pero el planificador decide cuándo vuelve a ejecutarse. No toda solicitud exige esperar un dispositivo: los datos podrían estar ya en caché.

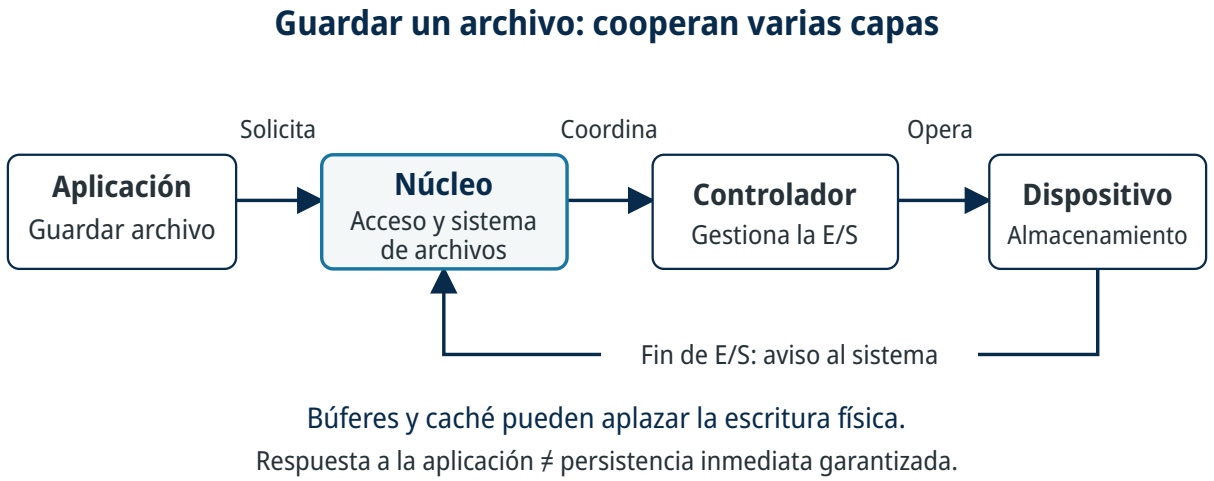


Figura 7: Aplicación, núcleo, controlador y dispositivo cooperan para guardar un archivo; búferes y caché pueden aplazar su persistencia.

Figura 6.2. Cooperación durante una escritura. Lee las flechas desde la petición de la aplicación hasta el dispositivo. El aviso de finalización vuelve al sistema; no selecciona por sí solo al siguiente proceso.

La figura representa **guardar**, mientras que el párrafo anterior describe **leer**. En ambos casos cooperan aplicación, SO, controlador y dispositivo. Lo que cambia es el sentido de los datos: al escribir van desde la memoria de trabajo hacia el destino; al leer se recuperan hacia la memoria. Además, una petición puede atenderse con datos ya disponibles en caché, sin esperar una nueva lectura física.

Por ejemplo, al enviar un documento a imprimir, la aplicación entrega el trabajo al servicio de impresión; este organiza la cola y utiliza el controlador para comunicarse con la impresora. La aplicación puede continuar antes de que salga la última página. El caso permite separar petición, transferencia y finalización física sin suponer que toda lectura o escritura siga los mismos tiempos.

6.4.1. Consulta del estado e interrupciones

En la **consulta activa**, también llamada **encuesta** o *polling*, software comprueba repetidamente el estado de una operación. Consume atención de CPU mientras consulta, aunque puede resultar adecuado en circunstancias concretas. No está limitado a sistemas monotarea ni significa que toda aplicación gestione directamente los registros del dispositivo.

Con **interrupciones**, el hardware avisa de un evento y el SO ejecuta su rutina de atención. Después puede actualizar el estado del proceso que esperaba. El dispositivo no selecciona directamente al próximo proceso: avisar del fin de E/S y planificar CPU son funciones diferentes.

6.4.2. Acceso directo a memoria

El **acceso directo a memoria** o DMA permite que un dispositivo/controlador transfiera datos hacia o desde RAM sin que la CPU copie cada unidad individualmente. El sistema prepara y supervisa la operación; su finalización puede notificarse mediante una interrupción. DMA no elimina toda intervención de CPU ni sustituye al controlador.

6.5. Búferes, caché y cola de impresión

Un **búfer** almacena temporalmente datos para coordinar productores y consumidores que trabajan a ritmos distintos. La **caché** conserva copias reutilizables para reducir accesos más lentos. Pueden intervenir en una misma operación, pero sus propósitos no son idénticos.

Al guardar, algunos datos pueden quedar pendientes de escritura en memoria antes de alcanzar almacenamiento persistente. Por eso apagar bruscamente o retirar un dispositivo sin expulsarlo puede perder cambios. No debemos prometer que toda escritura se aplaza: el comportamiento depende de operación y configuración.

Una **cola de impresión** mantiene trabajos pendientes y permite que la aplicación continúe sin controlar la impresora durante toda la impresión. Este ejemplo conecta el archivo que se envía, el proceso que lo solicita, el servicio de impresión y el dispositivo físico.

6.6. Actividad 06.1. Leer un árbol, sin administrar discos

Actividad de clase, sin nota · RA2.b.

Usa exclusivamente el árbol del apartado 6.2. El directorio actual es `/home/alumno/documentos`. Escribe las rutas absolutas y relativas hacia `informe.txt`, `main.py` y `practica.log`.

Archivo destino	Ruta absoluta	Ruta relativa desde documentos
informe.txt		
main.py		
practica.log		

Explica además si crear una carpeta exige crear una partición y distingue contenido de metadatos usando `informe.txt`. Entrega tabla y dos respuestas razonadas. No necesitas crear archivos ni ejecutar comandos.

6.7. Actividad 06.2. Una lectura que tarda

Actividad de clase, sin nota · RA2.b.

Reutilizarás este producto en la práctica integrada de funcionamiento del SO, apartado 6.8; no tiene una nota ni una entrega separadas.

Un proceso solicita una lectura síncrona de un archivo; los datos no están en caché y el dispositivo tarda en entregarlos. Hay otro proceso listo. Explica la relación causal entre estas fases: petición al SO, comprobación/localización, inicio de E/S mediante controlador, proceso bloqueado, ejecución de otro proceso, aviso de finalización y vuelta del primero a listo. Indica dónde ayuda DMA y qué cambiaría si los datos ya estuvieran en caché.

Entrega una secuencia de pasos y explica por qué “listo” no significa “ya ejecutándose”. Añade una razón para expulsar correctamente un dispositivo después de escribir datos. Se comprobará la relación entre gestión de procesos, memoria, archivos y E/S.

6.8. Práctica integrada. Funcionamiento del SO AULES

Práctica individual calificable · RA principal: RA2 · CE: RA2.b.

Reúne en un único documento los cuatro productos siguientes, elaborados progresivamente en clase. Usa los mismos escenarios, datos, tablas y reglas de los apartados enlazados; no necesitas otro mapa ni una memoria adicional. Esta es una sola entrega, con una única nota para el 30 % de actividades/prácticas de RA2.b. La fecha se comunicará en Aules.

6.8.1. Procesos e hilos

Reutiliza el esquema y el párrafo de **03.2**, en Procesos, hilos y estados, apartado 3.7. Explica los dos procesos del editor, qué comparten sus hilos y qué conservan por separado, y cómo progresan con uno o dos núcleos.

6.8.2. Decisiones de CPU

De **04.1**, en Planificación de CPU, apartado 4.8, incluye **solo FCFS y Round Robin con quantum 2**: dos Gantt, sus tablas completas y la comparación razonada entre estas dos políticas. Conserva llegadas, ráfagas y reglas de cola del enunciado. Comprueba las 13 unidades de CPU, esperas no negativas y las métricas de cada proceso; explica cómo cambia la atención a D y por qué la media no describe todos los casos.

SJF y prioridades siguen siendo trabajo de clase, junto con 04.2 y 04.3; SRTF continúa como ampliación opcional. Esta selección afecta únicamente a lo que reúnes en esta entrega. El simulador enlazado en el bloque 04 es opcional: una captura no sustituye Gantt, tablas y razonamiento.

6.8.3. Traducción de memoria

Reutiliza la tabla y la explicación de **05.1**, en Memoria y paginación, apartado 5.6. Comprueba página y desplazamiento para cada dirección, conserva este último al traducir y explica qué impide completar la dirección física de la página no residente sin inventar un marco.

6.8.4. Cooperación durante una lectura

Incluye la secuencia y las explicaciones de **06.2**, en el apartado 6.7 de este mismo bloque. Relaciona aplicación, SO, controlador, memoria y dispositivo; distingue bloqueo, vuelta a listo y ejecución. Explica la intervención de DMA, qué cambia si los datos están en caché y por qué expulsar correctamente un dispositivo tras escribir.

6.8.5. Antes de entregar y criterios de corrección

Comprueba que los cuatro apartados están identificados y que el documento contiene tus esquemas, tablas y explicaciones, no solo resultados del simulador. Se valorará:

- La distinción entre programa, proceso e hilo y la explicación de recursos compartidos y contexto propio.
- La coherencia de las decisiones de CPU con las reglas, los intervalos y las métricas calculadas.
- La descomposición y traducción de direcciones y el tratamiento de la página no residente.
- La relación causal entre funciones, componentes y estados durante la E/S, diferenciando listo de ejecución y respuesta de persistencia.

Los ejercicios de origen se practican y corrigen en clase, sin notas separadas: reutiliza sus productos aquí, sin subirlos de nuevo como entregas independientes. No se pide instalar ni configurar un sistema o ejecutar comandos.

Los fundamentos ya permiten justificar decisiones de instalación. La siguiente unidad, Virtualización e instalación de SO, comienza con la preparación de un laboratorio virtual para aplicar estos conocimientos a sistemas libres y propietarios.