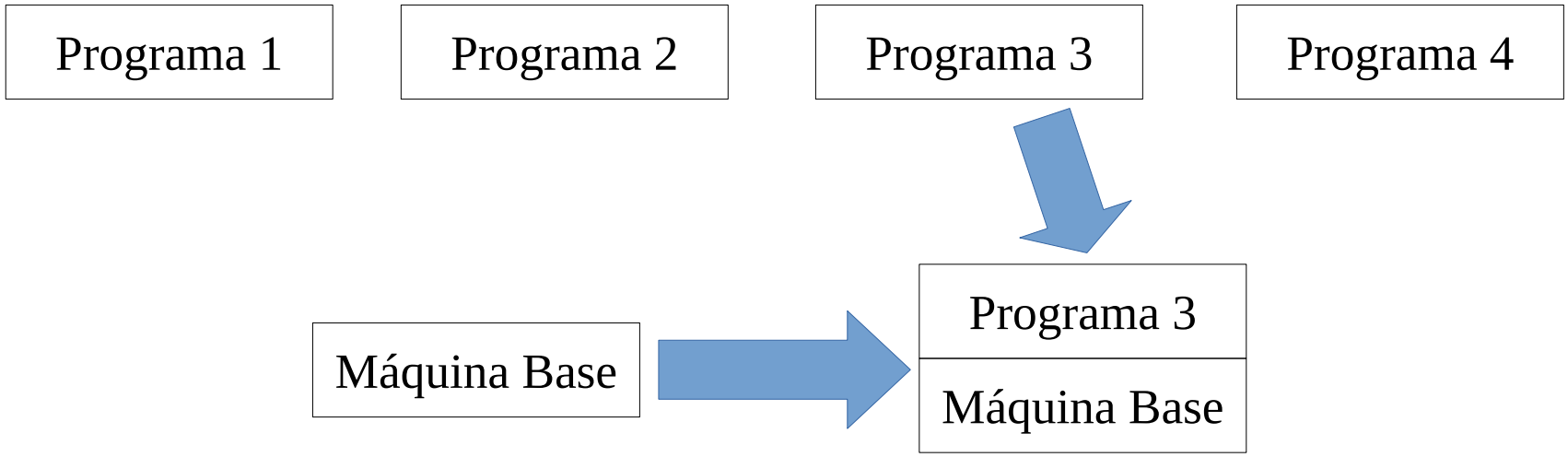
PARTE 1:
Introducción a los Sistemas Operativos y a su relación con la Arquitectura de Computadores

Sección 1. Introducción

¿Qué es un ordenador?

Máquina programable para el tratamiento de la información.

El comportamiento del ordenador lo determina el programa que ejecute.

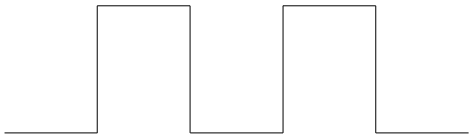


Ordenadores digitales

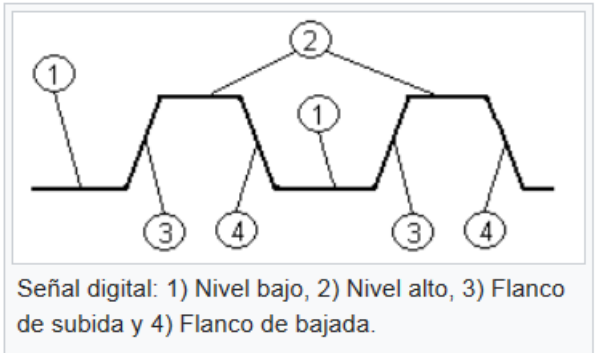
Trabajan con señales discretas (a diferencia de los ordenadores analógicos), esto es, señales que solo pueden tomar una serie de valores.

Mayoritariamente, señales digitales binarias: dos valores, que se pueden representar como 0 y 1 (o falso y verdadero, o 0v y +5v…).

Idealmente:



"Teóricamente":



Señal digital: 1) Nivel bajo, 2) Nivel alto, 3) Flanco de subida y 4) Flanco de bajada.

"Realmente":

Señal digital con ruido

Internamente usan circuitos síncronos (tienen un reloj interno que marca "el ritmo") por ser más fáciles de diseñar.

Programa almacenado en memoria (más fácil de programar).

Ventajas del procesamiento digital

“Fácil” codificación de la información.

Fácil procesamiento electrónico.

Menos sensibilidad al “ruido”.

Fácil replicación (y original = copia).

Densidad de almacenamiento muy grande.

Fundamentado matemáticamente (álgebra de Boole y otros).

Procesamiento electrónico

Basado en el transistor.

Alta velocidad de conmutación (varios GHz).

Alta miniaturización (circuitos integrados, 7nm).

Alta fiabilidad.

Bajo coste.

“Bajo” consumo. Ej.:

- 125w => Intel Core i7-11700KF, 14 nm, 8 núcleos, 3.6 GHz, año 2021 https://ark.intel.com/content/www/us/en/ark/products/212048/intel-core-i711700kf-processor-16m-cache-up-to-5-00-ghz.html
- 35w => Intel Core i7-11700T, 14 nm, 8 núcleos, 1.4 GHz, año 2021 https://ark.intel.com/content/www/us/en/ark/products/212251/intel-core-i711700t-processor-16m-cache-up-to-4-60-ghz.html

Sistema de numeración decimal

Base 10. Símbolos: 0 al 9.

Sistema posicional: $428 = 4x10^2 + 2x10^1 + 8x10^0$

Fácil construir cualquier número (natural):

Número ::= Dígito [Dígito]

Dígito ::= 0|1|2|3|4|5|6|7|8|9

"Fácil" hacer operaciones.

Muy fácil multiplicar y dividir por 10.

Ej. Sistema no posicional: Romano (Ej.: XXIX).

Ej. Sist. posic. no decimal: Sexagesimal (base 60).

Sistema de numeración binario (1/4)

Base 2. Símbolos: 0 y 1.

Sistema posicional: 101 bin. = $1x2^2 + 0x2^1 + 1x2^0$ = 1x4 + 0x2 + 1x1 = 5 dec.

“Fácil” hacer operaciones.

Muy fácil multiplicar y dividir por 2.

Sistema de numeración binario (2/4)

Bit: Mínima unidad información ≈ 1 dígito binario.

Byte: 8 bits (2^8 = 256 combinaciones).

Palabra (word): Unidad “natural” de organización de la memoria o los registros de la CPU. Suele coincidir con el tamaño (bits) que se usa para representar los enteros.

Los bits se numeran desde cero empezando por la derecha (bit menos significativo/lsb):

1	0	1
bit 2	bit 1	bit 0
msb		*lsb*

Sistema de numeración binario (3/4)

Coma fija:

- Sin signo (binario puro): Ej.: 11111111 bin. = 255 dec. Extensión añadiendo ceros por la izquierda.
- Complemento a dos: Ej.: 11111111 bin. = -1 dec. Extensión replicando el msb por la izquierda.

En complemento a dos, el bit más significativo indica el signo:

- 0=>positivo, binario puro (máx. 127).
- 1=>negativo, complemento a dos (mín. -128).

Sistema de numeración binario (4/4)

Acarreo (*Carry*): bit “extra” para suma (1 + 1 = 0 y C=1). También se usa para almacenar el bit “saliente” en los desplazamientos binarios lógicos o aritméticos. Ej.: Desplazamiento lógico a la izquierda de 1011 es 0110, con C=1.

Desbordamiento en suma (*overflow*): Cuando el valor resultante excede la capacidad de representación. Se detecta (complemento a dos):

A>=0 y B>=0 y A + B < 0

A < 0 y B < 0 y A + B>=0

Sistema de numeración hexadecimal

Base 16. Símbolos: 0 a 9 y A a F.

Sistema posicional: 7E4 hex. = $7x16^2 + Ex16^1 + 4x16^0 = 7x256 + 14x16 + 4x1 = 2020$ dec.

Cuatro veces más corto de escribir que el binario.

Muy fácil conversión binario ↔ hexadecimal

7	E	4
0111	1110	0100

Sistema de numeración octal

Base 8. Símbolos: 0 a 7

Sistema posicional: 421 oct. = $4x8^2 + 2x8^1 + 1x8^0$ = 4x64 + 2x8 + 1x1 = 273 dec.

Tres veces más corto de escribir que el binario.

Muy fácil conversión binario ↔ octal

4	2	1
100	010	001

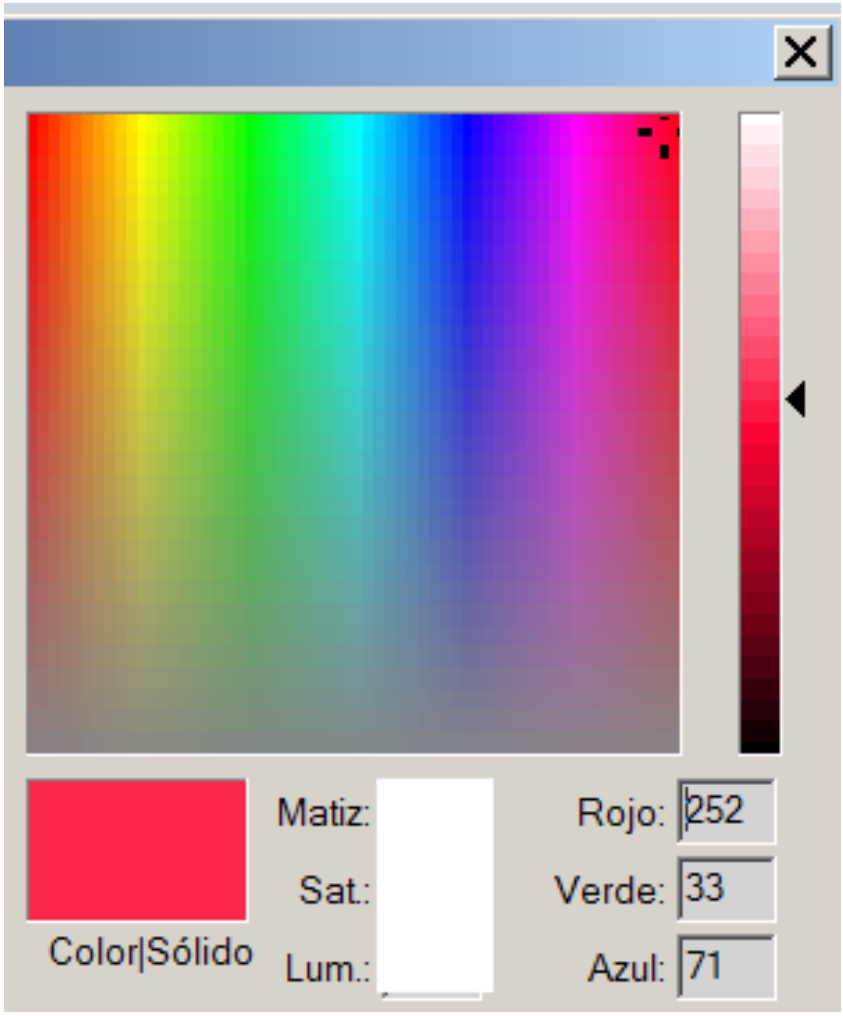
Representar la realidad con números (1/2)

"Letras": ASCII, Unicode, UTF-8…

Binario	Hex	Dec	Representa
0011 0000	30	48	0
0011 0001	31	49	1
0011 0010	32	50	2
0100 0000	40	64	@
0100 0001	41	65	A
0100 0010	42	66	B
0100 0011	43	67	C
0110 0001	61	97	a
0110 0010	62	98	b

Representar la realidad con números (2/2)

Colores: RGB, paletas…

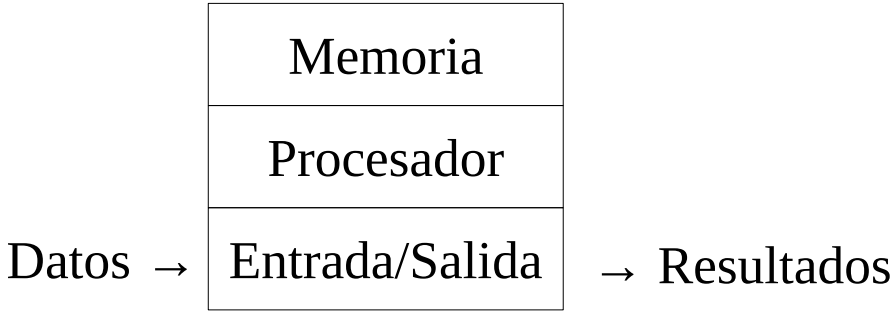


Estructura general de un ordenador

Memoria: Almacena datos y programas.

Procesador (*Central Processing Unit/CPU*): Realiza las operaciones indicadas en los programas.

Entrada/Salida: Comunicación con el “exterior”.



Programación de un ordenador

El procesador solo entiende código máquina (“números”).

El código máquina es muy complicado (para las personas :-)). Por ello se crean lenguajes que facilitan la programación.

Lenguaje bajo nivel: ensamblador.

Lenguajes alto nivel: C, Java, Python...

Lenguaje ensamblador

Utiliza mnemónicos en vez de números.

Permite el uso de etiquetas y otros mecanismos para facilitar la programación.

Se traduce de una forma “sencilla” a código máquina.

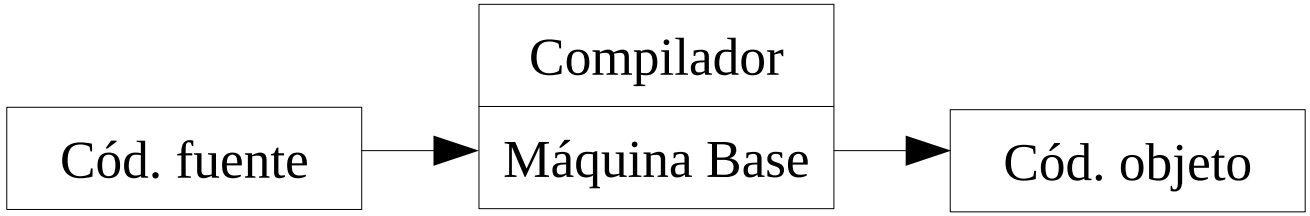
Permite un control “total” del sistema.

Ejemplo: mover al registro B el contenido posición memoria 64.

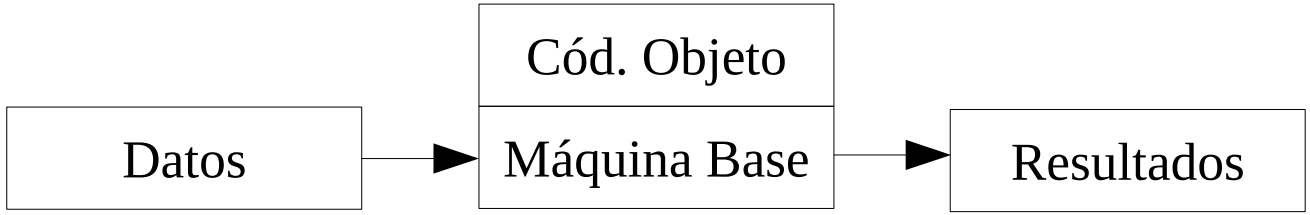
```
MOV B, [64]   ;en ensamblador
03   01   00   40    ← en código máquina (hex).
```

Lenguaje alto nivel compilado (ej.: C)

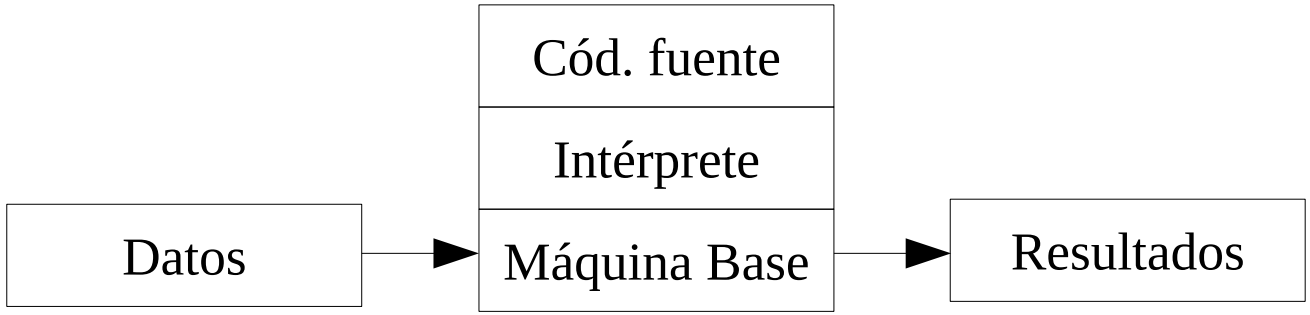
Compilación:



Ejecución:

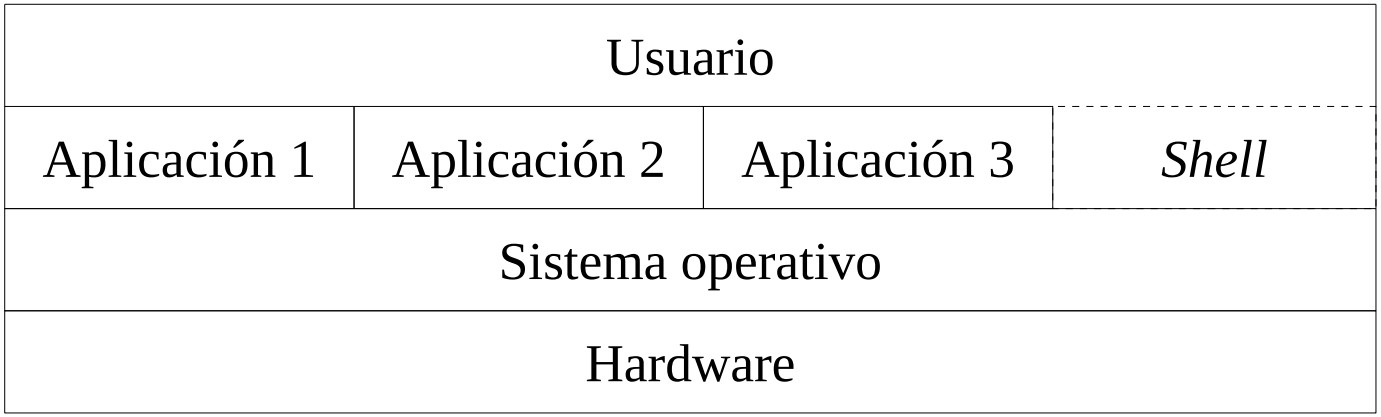


Lenguaje alto nivel interpretado (ej.: Python)



El sistema operativo (S.O.)

- Es un software.
- Es un componente fundamental de los sistemas informáticos (Ej.: Windows, Linux, iOS, Android…).
- Facilita el uso de los recursos físicos (pantalla gráfica, ratón, teclado, disco, etc.) y los gestiona.



Qué hace un sistema operativo

Virtualiza los recursos físicos (CPU, memoria, disco…) y los pone a disposición de los usuarios/procesos.

Gestiona la concurrencia (paralelismo, threads…).

Guarda información de forma persistente.

Objetivos en el diseño de un S.O.

Crear abstracciones que faciliten uso del sistema.

Conseguir un alto rendimiento del sistema.

Proteger/ “aislar” los procesos (y el propio S.O.).

Conseguir una alta fiabilidad.

Otros: Eficiente energéticamente, seguro (más allá de la mera protección de procesos), “móvil”, etc.

Carga del sistema operativo (resumen arranque sistema/*booting*)

1. Se conecta la alimentación y, tras estabilización tensiones, se manda señal power good.
2. Se inicializa la CPU y comienza a ejecutar un programa que se encuentra en la BIOS (“ROM”).
3. La BIOS realiza el power-on self-test (POST)
4. La BIOS determina el dispositivo de arranque, carga el primer sector en memoria y lo ejecuta (programa de arranque).
5. El programa de arranque cargar y lanza la ejecución del sistema operativo.

Historia SS.OO.: Inicios

- Ningún sistema operativo (máquina desnuda).
- Aparecen los ensambladores y cargadores.
- Más tarde librerías que facilitan las operaciones más comunes.
- Se desperdicia mucho tiempo en la carga de programas.

Historia SS.OO.: Sistemas batch

- Procesamiento por lotes: ejecuta programas uno detrás de otro (todavía no interactivos: tiempo ordenadores muy caro).
- Es un S.O. rudimentario llamado Monitor que siempre está en memoria. Éste pasa el control al programa correspondiente del lote una vez lo ha cargado y recibe de vuelta el control al completarse.
- Van apareciendo avances: protección de memoria, temporizador, instrucciones privilegiadas (modo supervisor y modo usuario) e interrupciones.

Historia SS.OO.: Batch multitarea

- También llamados multiprogramados.
- Aprovecha tiempos de espera de E/S de un programa para ejecutar otro.
- La interactividad ya no consume recursos, pero no es eficiente desde punto de vista del usuario (prioriza procesamiento sobre interactividad, esto es, no “quita” la CPU a un proceso hasta que no llega una espera por E/S).
- Añade mucha complejidad (y potencia) frente a un batch simple: requiere gestión de memoria y planificador tareas.

Sistemas operativos de tiempo compartido

- Protección contra monopolización de la CPU: interrupciones/time slicing.
- El tiempo de ejecución asignado a cada programa se divide en quantum.
- Si un programa en ejecución agota su quantum, la CPU pasa a ejecutar otro programa (round-robin)

	Batch multitarea	**Tiempo compartido**
Objetivo principal	Maximizar uso CPU	Minimizar tiempo respuesta
Origen de las directivas	Lenguaje control de trabajos incluido en propio programa	Entrada de comandos por el terminal

Sistemas operativos de tiempo real

- Objetivo: garantizar que los programas se ejecutan dentro de un plazo de tiempo acotado.
- Para satisfacer restricciones de tiempo real no basta con tener un hardware más potente.
- Tiempo real no es sinónimo de rápido.
- Suelen encontrarse en sistemas empotrados e IoT.
- Dos tipos de aplicaciones de tiempo real: duras (hard) y blandas (soft).

SS.OO. y arquitectura de computadores

Los sistemas operativos precisan de mecanismos proporcionados por la arquitectura de los computadores, en particular:

- Modo dual de ejecución: supervisor y usuario.
- Interrupciones.
- Gestión de excepciones.
- Llamadas al sistema (traps).
- Gestión de espacios de direccionamiento.

Modo dual de ejecución

Usualmente CPU dos modos ejecución: Supervisor (privilegiado o kernel) y usuario.

En modo supervisor: acceso a todas las instrucciones, incluidas las potencialmente “peligrosas” (instrucciones privilegiadas): acceso E/S, gestión interrupciones, protección memoria, etc.

El núcleo del sistema operativo se ejecuta en modo kernel.

Va a permitir gestionar los recursos, proteger la memoria y aislar fallos en programas de los usuarios (ej.: división por cero).

Interrupciones

Eventos externos a la CPU que cambian el flujo normal de ejecución.

Se pueden gestionar (habilitar, enmascarar, etc.).

Se tratan en modo supervisor (se guarda previamente el contexto para poder retomar la tarea interrumpida) por el manejador de interrupciones.

Interrupciones por E/S o DMA (Direct Memory Access): Permiten una mejor gestión de recursos.

Interrupciones por *timers*: Permiten, en particular, la gestión de procesos.

Excepciones

Eventos inesperados internos a la CPU: división por cero, intento de ejecución de instrucción privilegiada en modo usuario, intento de acceso a memoria protegida o de otro proceso, etc.

Se tratan en modo supervisor (se guarda previamente el contexto por si fuera posible retomar la tarea interrumpida) por el manejador de excepciones.

Permite al S.O. “matar” un proceso de usuario sin afectar al resto de procesos en ejecución.

Llamadas al sistema

Las llamadas se activan mediante una instrucción “trap”, que es una especie de excepción generada por software.

Se tratan en modo supervisor (se guarda previamente el contexto para retornar tras su terminación) por el manejador de llamadas al sistema.

Las llamadas al sistema se identifican generalmente por un número. Esto permite proteger el sistema (se valida, por ejemplo, el número de llamada).

Usualmente devuelven un resultado (o si todo ok).

Gestión de espacios de direccionamiento

La memoria se va a compartir por el kernel del S.O. y los programas de usuario (o S.O.) en ejecución.

Los programas para su ejecución y (generalmente) los datos para su lectura/escritura deben estar en la RAM.

La RAM es “escasa”. Si un programa no se está ejecutando puede hacerse “swap” de la memoria a disco.

Unos procesos no deben acceder (si no tienen autorización) a zonas de memoria de otros (sean de datos, código, pila…) para lectura, escritura o ejecución.

La Memory Management Unit (MMU) es el hardware que va a permitir gestionar la memoria.

Referencias:

- “Fundamentos de Programación” (tema 1). José A. Cerrada y Manuel E. collado, 2010. Ramón Areces.
- “Operating Systems: Internals and Design Principles”. William Stallings, 2018. Pearson.
- “Chapter 1. Introduction”. Luis Tarrataca. CEFET-RJ.
- Wikipedia.