29-09-2025

Unidad 1 – Conceptos fundamentales de los sistemas informáticos

1. **Definición y componentes del sistema informático** (hardware, software, firmware, usuarios).

2. **Arquitectura Von Neumann**: CPU, registros, buses, ciclo de instrucción (con ejemplos de ejecución).

3. **Jerarquía de memoria**: explicación detallada con tabla de latencias y ejemplo de impacto en un algoritmo.

4. **Subsistemas de almacenamiento**: HDD, SSD, NVMe y niveles de RAID.

5. **Mecanismos de E/S**: polling, interrupciones y DMA (con casos prácticos como teclado y tarjeta de red).

6. **Sistema operativo**: gestión de procesos, memoria, archivos y seguridad (modo kernel vs usuario).

7. **Técnicas de memoria avanzada**: paginación, segmentación, MMU y TLB.

8. **Sistemas de archivos**: inodos, journaling y comparación ext4/NTFS.

9. **Conceptos de fiabilidad y seguridad**: redundancia, backup, disponibilidad.

1. Definición de sistema informático

Un **sistema informático** es un conjunto de **recursos físicos (hardware), lógicos (software) y humanos (usuarios)**, organizados para procesar, almacenar y transmitir información.

Componentes esenciales:

- **Hardware** → recursos físicos: CPU, memoria, buses, dispositivos de E/S.
- **Firmware** → software de bajo nivel que controla el hardware (BIOS, UEFI, microcódigo).
- **Sistema operativo** → kernel que abstrae hardware y gestiona recursos.
- **Aplicaciones** → programas que utilizan los servicios del SO.
- **Usuarios** → humanos o procesos que interactúan.

En desarrollo de aplicaciones (DAM), **cada programa compite por CPU, RAM y E/S**, por lo que es vital entender la base del sistema.

2. Arquitectura de Von Neumann

Elementos principales:

- **CPU (Unidad Central de Proceso)**
    - **Unidad de Control (UC)**: interpreta instrucciones binarias (opcodes), coordina señales de control en buses.
    - **ALU (Unidad Aritmético-Lógica)**: suma, resta, comparaciones, operaciones lógicas.
    - **Registros**:
        - **Generales**: EAX, EBX en x86; R0, R1 en RISC.
        - **Especiales**:
            - PC (Program Counter): dirección de la próxima instrucción.
            - IR (Instruction Register): instrucción en ejecución.
            - SP (Stack Pointer): referencia a la pila.
            - FLAGS/PSW: banderas de estado (zero, carry, overflow).
- **Memoria principal (RAM)**: almacena datos e instrucciones en ejecución.
- **Buses del sistema**:
    - **Direcciones** → posición de memoria.
    - **Datos** → bits a transferir.
    - **Control** → señales de sincronización (lectura, escritura, interrupciones).
- **Dispositivos de E/S**: teclado, ratón, monitor, disco, red.

Ciclo de instrucción (fetch–decode–execute)

1. **Fetch**: CPU lee instrucción apuntada por PC.
2. **Decode**: UC interpreta opcode y operandos.
3. **Execute**: ALU u otra unidad ejecuta.

4. **Write back**: resultado en registro o memoria.
5. PC se incrementa (salvo saltos).

Un **bucle infinito** nunca libera el ciclo, saturando la CPU.

3. Jerarquía y latencia de memoria

Nivel	Latencia aprox.	Tamaño típico
Registros CPU	1 ciclo	32–256 bytes
Caché L1	1–2 ns	32–64 KB
Caché L2	4–12 ns	256 KB–1 MB
Caché L3	20–40 ns	4–32 MB
RAM DDR4/DDR5	80–120 ns	8–64 GB
SSD NVMe	100 µs	1–4 TB
HDD	5–10 ms	500 GB–2 TB

Impacto en programación:

- Algoritmos con **acceso secuencial** aprovechan **localidad espacial** → mejor rendimiento.
- Algoritmos con **acceso aleatorio** penalizan cache → mayor latencia.

4. Subsistemas de almacenamiento

- **HDD (disco magnético)**: acceso mecánico, latencia ~10 ms, cuello de botella clásico.
- **SSD SATA**: interfaz limitado a 600 MB/s.
- **SSD NVMe (PCIe)**: paralelismo de colas, >3 GB/s.

RAID (Redundant Array of Independent Disks)

- **RAID 0**: striping → máximo rendimiento, sin redundancia.
- **RAID 1**: mirroring → duplicación, seguridad alta, coste en capacidad.
- **RAID 5**: paridad distribuida, compromiso rendimiento/seguridad.
- **RAID 10**: striping + mirroring → rendimiento + redundancia, usado en servidores.

En bases de datos, se exige RAID con redundancia (no toleran pérdida de bloques).

5. Mecanismos de Entrada/Salida (E/S)

- **Polling**: CPU consulta al dispositivo → ineficiente.
- **Interrupciones (IRQ)**: el dispositivo interrumpe a la CPU para avisar de eventos.
- **DMA (Direct Memory Access)**: el controlador transfiere datos directamente a RAM sin pasar por CPU.

Ejemplo:
Una tarjeta de red descarga paquetes en memoria vía DMA → el kernel los procesa luego.

6. Sistema operativo como capa de abstracción

El **kernel** ofrece servicios esenciales:

- **Gestión de procesos**: planificación (round-robin, colas multinivel, prioridades).
- **Gestión de memoria**:
    - Segmentación: código, pila, datos.
    - Paginación: traducción virtual↔física con MMU y TLB.
- **Gestión de archivos**: inodos, journaling (ext4, NTFS).
- **Gestión de dispositivos**: drivers → abstracción estándar.
- **Seguridad**: modos usuario/kernel, control de accesos, auditoría.

La separación **modo usuario vs modo kernel** protege el hardware y la estabilidad del sistema.

Conceptos técnicos

1. Firmware

- Es un **software de bajo nivel** grabado en chips de memoria no volátil (ROM, EEPROM, Flash).
- Controla el hardware en su nivel más básico.
- Ejemplo: la **BIOS/UEFI** de un PC, que inicializa componentes antes de cargar el sistema operativo.

Piensa en él como el “manual de instrucciones” que permite que el hardware sepa arrancar y ser usable.

2. Registros de la CPU

- Memorias **ultrarrápidas** dentro del procesador.
- Usadas para operaciones inmediatas (sumar, comparar, mover datos).
- Tipos:
    - **Generales**: contienen datos y operandos.
    - **Program Counter (PC)**: indica la dirección de la próxima instrucción.
    - **Instruction Register (IR)**: almacena la instrucción actual.
    - **Stack Pointer (SP)**: referencia la cima de la pila.
    - **FLAGS/PSW**: banderas de estado (ej. `Z=1` si el resultado es cero).

Son mucho más rápidos que la RAM → de ahí su importancia en rendimiento.

3. Ciclo de instrucción (fetch–decode–execute)

1. **Fetch**: la CPU toma la instrucción de la RAM usando el valor del PC.
2. **Decode**: la UC interpreta la instrucción (qué operación y con qué datos).
3. **Execute**: la ALU u otra unidad realiza la operación.
4. **Write-back**: el resultado se guarda en registros o memoria.
5. El PC se incrementa para apuntar a la siguiente instrucción.

Este ciclo ocurre **millones de veces por segundo** (ej. 3 GHz ≈ 3.000 millones de ciclos).

4. Jerarquía de memoria

Se organiza por **velocidad, coste y capacidad**:

- **Registros (nanosegundos, bytes).**
- **Caché L1/L2/L3 (ns, KB–MB).**
- **RAM (ns, GB).**
- **Discos (ms, GB–TB).**

Cuanto más rápida → más cara → menos capacidad.
Ejemplo práctico: un acceso a RAM tarda cientos de ciclos más que un acceso a caché.

5. DMA (Direct Memory Access)

- Técnica donde un **controlador de E/S** transfiere datos entre un dispositivo y la RAM **sin intervención de la CPU**.
- Ejemplo: una tarjeta de red coloca paquetes directamente en memoria.
- Ventaja: libera a la CPU de estar moviendo datos byte a byte.

6. RAID

Conjunto de técnicas para combinar discos:

- **RAID 0 (striping)**: divide datos entre discos → más velocidad, cero redundancia.
- **RAID 1 (mirroring)**: duplica datos → seguridad, pero mitad de capacidad.
- **RAID 5 (paridad distribuida)**: tolera fallo de 1 disco, buen equilibrio.
- **RAID 10 (striping + mirroring)**: alto rendimiento + redundancia.

Muy usado en **servidores y bases de datos**.

7. Interrupciones

- Señales que envía un dispositivo a la CPU para avisar de un evento.
- Ejemplo: el teclado interrumpe cuando se pulsa una tecla.
- Ventaja: la CPU no pierde tiempo “preguntando” (polling).

8. Modo usuario vs. modo kernel

- **Modo kernel**: el SO accede directamente a hardware y memoria protegida.
- **Modo usuario**: las aplicaciones trabajan con abstracciones (archivos, sockets).

Esto evita que un programa normal pueda bloquear o dañar el sistema.

9. Paginación y MMU

- La **memoria virtual** divide el espacio en páginas (ej. 4 KB).
- La **MMU (Memory Management Unit)** traduce direcciones virtuales a físicas.
- Permite:
    - Que cada proceso crea que tiene “su propia RAM”.
    - Proteger memoria de procesos distintos.

10. Inodos y journaling

- En sistemas de archivos tipo **ext4**:
    - Cada archivo se representa con un **inodo** que guarda permisos, propietario, tamaño, punteros a bloques de datos.
    - **Journaling**: sistema de registro previo que evita corrupción tras un apagón (se anotan operaciones antes de ejecutarlas).

Actividades prácticas – Unidad 1

Actividad 1 – Análisis del hardware y del sistema operativo

Enunciado

Realiza un informe técnico sobre la **arquitectura de tu equipo** y la **configuración del sistema operativo**. Deberás identificar CPU, memoria, almacenamiento y versión del sistema operativo, justificando cómo estos elementos afectan al rendimiento.

Objetivo

Familiarizarse con los **componentes físicos y lógicos del sistema informático** y comprender su impacto en el rendimiento y estabilidad.

Paso a paso

1. **Identificar hardware**:
    - En **Windows**:
        - `winver` → versión del sistema.
        - `systeminfo` en PowerShell o CMD → datos completos.
        - Administrador de tareas → pestaña “Rendimiento” (CPU, RAM, discos).
    - En **Linux**:
        - `lscpu` → información de la CPU.
        - `lsblk` → discos y particiones.
        - `free -h` → memoria disponible.
        - `uname -a` → kernel y versión del SO.
2. **Documentar hallazgos**:
    - Número de núcleos, frecuencia CPU, arquitectura (x86-64, ARM…).

- RAM instalada vs RAM disponible.
- Tipo de disco (HDD, SSD, NVMe).
- Versión de SO y arquitectura (32/64 bits).

3. **Analizar impacto**:
    - Explica cómo el tipo de disco afecta al tiempo de carga.
    - Justifica por qué la arquitectura del procesador (32 vs 64 bits) limita el uso de memoria.

Entrega: informe con capturas de pantalla y explicación técnica.

Actividad 2 – Medición de rendimiento de memoria y disco

Enunciado

Compara el rendimiento de **memoria RAM y disco** de tu equipo ejecutando pruebas de latencia y transferencia. Interpreta los resultados relacionándolos con la jerarquía de memoria.

Objetivo

Entender cómo la **latencia de acceso** influye en el rendimiento global de un sistema.

Paso a paso

1. **Instalar herramientas**:
    - **Windows**: descargar **CrystalDiskMark** (para disco) y usar `winsat mem` para memoria.
    - **Linux**:
        - `hdparm -Tt /dev/sdX` → pruebas de disco.
        - `sysbench memory run` → pruebas de memoria.
2. **Ejecutar pruebas**:
    - Medir velocidad de lectura/escritura secuencial y aleatoria en disco.
    - Medir velocidad de transferencia de memoria RAM.
3. **Analizar resultados**:
    - Comparar valores obtenidos con latencias teóricas de jerarquía de memoria (RAM vs SSD vs HDD).
    - Reflexionar: ¿qué operaciones se benefician de un SSD NVMe frente a un HDD?

Entrega: tabla con resultados + interpretación técnica.

Actividad 3 – Gestión de procesos y memoria en tiempo real

Enunciado

Monitorea en tiempo real los **procesos y consumo de recursos** de tu equipo. Identifica qué programas consumen más CPU y memoria, y cómo el sistema operativo planifica su ejecución.

Objetivo

Comprender la **gestión de procesos y multitarea** en un sistema operativo moderno.

Paso a paso

1. **Abrir monitor de procesos**:
    - **Windows**: Administrador de tareas → pestañas “Procesos” y “Detalles”.
    - **Linux**: ejecutar `top` o `htop`.
2. **Analizar en vivo**:
    - Localizar el proceso que más CPU consume.
    - Identificar consumo de RAM de navegadores (varios procesos por pestañas).
    - Ver cómo cambia la prioridad (`nice` en Linux, “Establecer prioridad” en Windows).
3. **Probar multitarea**:
    - Abrir varias aplicaciones (ej. navegador, editor de texto, vídeo en streaming).
    - Observar cómo el sistema reparte CPU y memoria.
    - En Linux: usar `kill -STOP pid` y `kill -CONT pid` para pausar y reanudar procesos.