Fundamentos de Computadores

Bibliografía:

- Diapositivas de Fundamentos de Computadores (Curso 2017–18).
- Boletines de prácticas de Fundamentos de Computadores, Título de Grado en Ingeniería Informática, Facultad de Informática, Universidad de Murcia (Curso 2017–18).

Capítulo 1

Introducción

La **informática** es la ciencia que estudia el procesamiento automático de la información. Su consolidación como ciencia se produce a partir de los años 40 con el desarrollo de los **computadores**, con los cuales esta se ha podido desarrollar.

Un computador es una máquina que procesa información siguiendo las instrucciones de un programa, y se comunica a través de los dispositivos de entrada y de salida. También dispone de dispositivos para almacenar información y procesarla. Esta está expresada en forma binaria (0's y 1's). Distinguimos:

- **Hardware:** Conjunto de componentes que integran la parte material de un computador, incluyendo componentes eléctricos, electrónicos y mecánicos.
- **Software:** Conjunto de programas e instrucciones para ejecutar ciertas tareas en un computador. Es intangible, aunque se encuentren almacenados en hardware.

1.1. Hardware

Tradicionalmente, los computadores siguen el **esquema de Von Neumann**, que consiste en una unidad central de proceso (CPU), constituida por una unidad de control (UC) y un camino de datos (CD); la memoria principal, y los dispositivos de entrada y salida, incluyendo almacenamiento.

Los **buses** son un conjunto de hilos paralelos que conectan unidades, como se muestra en la figura. Cada hilo transmite 1 bit a la vez, y el **ancho de bus** es el nº de hilos que tiene un bus. En concreto:

- El **bus de direcciones** transmite la dirección en memoria, por lo que su ancho define la máxima memoria instalable como 2^n bytes.
- El **bus de datos** transmite información, por lo que a mayor ancho, mayor cantidad de información se puede transmitir en una sola operación.

La **memoria** está formada por un conjunto de celdas, normalmente de 1 byte, en la que se guardan datos e instrucciones. Cada celda tiene una **dirección** única y correlativa. Para leerla, la CPU «pone» la dirección en el bus de direcciones, activa el hilo de lectura en el bus de

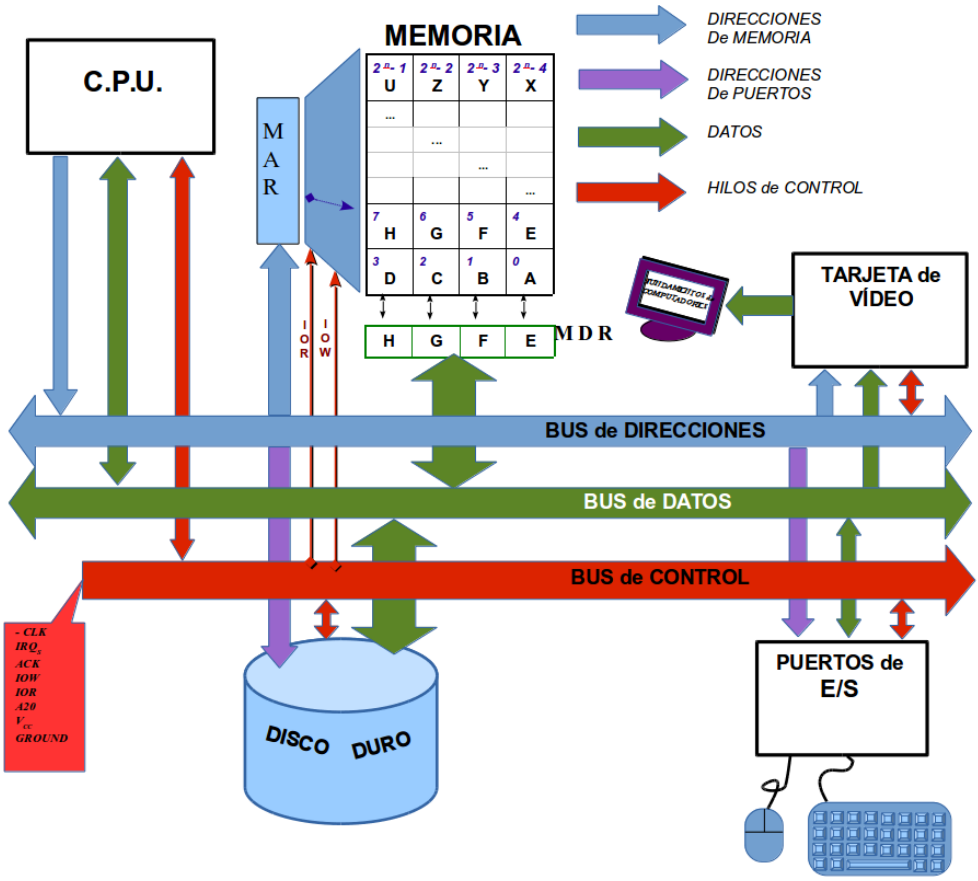


Figura 1.1: Esquema von Neumann en un ordenador moderno.

control y la memoria «deja» el contenido de la(s) celda(s) en el bus de datos. Para escribir, la CPU «pone» la dirección en el bus de direcciones, el dato en el bus de datos y activa el hilo de escritura en el bus de control.

1.2. Software

El **sistema operativo** un programa que gestiona los recursos del computador. Es el primer programa en ejecutarse (*), gestiona a todos los demás y actúa de intermediario con el hardware. Es un **programa de sistema**, al igual que el compilador y el ensamblador. Por contra, las **aplicaciones** son programas orientados al usuario, como procesadores de texto, hojas de cálculo, navegadores web, reproductores multimedia, etc.

El **firmware** es software de bajo nivel, almacenado de forma semipermanente. En general, puede cambiarse, pero no tan fácilmente como el resto de software.

1.3. Internet

Internet es un conjunto descentralizado de redes de comunicación interconectadas mediante protocolos estandarizados, de forma que las redes físicas que las componen funcionan como una única red lógica mundial. Existen muchos **servicios** proporcionados a través de la red, como la Web, correo electrónico, transmisión de archivos, chats, acceso remoto, etc.

1.4. Conceptos

La unidad mínima de información es el **bit**, que puede valer 0 o 1. Se pueden almacenar como tensión alta o baja en un circuito, superficie magnetizada en uno u otro sentido, superficie perforada, señal de alta o baja frecuencia en un cable, presencia o no de señal luminosa en un cable de fibra óptica, etc. Generalmente se usan secuencias de bits: 1 **byte** = 8 bits, codifica $2^8 = 256$ estados distintos.

Múltiplos del byte: $1\,\text{KB}$(kilobyte) $= 2^{10}\,\text{B}$, $1\,\text{MB}$(megabyte) $= 2^{20}\,\text{B}$, $1\,\text{GB}$(gigabyte) $= 2^{30}\,\text{B}$, $1\,\text{TB}$(terabyte) $= 2^{40}\,\text{B}$, $1\,\text{PB}$(petabyte) $= 2^{50}\,\text{B}$, $1\,\text{EB}$(exabyte) $= 2^{60}\,\text{B}$. Los prefijos también se aplican a bits.

Las variables numéricas que ocupan más de un byte se pueden guardar en memoria como:

- ***Little endian***: El byte menos significativo está en la posición de memoria más baja.
- ***Big endian***: En la posición más alta.

El **tiempo de ejecución** o **de respuesta** es lo que tarda el computador en realizar una tarea, incluyendo E/S, mientras que el **tiempo de CPU** se refiere solo al tiempo que tarda el procesador en realizar un cálculo, y se calcula como:

$$Tiempo_{CPU} = \frac{Instrucciones_{APLICACIÓN} \cdot \frac{Ciclos(Media)}{Instrucción}}{Frecuencia\ procesador}$$

1.5. Historia

En 1791-1871, Babbage intentó diseñar una máquina mecánica capaz de resolver problemas matemáticos, la cual nunca fue terminada. Podemos dividir la historia de la informática en varias generaciones de ordenadores:

- **Primera generación (1943-62):** A principios de siglo, Fleming inventó la válvula de vacío, permitiendo el desarrollo de la electrónica y los primeros ordenadores, de los que destacamos:
  - **Proyecto ENIAC (Eckert y Mauchly, 1943-46):** *Electronic Numerical Integrator and Calculator*: 30 toneladas, 180 m^2, 18000 válvulas de vacío, frecuencia 0,1 MHz, 20 registros de 10 dígitos decimales, programación cableando directamente, 1900 sumas por segundo.
  - **UNIVAC:** *Universal Automatic Computer*.
  - **Proyecto EDVAC (Von Neumann, 1952):** *Electronic Discrete Variable Automatic Computer*: El primero con programas almacenados; saltos condicionales, válvulas de vacío.
- **Segunda generación (1962-67):** Tras la invención del transistor en 1947; IBM System 360, PDP-8 (primer minicomputador, *Digital Equipment Corporation*).
- **Tercera generación (1967-78):** En 1958 se inventa el circuito integrado, con el que se integran varios elementos electrónicos en el mismo bloque. Aparece la microprogramación (propuesta por Wilkes en los 50) y el primer supercomputador (CDC 6000, *Control Data Corporation*, Seymour Cray, 1964), y en 1965 Wilkes propone el concepto de caché.
- **Cuarta generación (1971-):** En 1971 se diseña el primer microprocesador: Intel 4004, con 2300 transistores. Desde el 1981 y con el desarrollo de CPUs de Intel como el 8088 se desarrolla la informática de consumo.

Desde la mitad de los 90 con la aparición de la web, hay 3 grupos de ordenadores:

- **Ordenadores personales:** PCs, *tablets*, etc., para uso individual, con buen rendimiento a bajo coste.
- **Servidores:** Para muchos usuarios, buscan fiabilidad y escalabilidad. Distinguimos de clase baja (servidores de archivos, de impresión, etc.), media (centros de datos, «datacenters») y alta (supercomputadores con aplicaciones científicas).
- **Ordenadores embebidos** o **empotrados:** Ordenadores de coches, *gadgets*, etc. Para una sola aplicación, con grandes limitaciones, consumo de energía y fiabilidad.

En 2005, para ahorrar energía, aumentar el rendimiento sin aumentar el ciclo de reloj y mejorar la fiabilidad del diseño, surgen los procesadores **multinúcleo,** aunque requieren **programación paralela.**

1.6. Organización de un PC

- **Factor de forma** o **geometría:** Largo, ancho, ubicación de agujeros de montaje, tipo y ubicación de conectores y componentes, etc. Normalmente ATX, aunque los servidores suelen usar factores de forma como WTX, que tienen mayores dimensiones y permiten alojar varias CPUs y más memoria.
- **CPUs soportadas:** Depende del tipo de *socket* de CPU.
- **Módulos de memoria soportados:** Tipos (DDR [Double Data Rate], DDR2, DDR3, DDR4...), frecuencia del reloj de la memoria, nº de ranuras, capacidad máxima, etc.
- **Chipset:** Formado por:
  - **Memory Controller Hub** o **Northbridge:** Comunica la CPU, tarjeta gráfica, módulos de memoria y el Southbridge. Últimamente su funcionalidad se integra en el procesador.
  - **I/O Controller Hub** o **Southbridge:** Comunica el resto de elementos del sistema.
- **Ranuras de expansión:** PCI (más antiguo, siendo el más usado un bus de 32 bits a 33 MHz), PCI-X (servidores, 66 MHz o más) y PCI Express (más rápido, hasta 250 MB/s full duplex por canal, con ranuras de hasta 16 canales).
- **Interfaces de almacenamiento:** ATA (paralela, hasta 133 MB/s), actualmente sustituida por SATA (en serie, hasta 150 MB/s en la versión original), y SCSI (paralela), que será sustituida por SAS (serie).
- **Interfaces de audio y red.**
- **Puertos de conexión periféricos** como USB y Firewire.
- **Protección de la BIOS:** La BIOS (Basic Input/Output System) se encarga de arrancar el PC y dar soporte para ciertos dispositivos de entrada y salida. También ofrece una interfaz gráfica para configurar parámetros del PC. Actualmente la BIOS se almacena muchas veces en una Flash, por lo que se puede actualizar, pero de hacerse incorrectamente podría dejar el equipo inutilizable hasta cambiar el chip. Actualmente está siendo sustituida por UEFI (Unified Extensible Firmware Interface).

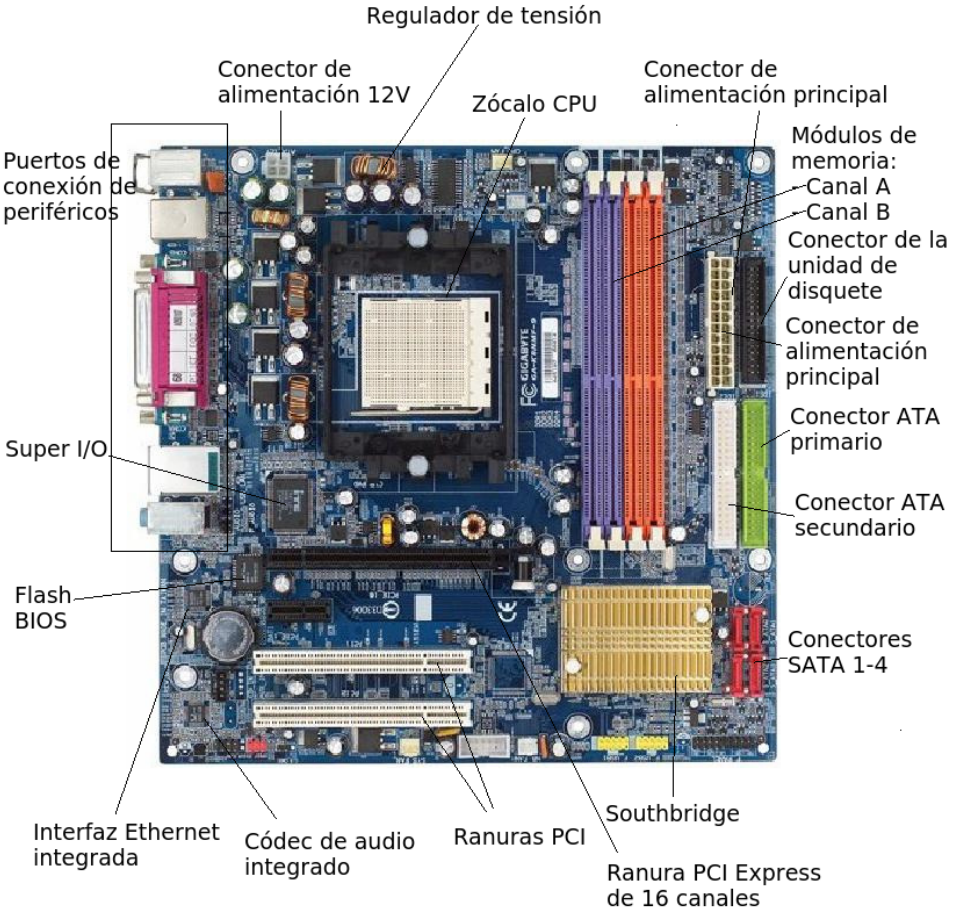


Figura 1.2: Placa base de un ordenador de escritorio típico.

Capítulo 2

Representación de la información

2.1. Representación de enteros

En un sistema de numeración posicional en base b, $N = \ldots + n_2 b^2 + n_1 b^1 + n_0 b^0 + n_{-1} b^{-1} + \ldots$. Nos centraremos en las bases 2 (binario), 8 (octal), 10 (decimal) y 16 (hexadecimal). Decimos $011)_2 = 3)_{10}$. Para convertir de decimal a binario, dividimos sucesivamente entre 2 la parte entera hasta obtener cociente binario, y tomamos los restos y el último cociente. Este es el bit más significativo, y el primer resto el menos significativo. Para la parte fraccionaria, multiplicamos por 2 la parte fraccionaria del número decimal, y el número binario se forma con lo que se va obteniendo.

El desplazamiento a la izquierda (añadir un 0 a la izquierda) multiplica por 2, y el desplazamiento a la derecha (eliminarlo) divide entre 2. También encontramos las operaciones lógicas `OR` ($0+0=0$, $0+1=1+0=1+1=1$), `AND` ($0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$; $1 \cdot 1 = 1$) y `NOT` ($\overline{0} = 1$; $\overline{1} = 0$).

Los valores sin signo se representan añadiendo 0s a la izquierda del número en binario hasta completar los 8, 16, 32, 64, etc. bits. Para los enteros con signo existen las siguientes representaciones:

- **Signo y magnitud:** El bit más significativo es de signo, y vale 1 si el nº es negativo. El problema es que el 0 tiene dos representaciones.
- **Sesgada:** Se añade una constante S (sesgo) al nº a representar para hacerlo positivo. Normalmente $S = 2^{\text{nº de bits}-1}$. Permite realizar restas como sumas.
- **Complemento a 2:** Si tenemos n bits, permite representar desde -2^{n-1} hasta $2^{n-1}-1$. Los positivos se representan normalmente. Para los negativos, se niegan todos los bits (`NOT`) del opuesto y se suma 1 al resultado. Así, además de poder realizar restas como sumas (despreciando los bits restantes), los números positivos se representan igual a como se representarían sin signo. Si el resultado de una operación sale del rango, se denomina **desbordamiento**, y la operación no funciona. Para representar un número con mayor cantidad de bits, se replica el bit de signo hacia la izquierda.

2.2. Códigos intermedios

Para expresar secuencias de bits de forma más concisa usamos la base **octal** ($8 = 2^3$ dígitos, $\{0, \ldots, 7\}$) agrupando los bits de 3 en 3 empezando por la coma decimal, así como la base **hexadecimal** ($16 = 2^4$ dígitos, $\{0, \ldots, 9, A, \ldots, F\}$), agrupándolos de 4 en 4. Las conversiones se realizan de forma similar a la conversión entre binario y decimal, pero con distinta base.

2.3. Representación de reales

Se usa la **notación exponencial**, en **coma flotante** o en **punto flotante:** $N = M \cdot B^E$, donde M es la mantisa, $B = 2$ es la base y E es el exponente. Se representa en tres campos:

- **Signo (S):** 0: positivo; 1: negativo.
- **Exponente (E):** Entero sesgado, con sesgo $S = 2^{n_E - 1} - 1$.
- **Mantisa (M):** Solo se representa la parte fraccionaria; la parte entera siempre es 1.

La norma IEEE 754 especifica reales de **simple precisión** (32 bits, $n_E = 8$), y de **doble precisión** (64 bits, $n_E = 11$). Situaciones especiales:

- Si $E = 0$, el 1 no está implícito. Así, si $E = 0$ y $M = 0$, el número es 0.
- Si $E = 2^{n_E} - 1$, entonces $n = +\infty$ o $-\infty$ si $M = 0$ o NaN (*Not a Number*) si $M \neq 0$.

Cuando un número como el resultado de una operación no se puede representar de forma exacta, se aplica el redondeo al par: Se toman los dos primeros bits que «no caben» en la mantisa (en orden, **bit de redondeo** y **bit retenedor**). Si el de redondeo es 0, se trunca. Si ambos son 1, se redondea al alza. En el caso restante, se redondea al par más cercano.

Algunos números que en decimal son exactos no se pueden representar con total exactitud porque en binario son periódicos, y algunas propiedades como la asociatividad de la suma pueden no cumplirse debido a errores de aproximación.

2.4. Representación de caracteres

Para representar un carácter entre un conjunto de n caracteres necesitamos $\lceil \log_2 n \rceil$ bits, de forma que a cada carácter le corresponde una combinación. El código es arbitrario, pero existen códigos normalizados:

- El ASCII (*American Standard Code for Information Interchange*) codifica desde los años 60 la mayoría de caracteres del idioma inglés. Usa 7 bits para 128 caracteres. Para rellenar 1 byte, el bit adicional se usaba para control de errores. Un ejemplo es el **bit de paridad**, que garantiza que el total de unos en el byte debe ser par. Si no lo es, ha habido un error en la transmisión.
- Más recientemente, este último bit se usa para extensiones de determinados idiomas. El ISO-8859-1 (Latin 1) incluye extensiones como acentos y «ñ». El ISO-8859-15 es similar pero añade el signo del euro.

- El Unicode permite representar cualquier sistema de escritura del mundo. Empezó siendo de 16 bits y actualmente define más de 1 millón de símbolos. A cada uno se le asigna un *code point* en el rango de 0 a $0x10FFFF$, que se representa como U+número. El conjunto de *code points* que caben en 16 bits se denomina *Basic Multilingual Plane* (BMP).
    - Los primeros 256 *code points* corresponden al ISO-8859-1.
    - Todos los *code points* se representan como secuencias de bits de varias formas, como el UTF-8 y el UTF-16 (*Unicode Transformation Format*). En UTF-8, cada carácter ASCII ocupa 1 byte, por compatibilidad. Para otros caracteres, se usan de 2 a 4 bytes.
    - En UTF-16, para distinguir entre *Big-Endian* y *Little-Endian*, se usa la llamada *Byte Order Mark* (BOM) al principio del texto, codificado como $0xFEFF$.

2.5. Representación de imágenes

Las imágenes se pueden representar mediante mapas de bits o de forma vectorial. Un **mapa de bits** está formado por una matriz de píxeles $M_{m,n}(px.)$ con resolución $n \times m$, de los cuales a cada uno se le asocia un color o un tono de gris. Se almacenan los píxeles sucesivamente. Algunos formatos de mapa de bits son BMP (Windows), PICT (Macintosh), PPM (*Portable Pix-Map*, de codificación sencilla) y JPEG (compresión normalmente con pérdida, buena calidad para fotos).

Mientras tanto, una **imagen vectorial** se representa como una colección de objetos como líneas, polígonos o textos, que se modelan mediante vectores y ecuaciones que se evalúan al visualizar las ecuaciones en pantalla. Son adecuados para gráficos geométricos e ideales para aplicaciones CAD, se pueden escalar a cualquier tamaño y suelen ocupar mucho menos espacio que los mapas de bits. Sin embargo, no son adecuadas para imágenes reales y suelen tener menor calidad de imagen. Algunos formatos son DXF (*Document eXchange Format*, para imágenes CAD), EPS (*Encapsulated PostScript*, de Adobe) y ODG (LibreOffice). A menudo estos pueden incluir mapas de bits embebidos.

El color se representa mediante escalas. En **escala de grises**, cada píxel toma un valor de gris. El modelo de color **RGB** es aditivo (la suma de los colores genera el blanco), se usa sobre todo en pantallas y representa la intensidad de rojo, verde y azul. El modelo **CMYK** es sustractivo (la suma de los colores genera el negro), se usa principalmente en impresoras y representa el cían, magenta, amarillo y negro.

2.6. Algunos formatos de archivo

2.6.1. PPM

El «Portable Pixel Map» está formado por un «número mágico» que identifica el tipo de archivo (caracteres **P6**), una cabecera y una ristra de bytes. La cabecera indica, entre otras cosas, el ancho y alto de la imagen y el valor máximo de un color, en decimal mediante caracteres ASCII (normalmente estos son los tres últimos elementos de la cabecera, los cuales suelen separarse por 0x0A). La ristra de bytes contiene los píxeles uno por uno, de arriba a abajo de izquierda a derecha, representados por los valores de intensidad del rojo, verde y azul.

Normalmente el valor máximo es 255. Si es más, se especifican en *big endian*. Así, en general, $\text{Offset} = (y \cdot \text{ancho} + x) \cdot 3 + \text{fin\_cabecera} + 1$.

2.6.2. HTML

El *HyperText Markup Language* se usa para crear páginas web. Se usan etiquetas con forma `<nombre attr="val" ...>...</nombre>`, como `<b>Hola</b>`, que indica que el texto está en negrita. Así:

- `<HTML>` envuelve a todo el documento.
- `<HEAD>` contiene la cabecera.
  - `<TITLE>` indica el título.
  - `<META NAME="AUTHOR" CONTENT="`(autor)`">`.
- `<BODY>` contiene al cuerpo del documento.
  - `<P>` indica un párrafo.
  - `<BR>` introduce una línea en blanco.
  - `<A HREF="`(URL)`">` muestra un enlace.

2.6.3. ODF

El formato *OpenDocument* es un formato estándar y libre para documentos, hojas de cálculo, gráficos, presentaciones... Es un archivo comprimido en ZIP que contiene varios ficheros y directorios, siendo los más importantes:

- `content.xml`: Almacena el contenido real del documento, salvo datos binarios como imágenes, y su formato es similar al HTML aunque bastante más complejo.
- `styles.xml`: Almacena los estilos para el formato y disposición del contenido, y existen varios tipos como los de carácter, párrafo, etc.
- `meta.xml`: Contiene metadatos como el autor, la última persona que lo modificó, fecha de la última modificación, etc.
- `settings.xml`: Incluye propiedades como el factor de zoom o la posición del cursor, pero no afectan al contenido.

Capítulo 3

Sistemas digitales: Circuitos combinacionales

En electrónica digital distinguimos dos niveles de tensión (alta y baja), que abstraemos con el sistema binario. Tipos de circuitos:

- **Combinacionales (sin memoria):** Las salidas solo dependen de las entradas actuales.
- **Secuenciales (con memoria):** Las salidas dependen de las entradas y un valor almacenado (**estado**).

El **álgebra de Boole** sirve para expresar circuitos lógicos. Tenemos las operaciones $\overline{A}$ (negación), $A+B$ (o) y $A \cdot B$ (y), y podemos expresar funciones de n variables $f : \{0,1\}^n \to \{0,1\}$ como $F(A,B,C) = \ldots$, o bien como tabla de verdad. Si a cada entrada (A,B,C) le asignamos su valor ABC_b, tenemos las **formas normalizadas:**

- **Suma de productos (minitérminos):** $F(A,B,C) = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + \ldots = m_2 + m_3 + \ldots = \sum m(2,3,\ldots)$. Tomamos las combinaciones en las que la salida es 1.
- **Producto de sumas (maxitérminos):** $F(A,B,C) = (A+B+C) \cdot (A+B+\overline{C}) \cdot \ldots = M_0 \cdot M_1 \cdot \ldots = \prod M(0,1,\ldots)$. Tomamos las combinaciones en las que la salida es 0 y negamos cada letra.

3.1. Mapas de Karnaugh

Representación gráfica de una tabla de verdad. Si la función tiene $m+n$ variables, realizamos una tabla $2^m \times 2^n$ y en las cabeceras de fila y columna ponemos los nombres de estas variables con sus posibles combinaciones, de forma que dos celdas adyacentes solo se diferencien en 1 byte. Ponemos 1, 0 o X (valor no determinado) en cada celda según el valor de salida para las variables. Un cuadrado tiene n cuadrados adyacentes (que se diferencien en solo una entrada), y estos se combinan en grupos de 2^k celdas con igual valor de salida eliminando k variables. Debemos intentar cubrir todos los unos (o ceros) en el menor número de grupos posible. Una celda puede estar en varios grupos.

Terminología:

- **Implicante:** Producto de variables.
- **Implicante primo:** Implicante no contenido en otro.
- **Implicante primo esencial:** Implicante primo con al menos un 1 cubierto solo por él.
- **Cubierta:** Conjunto de implicantes primos que cubren todos los unos.

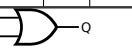
Al simplificar, las X pueden interpretarse a conveniencia como 1s o 0s.

3.2. Puertas lógicas

$A \cdot B$	0	1
0	0	0
1	0	1

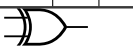
AND

$A + B$	0	1
0	0	1
1	1	1



OR

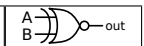
$A \oplus B$	0	1
0	0	1
1	1	0



XOR

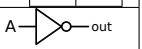
NAND

NOR



XNOR

$\overline{A}$:	0	1
	1	0



NOT

A veces la puerta NOT se representa simplemente con el círculo pequeño, pegado a otra puerta. Las puertas NAND, NOR y XNOR equivalen a una puerta AND, OR o XOR, respectivamente, seguida de una puerta NOT. La mayoría de circuitos actualmente se encuentran en chips (circuitos integrados) todo lo grandes o pequeños que queramos. Según el número de puertas lógicas:

- **SSI:** 1-10 puertas.
- **MSI:** 10-100 puertas.
- **LSI:** 100-100000 puertas.
- **VLSI:** >100000 puertas.

3.3. Retardo

Existe un **retardo** de algunos nanosegundos desde que cambia la señal a la entrada hasta que se estabiliza la señal de salida en el valor deseado. Podemos considerar que este es el máximo que debe «recorrer» una señal dentro del circuito una vez sabemos el retardo de cada puerta lógica.

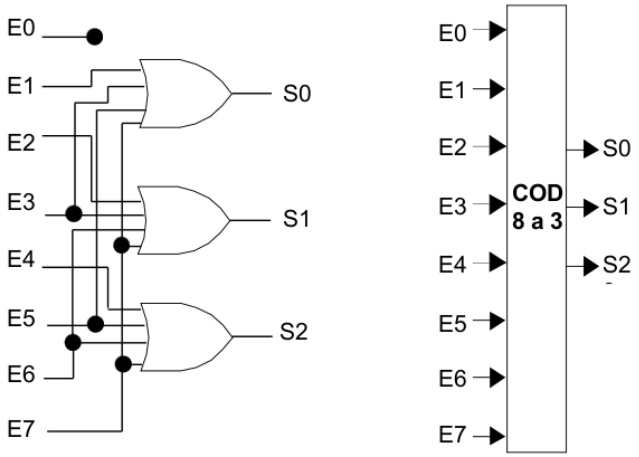


Figura 3.1: Codificador.

3.4. Implementación con puertas NAND/NOR

Para implementar una función en forma de suma de productos sólo con puertas NAND, simplificamos, negamos dos veces y aplicamos De Morgan una vez, obteniendo un resultado que se puede interpretar con puertas NAND. El proceso es el mismo para implementar una función en forma de producto de sumas sólo con puertas NOR.

3.5. Bloques lógicos

Son bloques que contienen una parte del circuito y se usan para simplificar la representación. Se representan como un cuadrado con flechas entrantes a un lado y salientes a otro con un indicador textual del bloque.

3.5.1. Codificador

Circuito con 2^n líneas de entrada y n de salida. Solo una línea de entrada se activa en cada momento y su nº se representa en binario en la salida.

3.5.2. Decodificador

Circuito con n líneas de entrada y 2^n de salida que activa la línea de salida cuyo nº corresponde a la entrada en binario.

Podemos implementar una función con un decodificador conectando las salidas correspondientes a un $F(\ldots) = 1$ como entrada de una puerta OR.

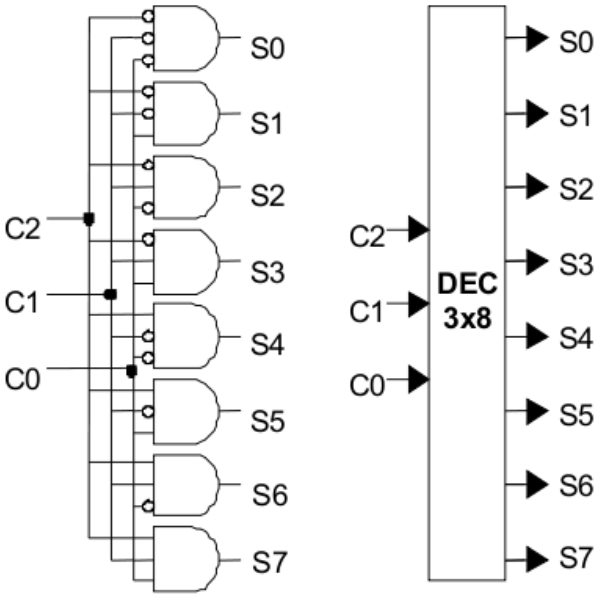


Figura 3.2: Decodificador.

3.5.3. Multiplexores

Circuito con 2^n líneas de entrada de datos, n de entrada de control y una de salida. Las líneas de control seleccionan qué entrada de datos pasa a la salida.

Podemos implementar funciones como multiplexores conectando a cada entrada de datos el valor de la salida correspondiente y conectando las entradas como entradas de control. Si el multiplexor tiene una entrada de control menos que la función, elegimos una variable que no se conecta como entrada de control, agrupamos en el mapa de Karnaugh las celdas que tienen el resto de entradas iguales y como entradas conectamos 0, 1 o la variable en cuestión, negada o no.

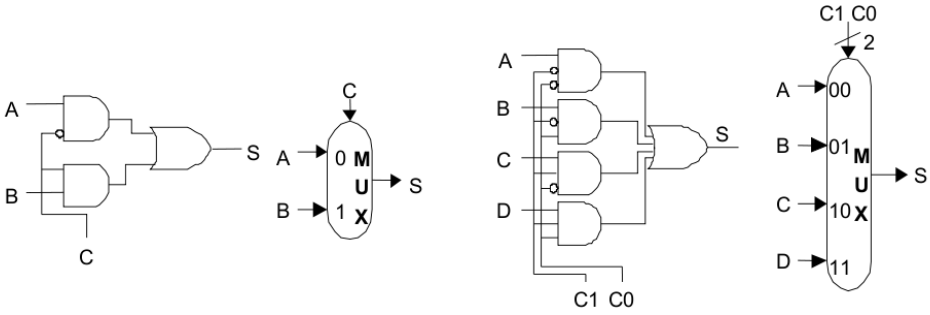


Figura 3.3: Multiplexores.

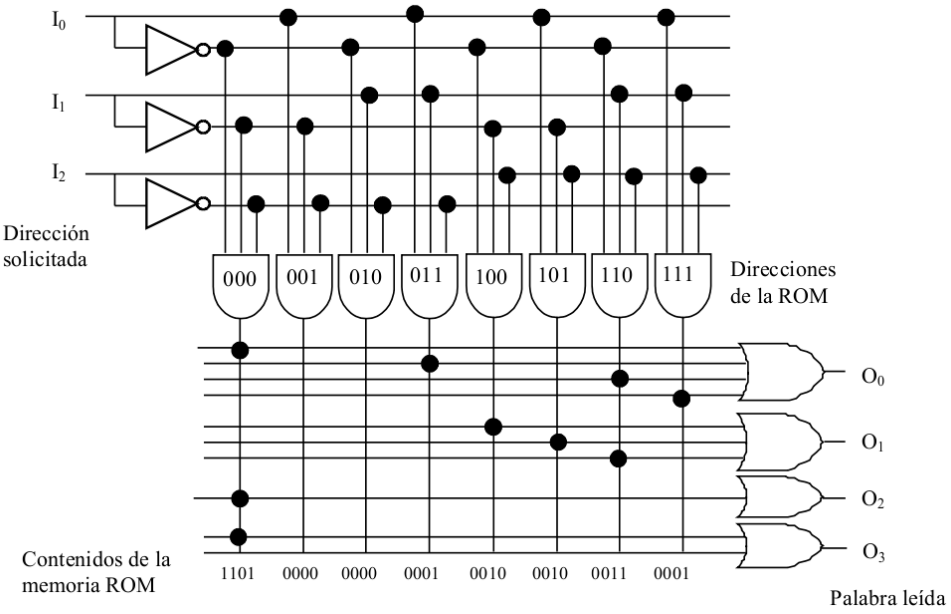


Figura 3.4: Memoria ROM.

3.5.4. Memorias ROM

Una **ROM** (*Read Only Memory*) es un circuito combinacional con m entradas y n salidas que almacena 2^m celdas (**altura** de la ROM) de n bits (**anchura** de la ROM). Se implementa con un plano AND con 2^m puertas de m entradas cada una y un plano OR con n puertas de salida:

Variantes:

- **PROM:** *Programmable ROM.*
- **EPROM:** *Erasable PROM.*
- **EEPROM:** *Electronically Erasable PROM.*
- Memorias flash (permiten borrado y reescritura por bloques, miles de veces).
- **PLA:** *Programmable Logic Array.* Como una ROM, pero solo se implementan los productos (puertas AND) necesarios. Útil cuando pocas combinaciones se usan realmente.

Capítulo 4

Introducción a los sistemas operativos

Un sistema operativo es una capa de software situada entre el hardware y las aplicaciones, que se encarga de «enmascarar» la complejidad del software a usuarios y programadores, administrando:

- La CPU, que se comparte entre los procesos y el núcleo del SO.
- La memoria, que también se comparte, impidiendo que un proceso acceda a la de otro indebidamente.
- Los dispositivos, que también se comparten, protegiendo de accesos indebidos y ofreciendo una interfaz uniforme a los distintos tipos de dispositivos.

Ejemplos: Windows (2000, XP, Vista...), Linux (Ubuntu, Fedora...), Unix, FreeBSD, MacOS, etc. Una distribución de un sistema operativo incluye al núcleo del SO junto con software adicional. Tipos de SO: De propósito general, de servidores, de tiempo real, integrados, de tarjeta inteligente, de supercomputadores, etc. Conceptos:

Usuario Persona que trabaja en el sistema.

Sesión Periodo de tiempo en el que un usuario interactúa con el sistema.

Programa Código ejecutable almacenado en disco. Concepto estático.

Proceso Programa en ejecución, que necesita recursos. Concepto dinámico con estado cambiante. Unidad de trabajo del SO.

Fichero Unidad lógica de almacenamiento de datos persistentes. Secuencia de bytes con un formato determinado.

Programas del sistema Programas que suelen acompañar al SO, como administrador de archivos, intérprete de comandos, programas para información de estado, aplicaciones básicas o utilidades de programación.

Interfaz de usuario Permite al usuario dar órdenes al sistema.

GUI (*Graphical User Interface*) Presenta una visión intuitiva del sistema. Se basa en un gestor de ventanas que permite arrancar y terminar aplicaciones y trabajar con varias al mismo tiempo.

Línea de comandos Órdenes tecleadas. Permite llamar a **órdenes internas** reconocidas por el intérprete y **programas externos**, en su propio ejecutable.

4.1. Funcionamiento de un SO

4.1.1. El arranque

Al encender, la CPU ejecuta un programa en ROM (**iniciador ROM** o **ROM BIOS**) que realiza un autodiagnóstico rápido del hardware y lee el **disco** el programa **cargador**, que posiblemente permita seleccionar entre varios SO. Este se encarga de cargar el kernel del SO en memoria, que toma el control, establece sus estructuras internas básicas (tabla de procesos, memoria, E/S, etc.) y ejecuta el **proceso inicial**, que empieza a lanzar procesos auxiliares y **demonios** (para impresión, red, etc.) según esté configurado y, finalmente, lanza uno (o varios) procesos de login, que permiten al usuario autenticarse y comenzar a trabajar.

4.1.2. Interrupciones

Las **interrupciones** son un mecanismo que permite pasar el control al núcleo del SO. Tipos:

- **Interrupciones software**, **llamadas al sistema** o **traps:** Las inicia un proceso para llamar a un servicio del sistema.
- **Interrupciones hardware:**
  - **Excepciones** o **desvíos:** Las produce un error en la ejecución, como una instrucción errónea, acceso indebido, error numérico, etc.
  - **Interrupciones** propiamente dichas: Las causa un evento externo como el reloj del sistema o un dispositivo de E/S, y llegan a la CPU mediante el bus de control.

Algunos tipos de llamadas al sistema:

- **Procesos:** Creación y terminación de procesos e hilos...
- **Acceso a dispositivos:** Apertura, cierre, lectura y escritura de ficheros y dispositivos de E/S...
- **Gestión de la memoria:** Solicitud y liberación de espacio...
- **Manipulación del sistema de ficheros:** Creación y borrado de ficheros y directorios, movimiento por directorios, manipulación de permisos, acceso a metadatos...
- **Otros:** Sincronización y comunicación entre procesos...

4.1.3. El subsistema de gestión de procesos

Los SO permiten la **multiprogramación**, el uso compartido de la CPU entre varios procesos, y de esta forma permite múltiples usuarios. Periódicamente, el reloj interrumpe al proceso en ejecución para ejecutar código del kernel, que cambia el **contexto** para ejecutar otro proceso. Los procesos van avanzando con sensación de simultaneidad, y los que quedan a la espera de E/S ceden la CPU a otro proceso. La parte del kernel encargada de optimizar el uso de CPU es el **planificador** o ***scheduler***.

Todos los procesos son creados por otro, llamado **proceso padre**, siendo la raíz de la jerarquía el proceso inicial (en Linux, `init`), y terminan de forma voluntaria o externa. Los procesos también pueden ser monitorizados, y se autorizan intentos de comunicación entre procesos.

4.1.4. El subsistema de gestión de memoria

La RAM es compartida por todos los procesos y el propio kernel, el cual aísla a unos procesos de otros ubicándolos en **espacios de direccionamiento** independientes. Cada proceso tiene un **espacio de direcciones virtual**, que es mapeado por el sistema (con ayuda del hardware) a distintas **direcciones físicas**. Si es necesario, parte del espacio virtual de un proceso se mantiene en disco. El SO se encarga de controlar las zonas de memoria libres y ocupadas, asignar y recuperar espacio y mover datos y código entre la RAM y el disco según sea necesario.

4.1.5. El subsistema de gestión de E/S

El SO oculta las particularidades de distintos tipos de dispositivos. Por ejemplo, mientras que el usuario ve un sistema de ficheros organizado jerárquicamente, el disco en el que este está almacenado, con el que interactúa el SO, se muestra como una inmensa tabla de sectores de 512 bytes cada uno.

Los dispositivos se manejan mediante una **controladora**, un CI en el propio dispositivo que lo controla físicamente y acepta comandos elementales, y un **manejador de dispositivo** o **«driver»**, un software ejecutado por el kernel del SO en **modo privilegiado** (uso no restringido del procesador) que se comunica con la controladora.

Las operaciones de E/S tardan un cierto tiempo en ejecutarse, por lo que al enviar un comando a la controladora, el SO suspende el proceso en ejecución y otorga la CPU a otro proceso, de forma que cuando termina la transferencia de datos, se envía una **interrupción** al procesador, causando que el SO vuelva a tomar el control y despierte al proceso bloqueado.

4.2. Linux

Es un clon de Unix creado por Linus Torvalds en 1991. Su código fuente está disponible bajo GPL (puede usarse, modificarse y distribuirse libremente). Es multiplataforma, pues funciona en gran cantidad de procesadores por estar escrito casi todo en C, multiusuario, multitarea y multinúcleo. Usa memoria virtual con espacios de direccionamiento diferentes, y soporta múltiples sistemas de archivos, protocolos de red e infinidad de dispositivos. Existen múltiples distribuciones y miles de aplicaciones disponibles, libres y comerciales.

Se accede al sistema mediante nombre de usuario y contraseña, y solo el usuario `root` tiene el control total. El *prompt* de la línea de comandos, configurable, proporciona información del usuario, la máquina y el directorio actual:

usuario`@`*máquina*`:`*directorio_actual*`$`

La interfaz gráfica más común es X-Window, que actúa como proceso «servidor» al que se conectan las aplicaciones gráficas «cliente». Su diseño en red permite ejecutar aplicaciones gráficas remotas siempre que en nuestra máquina se ejecute el servidor X. Este captura los eventos de teclado y ratón y los envía a la aplicación, a la vez que muestra la salida gráfica de la misma en ventanas. Se arranca con `startx`, aunque la mayoría de distribuciones ya lo ejecutan al inicio.

El principal cliente de X-Window es el gestor de ventanas, que determina la apariencia del escritorio y ventanas. En Linux son populares KDE y GNOME, que cuentan con aplicaciones de todo tipo las cuales son compatibles entre sí por funcionar con X, aunque existen muchos más, como otros más ligeros.

4.2.1. La línea de comandos

La consola de texto, terminal, intérprete, línea de comandos o *shell* de Linux es un programa denominado `bash`. Permite usar, configurar, personalizar y monitorizar el sistema de forma muy avanzada, y realizar tareas repetitivas mediante «guiones shell» o *scripts*, series de órdenes que se almacenan en un archivo para ser ejecutadas posteriormente con una sola orden.

Los caracteres `/|\!?*<>&~()[];#`, así como el espacio, tienen un significado especial en `bash`, por lo que no es conveniente usarlos en nombres de archivos y directorios.

Los comandos se especifican como «`<comando> [parámetros...]`». Los comandos que aceptan varias opciones (con -(*letra*)) permiten ponerlos separados por espacios o juntos, en cuyo caso solo se indica el guión en el primero.

El tabulador completa una ruta o una orden cuando ya hemos escrito suficientes caracteres para distinguirla. Las teclas arriba y abajo permiten navegar por el historial de órdenes, al que podemos acceder con `CTRL-R` y tecleando una subcadena, con el comando `history` o con `!`*`número_de_orden`*.

Para evitar la interpretación de cualquier caracter reservado, se introduce \ delante o se encierra todo (el parámetro o parte de él) entre comillas simples o dobles (la interpretación de estas, y del \, también pueden ser anuladas por \).

Para salir de la línea de comandos se usa el comando interno `exit`. Linux tiene una serie de terminales virtuales. Para acceder a una, se pulsa `CTRL-ALT-F`n, donde n es el número de la terminal. En una de ellas está la interfaz gráfica, en el caso de Ubuntu, en la 7. Para copiar y pegar en dichas terminales virtuales, se marca el texto a copiar de principio a fin pulsando el botón izquierdo del ratón y después, con el cursor situado donde se quiere pegar el texto, se pulsa el botón central.

4.2.2. Órdenes de ayuda

- `man` *`page`*: Muestra un manual de la orden que se le indica como parámetro, del que se sale pulsando `q`.
- `info` *`page`*: Misma función, aunque funciona de forma distinta.

- `help` *`command`*: Muestra algo de información sobre un comando interno.
- También se puede obtener información llamando al comando con `-h`, `-?` o `--help`.

4.2.3. El sistema de ficheros

Todo el almacenamiento se considera un sistema único de ficheros jerárquico que parte del directorio raíz (`/`) y cuyas entradas pueden ser ficheros (regulares, por caracteres, por bloques o enlaces simbólicos) u otros directorios.

- Los **enlaces simbólicos** son «punteros» a otras entradas del sistema de ficheros, que almacenan la ruta de estas. Si el archivo original cambia de lugar o es eliminado, el enlace queda «colgando», apuntado a nada.
- Los **enlaces físicos** o **duros** se diferencian en que el fichero creado originalmente es indistinguible del enlace, de forma que solo al borrar el último enlace se libera el espacio en disco. Sin embargo, no se puede crear un enlace físico a un fichero en otro disco.

Al indicar una ruta, se usa el caracter `/` para separar directorios y ficheros. El punto (`.`) en un nombre de archivo se usa opcionalmente para agrupar archivos que serán abiertos con la misma aplicación, diferenciando por **extensiones**, y si se pone al principio de un archivo o directorio, este cuenta como **oculto**.

Una ruta que empiece por `/` es **absoluta** y parte de la raíz. Cualquier otra ruta es **relativa** y parte del directorio actual. El directorio actual se denota por `.`, el directorio «padre» de otro por `..` y el directorio «home» (del usuario que ha iniciado sesión) por `~`. Los nombres de archivos diferencian mayúsculas de minúsculas.

Los directorios importantes son los siguientes:

- `/bin`: Programas básicos del sistema.
- `/usr/bin`: Aplicaciones y otros programas.
- `/sbin`: Programas de administración (para superusuario).
- `/lib`: Bibliotecas del sistema.
- `/usr/lib` Bibliotecas de aplicaciones.
- `/etc`: Ficheros de configuración.
- `/home`: Directorios de usuarios. En general, cada usuario tiene una carpeta `/home/<user>`.
- `/tmp`: Ficheros temporales.
- `/dev`: Dispositivos.
- `/proc`: Visión dinámica del sistema.

Los dos últimos directorios mencionados no existen en el disco, sino que el SO proporciona esta percepción por comodidad. A veces `/tmp` tampoco está en disco, sino que solo existe en la RAM.

Los discos «tal cual» se representan como archivos, que habitualmente se denominan `/dev/sda`, `/dev/sdb`, etc., y sus particiones `/dev/sda1`, `/dev/sda2`, etc., y `/dev/null` indica un archivo virtual que descarta todo lo que se escribe en él.

En `bash`, si un parámetro es una ruta, podemos usar comodines, y el parámetro se sustituye por la lista de rutas de archivo que cumplen la condición (si existen).

- `*`: Cero o más caracteres.
- `?`: Un caracter.
- `[`a-z`]`: Un caracter en el rango.
- `[!`a-z`]`: Un caracter que no esté en el rango.
- {$n1$,$n2$,...}: Cualquier secuencia de caracteres de la lista.

Comandos:

- `pwd`. Indica el directorio en que nos encontramos (ruta completa).
- `cd [-|dir]`. Cambia al directorio especificado (o a `~`). Si se indica `-`, vuelve al último directorio en que estuvimos.
- `ls [-dlrRaStu] [dir]`. Lista las entradas del directorio `dir` o del actual:
  - `-l`: Formato largo. En vez de mostrar solo el nombre, muestra el total y a continuación, para cada archivo:
    - Tipo (`-`: Archivo; `d`: Directorio; `l`: Enlace simbólico; `c`: Fichero por caracteres; `b`: Fichero por bloques) y permisos.
    - Número de enlaces duros.
    - Usuario.
    - Grupo.
    - Tamaño (bytes).
    - Fecha y hora de modificación.
    - Nombre.
  - `-R`: Lista recursiva. Muestra también el contenido de todos los subdirectorios.
  - `-a`: Muestra también las entradas ocultas (cuyo nombre empieza por `.`).
  - `-S`: Ordena por tamaño.
  - `-t`: Ordena por fecha y hora de última modificación.
  - `-u`: Ordena por fecha y hora de último acceso.
  - `-U`: Ordena por fecha y hora de creación.
  - `-r`: Invierte el orden escogido por `-S|t|u|U`.
  - `-d`: Lista la entrada correspondiente al subdirectorio indicado en lugar de las entradas que este contiene.
- `cat [file]`. Muestra el contenido del `file`, o de la entrada estándar.

- `more [file]`. Igual pero permite mostrarlo poco a poco.
- `less [file]`. Igual pero permite desplazarse arriba y abajo y buscar palabras con */palabra*.
- `hexdump -C [file]`: Muestra el contenido de `file`, o de la entrada estándar, en hexadecimal con formato.
- `find [path] [match] [expr]`. Busca en `path` (o en el directorio actual) y sus subdirectorios, archivos que cumplan los criterios (`match`) (o todos) y hace lo que indique la expresión (`expr`) (o muestra el nombre por la salida).
  - Criterios:
    - `-name` *nombre*: Nombre de la entrada (se pueden usar comodines, pero de forma que no los sustituya la *shell*).
    - `-iname` *nombre*: Similar pero sin distinguir mayúsculas y minúsculas.
    - `-user` *usuario*: Usuario al que pertenece.
    - `-group` *grupo*: Grupo al que pertenece.
    - `-type [c|b|d|l|f]`: Tipo de fichero (igual que en `ls -l`, salvo que `f` para fichero regular).
    - `-mtime [+|-]`n: Modificado hace más (`+`), menos (`-`) o exactamente n días.
    - `-atime [+|-]`n: Igual pero con el último acceso.
    - `-amin [+|-]`n: Igual que `-atime` pero en minutos.
    - `-size [+|-]`n: Tamaño de más (`+`), menos (`-`) o exactamente $512n$ bytes.
    - `!`: Negación de lo siguiente.
    - `-o`: Disyunción lógica.
    - `-a`: Conjunción lógica (por defecto cuando se usan varios criterios).
  - Expresiones:
    - `-printf` ...: Imprime la secuencia de caracteres que se indica. Secuencias de escape: `%s`: Tamaño; `%u`: Usuario; `%p`: Ruta completa; etc.
- `which` *command*. Muestra la ubicación en el sistema de ficheros de un comando externo. La lista de directorios donde la *shell* busca comandos externos se denomina `PATH` y se puede consultar con `echo $PATH`.
- `touch <file>`. Crea un fichero o, si ya existía, actualiza su fecha de modificación.
- `cp [-iprRu] srcfile ... destfile|destdir`. Copia un fichero a otro, o varios ficheros a un directorio.
  - `-R`, `-r`: Copia recursiva, para copiar directorios.
  - `-i`: Pide confirmación si el fichero de destino ya existía, para evitar sobreescribirlo.
  - `-p`: Conserva la fecha de modificación al copiar.
  - `-u`: No se copia si el destino tiene una fecha de modificación igual o posterior al origen.

- `mv srcfile|dirfile ... destfile|destdir`. Como `cp` pero cambia la posición del archivo en vez de copiarlo. Se puede usar para cambiar el nombre de un fichero o directorio.
- `rm [-ifrv] file|dir ...`. Elimina ficheros y directorios.
  - `-v`: Muestra los nombres de las entradas conforme se eliminan.
  - `-i`: Pide confirmación para cada entrada.
  - `-r`: Borrado recursivo, para eliminar directorios.
  - `-f`: Nunca pide confirmación.
- `mkdir dir`. Crea un directorio vacío en la ruta especificada.
- `rmdir dir`. Elimina el directorio indiciado si está vacío.
- `tar ...`. Para manipulación de archivos. Estos suelen tener la extensión `.tar` o `.tar.gz` (comprimido, también `.tgz`).
  - `tar c[zv][f archive] [files|dirs ...]`. Comprime los ficheros y directorios indicados.
  - `tar t[zv][f archive]`. Lista los contenidos del archivo.
  - `tar x[zv][f archive]`. Extrae el contenido del archivo, recuperando también los permisos, estructura de directorios, etc.
  - `z`: Indica compresión GZip (`.gz`).
  - `v`: Muestra los archivos conforme se comprimen o descomprimen.
  - `f archive`: Indica el nombre del archivo.
- `ln [-s] target link`. Crea un enlace llamado `link` al fichero `target`. Si `link` es un directorio existente, se crea dentro el enlace con el nombre de `target`.
  - `-s`: El enlace es simbólico (si no se especifica, se crea un enlace duro). En tal caso, la ruta del fichero `target` se indica de forma relativa al directorio en que está `link`.
- `df`. Informa del espacio total y libre en todos los sistemas de archivos montados.
- `du [-hs] dir ...`. Muestra lo que ocupan realmente en el disco los directorios indicados junto con todos sus ficheros y subdirectorios de forma recursiva.
  - `-h`: Muestra en unidades más legibles como KB, MB o GB, en vez de en bloques de 1024 bytes.
  - `-s`: Muestra sólo el tamaño total para cada argumento en lugar de mostrar lo que ocuparía cada entrada dentro de estos.

4.2.4. Usuarios, grupos y permisos

Existen tres tipos de **usuarios**: normales; del sistema, vinculados a ciertas tareas del SO, y `root` o **superusuario**, que tiene control total y en el prompt aparece con `#` en vez de `$`. La información sobre los usuarios se guarda en `/etc/passwd` y las contraseñas cifradas en `/etc/shadow`. Los usuarios se pueden organizar en **grupos**, con diferentes permisos. Cada usuario tiene un grupo principal, pero puede pertenecer a varios. La información se almacena en `/etc/group`.

Un usuario puede tener tres permisos sobre un fichero o directorio:

- r: Lectura.
- w: Escritura.
- x: Ejecución de un fichero o acceso al contenido de un directorio.

Cada entrada del sistema de ficheros lleva asociado un usuario y un grupo, y existen permisos distintos para el usuario, el grupo y «el resto». Se suelen indicar con tres secuencias `rwx` (para usuario, grupo y resto, en orden), de forma que para los permisos que no se concedan se sustituye la letra por un `-`. Así es como se indica en `ls -l`. Comandos:

- `chmod perms file|dir`. Cambia los permisos de un fichero o directorio. El campo `perms` indica los permisos en octal, asignando 1's a los permisos que se desea conceder y 0's al resto, en el orden en que se muestran en `ls -l`.
- `chown user file|dir`. Cambia el propietario de un fichero o directorio. Requiere permisos de superusuario.
- `chgrp group file|dir`. Igual pero con el grupo.
- `whoami`. Muestra nuestro nombre de usuario.
- `who`. Muestra los usuarios actualmente conectados, con sus horas y lugares (terminales) de inicio de sesión.
- `w`. Similar, pero además muestra qué está ejecutando cada usuario.
- `groups`. Muestra a que grupos pertenecemos.
- `sudo command pars ...`. Ejecuta el comando dado como superusuario (pide contraseña).
  - `-s`: Si se indica esto (sin especificar un comando), simplemente cambia a superusuario.
- `login user`. Cambia al usuario especificado (pide su contraseña).

4.2.5. Los procesos

Los comandos separados por `;` se ejecutan uno detrás de otro, y situar `&` detrás de un comando (o entre dos comandos, en cuyo caso afecta al de la izquierda) lanza un proceso en segundo plano, ejecutando lo que va delante sin esperar o permitiendo usar la shell sin que

termine de ejecutarse. `bash` muestra entonces el PID (identificador de proceso, entero único para cada uno). `CTRL-C` «mata» a un proceso en primer plano, y `CTRL-Z` lo pausa (lo «duerme»).

El directorio virtual `/proc` se consulta como un sistema de ficheros normal, pero realmente lo mantiene el núcleo en tiempo real, y contiene un subdirectorio por cada PID de proceso activo, con información sobre el mismo como ficheros abiertos, mapa de memoria, etc. También contiene:

- `/proc/cpuinfo`: Información sobre la CPU.
- `/proc/meminfo`: Información sobre la memoria.
- `/proc/version`: Versión del núcleo.
- `/proc/sys`: Directorio con los parámetros de distintos subsistemas del núcleo. Si se tienen los permisos, se puede hasta cambiar el comportamento del núcleo en tiempo de ejecución.
- `kill [-9] PID`: Lanza una señal para terminar el proceso con el PID indicado. Este puede «capturar» dicha señal evitando su finalización (puede ser útil en ciertos casos). Si aun así se desea terminar dicho proceso, la opción `-9` manda una señal «no capturable».

Cada proceso tiene una entrada estándar (`stdin`, por defecto el teclado), una salida estándar (`stdout`, por defecto la pantalla) y una salida estándar de error (`stderr`). En `bash`, detrás de un comando, `>` *archivo* redirige `stdout` a un archivo, que sobreescribe en caso de existir, y `>>` *archivo* hace lo mismo pero, si el archivo ya existe, en vez de sobreescribirlo añade la salida al final. Igualmente, `2>` y `2>>` hacen lo mismo pero con `stderr`, y $<$ *archivo* toma `stdin` de ese archivo.

Otra forma de comunicación entre procesos son las **tuberías** que permiten redireccionar el `stdout` de un fichero con el `stdin` del siguiente. En `bash` se indican con `/`. Existe un repertorio de comandos útiles, llamados **filtros**, especialmente diseñados para comunicarse mediante tuberías.

Comandos:

- `ps [-Af]`. Lista los procesos activos.
  - `-A`: Muestra todos los procesos en lugar de solo los lanzados desde ese terminal.
  - `-f`: Muestra información adicional de interes, como el PID, el consumo de CPU, el PID del proceso padre (PPID), hora de lanzamiento, etc.
- `top`. Monitoriza en tiempo real los procesos activos, mostrando información de ellos similar a `ps -Af`.
- `fg`: Pasa el último proceso pausado (despertándolo) o el último en ser iniciado en segundo plano al primer plano.
- `bg`: Despierta un proceso pausado pasándolo al segundo plano.

4.2.6. Otros comandos

- `date`. Devuelve la fecha y hora actuales.
- `cal`. Calendario del mes/año actual.
- `clear`. «Limpia» la pantalla del terminal.
- `reset`. Resetea el terminal (para cuando se queda con caracteres extraños, como tras hacer `cat` de un fichero binario).
- `expr`. Para cálculos aritméticos. Ejemplo: `expr 101 + 12`.
- `xargs ...`. Toma lo que se le pasa como parámetros (algún comando), le añade lo que le llega por la entrada estándar, lo interpreta como comando + parámetros y lo ejecuta.
- `split [-d] -b`n `[file [prefix]]`. Divide el archivo `file` (o la entrada estándar) en trozos de tamaño n (por defecto bytes salvo que le suceda una `k` (KB), etc.) cuyo nombre empieza por `prefix` (o `x`).
  - `-d`: Usa sufijos numéricos en vez de por letras.

Capítulo 5

Lenguajes del computador: alto nivel, ensamblador y máquina

Una **instrucción** es un conjunto de símbolos que representan una operación a realizar por la CPU, y un **programa** es un conjunto ordenado de instrucciones que debe ejecutar el computador sobre unos datos para procesarlos y obtener un resultado. Las instrucciones se almacenan en memoria principal y se ejecutan en secuencia, salvo por instrucciones de salto. Tipos de instrucciones:

- **De movimiento de datos** entre registros de la CPU y direcciones de memoria.
- **Aritmético-lógicas:** Suma, resta, multiplicación, división, AND, OR, desplazamientos, ... y operaciones de punto flotante.
- **Instrucciones de salto:** Condicionales, incondicionales y de manejo de subrutinas.

Las instrucciones se organizan en campos de bits, que indican, en un determinado formato, la operación a ejecutar, los operandos de entrada y el lugar donde dejar el resultado. Distintos tipos de instrucción utilizan distintos formatos, pues necesitan codificar información distinta.

La CPU está formada por un **camino de datos** (**CD**), encargado del procesamiento, y una **unidad de control** (**UC**), que decodifica las instrucciones y «controla» al camino de datos. En todo momento, la UC mantiene un **contador de programa** (**PC**, *program counter* o **IP**, *instruction pointer*) que contiene la dirección de la siguiente instrucción a ejecutar, y un **registro de instrucción** (**RI**), que contiene la instrucción a ejecutar. A mayor número de instrucciones mayor complejidad de la UC y más número de bits necesarios en el campo de código. Dos tendencias:

- **RISC** (***Reduced Instruction Set Computers***): pocas instrucciones, sencillas y que se ejecutan en pocos ciclos.
- **CISC** (***Complex Instruction Set Computers***): muchas instrucciones, complejas y que requieren muchos ciclos de reloj.

5.1. Jerarquía de traducción

Las instrucciones de la CPU, llamadas **instrucciones máquina**, se almacenan en binario. Programarlas directamente en **lenguaje** o **código máquina** es muy difícil y propenso a errores, por lo que los **lenguajes de programación** representan las instrucciones de forma simbólica. Existen principalmente dos tipos:

- **Lenguaje ensamblador:** Instrucciones representadas simbólicamente, en ASCII, que se corresponden directamente con instrucciones máquina y datos binarios.
- **Lenguajes de alto nivel:** Permiten expresar los programas de forma más cercana a la forma de pensar del programador, con variables, tipos de datos, funciones/procedimientos, condiciones, bucles, etc. Existen multitud de paradigmas (imperativo, orientado a objetos, funcional...) y de lenguajes (C, C++, Java, Haskell, ...). El lenguaje C, aun siendo de alto nivel, es más cercano a la máquina, y es el lenguaje nativo de UNIX/Linux.

El programa **compilador** traduce el código en un lenguaje de alto nivel a ensamblador, y el **ensamblador** convierte este código a un **fichero objeto**, que contiene:

- **Segmento de código (`.text`),** con instrucciones máquina.
- **Segmento de datos (`.data`),** con enteros, reales en punto flotante, cadenas de caracteres, etc.
- **Información de reubicación:** Para accesos a memoria, saltos, etc. Necesaria a la hora de unir los distintos ficheros objeto.

Una **biblioteca** (en los apuntes pone **librería** pero es una mala traducción) es un conjunto de módulos de código relacionados que pueden ser usados en distintos programas. Por ejemplo, en C existe una biblioteca estándar con funciones de E/S, funciones matemáticas, etc., pero también existen bibliotecas para cálculo matricial, gráficos en 3D, acceso a redes, etc. Hay dos tipos:

- **Estáticas:** El código se incluye dentro del fichero ejecutable final.
- **Dinámicas:** El código no se incluye en el ejecutable, sino que este almacena la información necesaria para cargar dicho código cuando va a ejecutarse. Estas pueden ser usadas por varios ejecutables al mismo tiempo usando el mismo espacio en memoria, y tampoco se desperdicia espacio en disco al no tener que copiarse en cada ejecutable que las use, pero al cambiar el ejecutable de máquina, este puede no funcionar por no tener una biblioteca necesaria.

El **enlazador** o ***linker*** une los distintos ficheros objeto generados junto con las funciones de bibliotecas estáticas utilizadas, para generar un **fichero ejecutable** final. Entonces el **cargador** o ***loader***, una parte del SO, lee este fichero del disco, lo ubica en memoria, realiza las transformaciones necesarias y le pasa el control.

5.2. Ensamblador de x86-64

Los archivos de código ensamblador están formados por segmentos como el segmento de datos, indicado por `.data`, y el segmento de código, indicado por `.text`. En cualquier punto se puede usar una **etiqueta** (identificador seguido de `:`) para representar una dirección de memoria y referirnos a ella en otra parte del código.

El segmento de datos contiene directivas como `.long` n, que indica un número de 32 bits, o `.string "..."` que genera una secuencia de caracteres acabada en un byte 0. El segmento de código contiene instrucciones en ensamblador. Sus operandos pueden ser:

- **Registros:** Son de acceso muy rápido, al estar en la propia CPU, y contienen valores intermedios de los cálculos. En x86-64, son de 64 bits, y son los siguientes:
  - De uso general: `RAX`, `RBX`, `RCX`, `RDX`, `R8`–`R15`.
  - Índices: `RSI`, `RDI`, para acceder a posiciones de una tabla.
  - Para la pila: `RSP` (puntero de pila), `RBP` (puntero base de pila).
  - Puntero de instrucción: `RIP`.
  - Registro de estado: `RFLAGS`, contiene información sobre el estado del procesador y el resultado de la ejecución de instrucciones, y afecta a los saltos condicionales.
  - También se puede trabajar con menos de 64 bits. Así, `EAX` son los 32 bits inferiores de `RAX`, `AX` los 16 bits inferiores de `EAX`, `AL` los 8 bits inferiores y `AH` los 8 bits superiores de `AX`.
  - En ensamblador, se representan con *`%nombre_en_minúsculas`*.
- **Memoria:** Hay hasta 2^{64} celdas de memoria direccionables de 1 byte. Realmente los programas se mueven en un espacio virtual de direcciones que el hardware transforma a direcciones físicas. En x86-64 se puede trabajar muchas veces directamente en memoria, sin pasar por registros, aunque el acceso es más lento. En ensamblador, para hacer referencia a una dirección de memoria, se usa *`dirección`*`(`*`%reg1`*`,`*`%reg2`*`,`*`potencia_de_2`*`)` para acceder a la celda $dirección + \%reg1 + \%reg2 \cdot potencia\_de\_2$, donde la potencia de 2 debe ser pequeña y todos los campos son opcionales. Abreviaturas: `(`*`%reg1`*`)`, *`dirección`*.
- **Constantes:** Números enteros directamente en el código, que en ensamblador se escriben `$`n. Se les suele llamar **inmediatos**.

A continuación vemos el **repertorio de instrucciones** o ISA (*Instruction Set Architecture*) de la arquitectura Intel x86-64, CISC, presente en los procesadores de los PCs de 64 bits. Los objetivos de un ISA son permitir que el diseño del procesador y del compilador sean sencillos, maximizar el rendimiento y minimizar el coste. En el caso de Intel, otro objetivo fue mantener la compatibilidad con procesadores anteriores, lo que llevó a soluciones menos elegantes y eficientes pero ayudó a mantener la cuota de mercado.

Algunos tipos de instrucciones:

1. **Aritmético-lógicas:** `add`, `sub`, `imul`, `and`, `or`, `xor`, ... *`(instr.)`* `src, dst`. $dst = dst \circ src$.
   a) **Incrementos y decrementos:** `inc`, `dec`. *`(instr.)`* `dst`. $dst = dst \pm 1$.

   b) **Desplazamiento de bits:** `shr` (despl. lógico a la der.), `shl` (despl. lógico a la izq.), `sar` (despl. aritmético a la der.), `sal` (despl. aritmético a la izq.) ... *`(instr.) n, dst`*.

2. **De movimiento de datos:** `mov` *`src, dst`*. *dst* ← *src*.

3. **Saltos incondicionales:** `jmp` *`pos`*. `RIP` ← *pos*.

4. **Saltos condicionales:** `je` (=), `jne` (≠), `jg` (>), `jge` (≥), `jl` (<), `jle` (≤), `ja` ($u >$), `jae` ($u \geq$), `jb` ($u <$), `jbe` ($u \leq$)... donde u en la notación signfica «sin signo». *`(instr.) pos`*. Si la instrucción anterior es `cmp` *`a, b`*, entonces esta hace `RIP` ← *pos* si y sólo si *bRa* (al revés de lo lógico).

La **pila** es una zona de memoria RAM gestionada como una estructura LIFO (*Last In First Out*) con dos operaciones posibles:

- **Apilar:** `push` *`src`*, guarda el contenido de un registro sobre la cima de la pila. Equivale a `sub $8,%rsp` + `mov` *`src`*`,(%rsp)`.
- **Desapilar:** `pop` *`dst`*, extrae lo que hay en la cima de la pila a un registro. Equivale a `mov (%rsp),`*`dst`* + `add $8,%rsp`.

Una **subrutina** es una secuencia de instrucciones que recibe (o no) unos parámetros, realiza alguna acción y devuelve (o no) un resultado al código que «llamó» a la subrutina. Para dar soporte a subrutinas se utiliza la pila, que se divide en **marcos de pila** o ***stack frames***, formados por:

- `8(%rbp)`: Dirección de retorno.
- `(%rbp)`:`%rbp` del marco de pila anterior.
- `-...(%rbp)`: Variables locales, valores de registros guardados (usados por la subrutina anterior).
- `+...(%rsp)`: Argumentos de salida para otras subrutinas (hasta 6 se pasan por registros).

Manejo implícito:

- `call` *`sub`*: Llama a una subrutina. Equivale a `push%rip` + `jmp` *`sub`*.
- `ret`: Vuelve de una subrutina. Equivale a `pop%rip`.
- `leave`: Desapila todo el marco de pila salvo la dirección de retorno y restablece `%rbp`. Equivale a `mov%rbp,%rsp` + `pop%rbp`.

Los mnemónicos vistos de instrucciones que admiten parámetros de distintos tamaños (todos salvo saltos y manejo implícito de subrutinas) realmente son versiones generales de otros que son iguales pero se le añade una letra detrás al nombre de la instrucción: `b` (byte) 8 bits, `w` (word) 16 bits, `l` (long) 32 bits y `q` (quad) 64 bits. Otros caso es el de, por ejemplo, `movslq` (*Move Signed Long to Quad*, mover entero con signo de 32 bits a 64).

5.3. Codificación de las instrucciones

Las instrucciones de x86-64 (CISC) son de longitud variable, mientras que las de MIPS 32 (RISC) son todas de 32 bits. En x86-64, el código de operación es el primer byte, en el que a veces se codifica el registro involucrado.

En un fichero objeto, el código se encuentra sin reubicar, de forma que los campos de las instrucciones correspondientes a las direcciones de memoria están vacíos o contienen direcciones relativas. Tras el enlazado y la carga, el programa se dice que se **reubica**, es decir, los campos de direcciones de memoria son modificados de acuerdo a la posición de la memoria (virtual) en la que se carga el programa. La memoria virtual es administrada por la **unidad de manejo de memoria** (**MMU**).

5.4. Herramientas de GNU

El comando `gcc` es el compilador GNU de C, el más utilizado en Linux. El comando `gcc main.c -o main` compila el archivo `main.c` y genera el ejecutable `main`. Para generar sólo el código en ensamblador, usamos `gcc main.c -fno-asynchronous-unwind-tables -S -o main.s`, que genera el archivo `main.s` en ensamblador, donde `-S` indica salida en ensamblador y `-fno-asynchronous-unwind-tables` sirve para generar un código bastante más limpio que es más útil para su estudio.

La opción `-c` indica salida como fichero objeto (sin enlazarlo). Si hacemos `gcc main.c -c -o main.o` y luego `objdump -d main.o`, nos aparece una versión desensamblada del fichero objeto en el que podemos ver cómo este se organiza. A continuación podemos enlazar el fichero objeto con `gcc main.o -o main`. Entonces podemos usar `ldd main` para ver la lista de bibliotecas dinámicas que necesita el programa y la dirección virtual del programa a la que estas son mapeadas. En particular, un programa sencillo necesita `libc`, la biblioteca estándar de C, y `linux-vdso` y `linux-x86-64`, que se corresponden con las llamadas al sistema de Linux. La opción `-static` de `gcc` (`gcc -static main.o -o main`) enlaza solo con bibliotecas estáticas (lo que aumenta considerablemente el tamaño).

La opción `-g` añade al ejecutable la información necesaria para depurar el programa (es decir, cargarlo en memoria, reubicado, y ejecutarlo de manera controlada). Entonces podemos usar `gdb`, el depurador de GNU, llamándolo con `gdb main`. Este tiene su propio intérprete, en el cual:

- `l` lista el código original.
- `disassemble` *sub* lista el desensamblado de la subrutina *sub*.
- `x/`*n*`bx` *tag* muestra los *n* primeros bytes a partir de la etiqueta *tag*.
- `x/`*n*`w` *tag* hace lo mismo pero con los *n* primeros *words* (grupos de 4 bytes).
- `b` *n* introduce un punto de interrupción (*breakpoint*) en la línea *n* del programa.
- `r` ejecuta el programa.
- `p` *var* muestra el valor de una variable.
- `c` continúa la ejecución del programa por donde se dejó.

Capítulo 6

Introducción a las redes de ordenadores

Internet es una red compuesta de millones de dispositivos, conocidos como **hosts** o sistemas finales, que se conectan mediante distintos tipos de enlaces con equipos de interconexión. Un ejemplo de red es la RedIris, que conecta universidades y centros de investigación y desarrollo españoles. Los **hosts** están en el extremo de la red, y pueden ser ordenadores, PDAs, teléfonos móviles, sensores, etc.

La red se organiza por **protocolos**, que definen el formato y orden de mensajes enviados y recibidos entre entidades y las acciones realizadas al enviar o recibir dichos mensajes. Estos se organizan en una arquitectura por **capas**, en las que cada capa realiza un conjunto de tareas relacionadas, proporcionando servicios a la capa superior y usando los de la capa inferior. Las entidades en la misma capa pero distintos hosts se llaman **procesos pares**, y pueden comunicarse mediante unos protocolos. Llamamos **arquitectura de red** al conjunto de capas y de protocolos usados en cada una. Las capas más importantes, de abajo a arriba, son:

1. **Física:** Transmisión de bits sobre el medio.

2. **Enlace:** Transferencia de datos entre elementos conectados directamente. Protocolos como **PPP** (*Point to Point Protocol*), **Ethernet** (IEEE 802.3), **WiFi** (IEEE 802.11) o **HLDC** (*High-Level Data link Control*).

3. **Red:** Encaminamiento de paquetes del origen al destino. Protocolo **IP** (*Internet Protocol*).

4. **Transporte:** Transferencia de información entre procesos. Proporciona una forma de distinguir aplicaciones dentro de una misma máquina. Protocolos **TCP** (*Transmission Control Protocol*) y **UDP** (*User Datagram Protocol*).

5. **Aplicación:** Transferencia de archivos, e-mail, web... Protocolos como **FTP** (*File Transfer Protocol*), **HTTP** (*HyperText Transfer Protocol*), **SMTP** (*Simple Mail Transfer Protocol*), **POP3** (*Post Office Protocol*), **BitTorrent**...

6.1. Medios de transmisión

- **Par trenzado:** Dos hilos de cobre dispuestos de forma helicoidal y cubiertos por un aislante de plástico. Permite frecuencias de hasta 250 MHz.
- **Cable coaxial:** Núcleo de cobre recubierto por un aislante envuelto a su vez en un conductor externo. Permite mayor distancia de transmisión e inmunidad al ruido, con frecuencias de hasta 900 MHz.
- **Fibra óptica:** Núcleo de fibra de vidrio (mayor densidad) recubierto de cristal o plástico (menor densidad). La presencia o ausencia de luz codifica un bit. Permite una mayor velocidad de transmisión (varios GHz y por tanto varios Gbps), distancia e inmunidad al ruido, además de tener menor coste.

6.2. Redes de acceso

Para acceso residencial:

- **Acceso vía módem:** Hasta 56 kbps de ancho de banda, y no se puede hablar por teléfono mientras se usa.
- **DSL** (*Digital Suscriber Line*): Se basa en una línea dedicada hasta la central telefónica, permitiendo combinar datos y voz.
- **ADSL** (*Asymmetric DSL*): Modalidad donde la velocidad de bajada y la de subida son distintas, con hasta $3,5$ Mbps en la línea ascendente (hacia Internet) y 24 Mbps en la descendente (hacia el host).
- **DOCSIS** (*Data Over Cable Service Interface Specification*) sobre **HFC** (*Hybrid Fibre Coaxial*). Une el hogar al router del ISP mediante cable y fibra, y al igual que ADSL, es asimétrico. El enlace hacia el router es compartido entre hogares, y estas redes son desplegadas por compañías de cable y TV.

Suele haber un módem ADSL o de cable, un router (normalmente con cortafuegos y NAT), una red Ethernet y un punto de acceso WiFi, si bien varios de estos elementos suelen agruparse. Para acceso institucional, suele haber una red de área local conocida como **intranet**, conectada a un router que conecta a Internet. En ella se usa:

- **Ethernet:** De 10 o 100 Mbps o 1 o 10 Gbps. Suele usarse sobre par trenzado, y los hosts suelen conectarse con conmutadores.
- **WiFi:** 802.11g (54 Mbps), 802.11n (600 Mbps) u 802.11ac (1 Gbps).

Un ejemplo de red es la RedIris (red académica española), que comunica las universidades y centros de I+D españoles.

6.3. Direccionamiento IP

En IPv4 (versión 4), cada interfaz de red tiene asignada una dirección IP de 32 bits, que se expresa como 4 números entre 0 y 255 separados por puntos.

Internet lo forma una serie de subredes interconectadas, y cuando un host desea enviar información a otro, lo que hace mediante **paquetes** IP, esta pasa normalmente a través de una serie de routers interconectados, que deben decidir el siguiente router por el que enviarlo. Hay dos formas de trabajar:

- **Conmutación de circuitos:** Se establece un circuito por cada conexión, permitiendo la entrega en orden y con calidad e servicio.
- **Conmutación de paquetes:** Cada paquete se encamina por separado, por lo que la entrega puede ser fuera de orden y no garantiza la calidad.

Cualquier modalidad requiere **algoritmos de encaminamiento**.

En **IPv4** (versión 4), cada interfaz de red tiene asignada una dirección IP de 32 bits, que suele expresarse como 4 números de 0–255 separados por puntos. Así, los routers suelen tener varias interfaces de red, mientras que los hosts solo suelen tener una. Una dirección IP se compone de dos partes:

- **Dirección de red** («netid»): Los n bits más significativos. El tamaño es variable, y una dirección IP de la forma $xxx.xxx.xxx.xxx/nn$ indica que la dirección de red ocupa nn bits.
- **Dirección de host** («hostid»): El resto de bits.

Anteriormente existían 4 tipos de redes (A, B, C y D) dependiendo del tamaño de la dirección de red, pero actualmente existe el **CIDR** (*Classless InterDomain Routing*) que permite que el tamaño de la dirección de red sea arbitrario. La **máscara de red** es un número (con forma de dirección IP) que se forma tomando un netid formado solo por 1's y un hostid formado solo por 0's.

El organismo encargado de asignar direcciones de red es el **ICANN** (*Internet Corporation for Assigned Names and Numbers*). Cuando dos hosts están en la misma red, pueden comunicarse directamente. En otro caso, el host origen envía el paquete al **router** por defecto (cuya dirección IP debe saber), que actúa como nexo de unión, y la dirección IP del paquete es la del host destino. Direcciones especiales:

- **Dirección de red:** El hostid está formado solo por 0's.
- **Dirección de *broadcast*** (difusión): El hostid está formado solo por 1's, y el paquete llega a todos los miembros de la red.

DHCP (*Dynamic Host Configuration Protocol*) permite configurar hosts de forma dinámica, de forma que el router asigna al host una dirección IP, máscara de subred y dirección del router por defecto. Además, la mayoría de routers pueden reenviar las solicitudes de configuración DHCP, por lo que no es necesario tener servidores DHCP en cada subred. Esquema general:

1. **Descubrimiento:** El host envía una solicitud de descubrimiento DHCP por su interfaz.

2. **Oferta:** El router responde informando de su IP y ofreciendo otra, con fecha de caducidad pero renovable.
3. **Solicitud:** El host solicita usar dicha IP.
4. **Confirmación:** El router confirma que el host pueda usarla.

La limitación en el rango de direcciones IP ha llevado a la creación del protocolo **IPv6** (IP versión 6), que permite asignar 2^{128} direcciones, reduce el tamaño de las tablas de enrutamiento, simplifica el protocolo para permitir el procesamiento más rápido de paquetes, proporciona seguridad al incluir cabeceras de autenticación y confidencialidad, presta más atención al tipo de servicio (especialmente al tiempo real), posibilita que un host sea móvil sin cambiar su dirección y elimina la sobrecarga del NAT al haber direcciones suficientes.

6.4. TCP y UDP

Las distintas capas de una arquitectura pueden ofrecer distintos tipos de servicio:

- **Servicio orientado a conexión:** Hay un establecimiento de conexión, una fase de transmisión de datos y una liberación de la conexión, de forma que los datos se entregan en orden.
- **Servicio no orientado a conexión:** Cada mensaje se procesa de forma independiente, incluyendo en cada uno la información de direccionamiento, y no se garantiza la entrega en orden.
- **Servicio confirmado:** El emisor tiene constancia de la recepción correcta de los datos.
- **Servicio no confirmado:** No hay tal confirmación.

El término **QoS** (*Quality of Service*) se refiere a las tecnologías que garantizan la transmisión de datos en un tiempo dado, y es importante para vídeo y voz. En la capa de transporte se tienen dos protocolos:

- **TCP:** Orientado a conexión.
- **UDP:** No orientado a conexión, para cuando prima el tiempo de llegada de los datos sobre la posible existencia de fallos en la transmisión.

Estos servicios permiten además definir un **puerto** de origen y uno de destino, que identifican a los distintos procesos existentes en los hosts que se comunican.

Por otro lado, dada la limitación en el rango de direcciones IP, y que normalmente en entornos domésticos solo se asigna una IP pública, en estos se suele usar **NAT** (Network Address Translation). Los hosts usan direcciones IP en una serie de **rangos de direcciones privadas** definidos, que para el exterior se traducen en una única IP pública, pues el router se encarga de la conversión.

Cuando un host envía información (por TCP o UDP) a una dirección IP pública, lo hace con un puerto de origen y uno de destino. El router entonces hace corresponder un puerto de su IP pública al conjunto de la IP privada y el puerto de origen, y envía entonces la información a la IP y el puerto de destino desde su propia IP pública y el puerto que ha asignado. Cuando recibe la respuesta, le llega con el puerto asignado como puerto de destino, de forma que puede redirigir la información a la IP privada y puerto correspondientes.

6.5. Capa de aplicación

Existen dos modelos mediante los cuales los hosts pueden conectarse en la capa de aplicación:

- **Modelo cliente-servidor:** Un servidor, siempre conectado, posiblemente replicado y con dirección IP fija, atiende solicitudes de los clientes, que no se comunican entre sí directamente y que posiblemente tengan direcciones IP dinámicas.
- ***Peer to peer* (P2P):** Los *peers* se comunican entre sí directamente. Pueden estar conectados intermitentemente y tener direcciones IP dinámicas. Este es un esquema muy escalable, pero difícil de gestionar.

BitTorrent es un sistema de distribución de ficheros P2P. Está formado por una serie de ***trackers***, que registran a los *peers* que participan en un *torrent*, y ***torrents***, grupos de *peers* que intercambian partes de un fichero.

El protocolo **DNS** (*Domain Name System*) es un servicio de traducción de nombres de host a direcciones IP, implementada como una base de datos distribuida mediante una jerarquía de servidores DNS y un protocolo de consulta a nivel de aplicación sobre UDP en el puerto 53. Los hosts tienen configurado un servidor DNS primario, al que realizan las consultas, y si no obtienen respuesta consultan al servidor DNS secundario. Este servicio también permite realizar traducciones inversas, gestionar alias (distintos nombres para la misma máquina) y otras acciones relacionadas con el correo electrónico o balanceo de carga.

FTP (*File Transfer Protocol*) es un protocolo cliente-servidor utilizado para transferencia de archivos.

La ***World Wide Web*** es el conjunto de servidores web de todo el mundo. Una página web está formada por objetos como páginas HTML, imágenes, *applets* Java, archivos de audio... en la cual una página HTML base incluye referencias a otros objetos. Cada objeto está identificado por una **URL** (*Uniform Resource Locator*), cuya sintaxis básica es: *protocolo*`://`*nombre_de_host*`/`*ruta_de_objeto*`.`

Se usa el protocolo **HTTP** (*HyperText Transfer Protocol*), en el que el cliente solicita al servidor objetos web, los recibe y los muestra. Se basa en TCP, y habitualmente el servidor escucha en el puerto 80. Existen dos tipos de mensajes HTTP: *request* y *response*, basados principalmente en ASCII. Ejemplo:

HTTP *request*

`GET /somedir/page.html`
`HTTP/1.1` (línea de solicitud: GET, POST...)
`Host: www.someschool.edu`
`User-agent: Mozilla/4.0`
`Connection: close`
`Accept-language: fr`
(cabecera)

HTTP *response*

`HTTP/1.1 200 OK` (línea de estado)
`Connection close`
`Date: Thu, 06 Aug 2008 12:00:15 GMT`
`Server: Apache/1.3.0 (Unix)`
`Last-Modified: Mon, 22 Jun 2008`
`ETag: "7ec0a8-141-4912c4c727280"`
`Accept-Ranges: bytes`
`Content-Length: 321`
`Content-Type: text/html; charset=es_ES.UTF-8`
(cabecera)

... (datos solicitados)

El **correo electrónico** funciona mediante los siguientes componentes:

- **Agentes de usuario:** Lectores de correo, usados por los clientes para editar, enviar y recibir mensajes. Eudora, Outlook, elm, Mozilla Thunderbird...
- **Servidores de correo:** Mantienen los mensajes que llegan a los usuarios (anteriormente solo los que no se habían enviado ya a estos), y también se encargan del envío.
- Protocolos:
  - **SMTP** (*Simple Mail Transfer Protocol*): Para el envío de mensajes, tanto desde los agentes como entre servidores.
  - **POP** (*Post Office Protocol*): El agente se conecta con el servidor, se autentica y descarga los mensajes.
  - **IMAP** (*Internet Message Access Protocol*): Más complejo que POP, permite manipular mensajes sobre el servidor.
  - **HTTP:** La mayoría de servidores de correo tienen una interfaz web.

Así, si Alicia quiere mandar un mensaje a Bob, su agente envía un mensaje a su servidor de correo con SMTP, y este queda a la cola para ser enviado. Entonces el servidor de Alicia manda el mensaje por SMTP al de Bob, que lo coloca en su buzón, de forma que Bob puede acceder mediante su agente con POP o IMAP.

6.6. Comandos de Linux

- `ifconfig`: «Interface configure». Para configurar la interfaz `eth0` de un host con dirección `192.168.0.21/24`, `sudo ifconfig eth0 192.168.0.21 netmask 255.255.255.0 up` o `sudo ifconfig eth0 192.168.0.21/24 up` (realmente no hace falta el `up` porque es la opción por defecto). Para apagarla, `sudo ifconfig eth0 down`.
- `route`: Para establecer el router por defecto, `sudo route add default gw 192.168.0.1 eth0`. Para visualizar las tablas de enrutamiento de nuestro sistema, `route -n`.
- `dhclient` *`interfaz`*: Configura una interfaz de red con DHCP.
- `ping` *`servidor`*: Envía paquetes a un servidor esperando a que responda, como herramienta de diagnóstico.
- `host`: Para obtener la IP correspondiente a un nombre de host, junto con posible información como alias, `host` *`nombre_host`*. Para realizar una consulta inversa, `host` *`dir_IP`*.
- `nm-tool`: Muestra la lista de servidores DNS configurados.
- `telnet` *`servidor`* `[`*`puerto`*`]`: Abre conexiones TCP.
- `netstat`: `-net` muestra todas las conexiones TCP abiertas; `-a` muestra todas las conexiones y puertos TCP y UDP incluyendo las que están «en escucha», y `-n` muestra los puertos con su identificación numérica y no de texto.