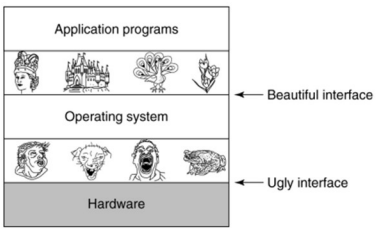
TEMA 1: INTRODUCCIÓN

- Una computadora es un **sistema complejo con muchos componentes** (memoria principal, discos, dispositivos de E/S, etc). La tarea de administrarlos y utilizarlos de manera óptima es difícil, por eso, el **SO** se ocupa de proporcionar un **modelo de computadora más fácil** de utilizar y de **administrar** todos estos **recursos**.

¿QUÉ ES UN SISTEMA OPERATIVO?

- EL PROGRAMA DE INTERFAZ DE USUARIO es el programa con el que los **usuarios** interactúan normalmente.
  - Se denomina SHELL cuando está basado en **texto** o GUI (Graphical User Interface) cuando utiliza **elementos gráficos** o iconos.
  - ↳ En realidad, **no forma parte del SO**, si no que es el nivel más bajo del software de **modo usuario**.
  - ↳ El SO lo utiliza para realizar su trabajo, permitiendo la **ejecución de programas**.
- EL SISTEMA OPERATIVO es un programa (por tanto, puramente compuesto por software) cuyas **funciones** son:
  - Proporciona a los **programadores y programas** un conjunto **abstracto** de recursos simples:
    - ↳ Los verdaderos **clientes** del SO son los programas y programadores, que interactúan con las **abstracciones** que les presenta el SO.
    - ↳ Los **usuarios finales** del SO interactúan con la **interfaz de usuario**, que es otro tipo de abstracción que no tiene que coincidir con la proporcionada a los clientes.
      - Así, el sistema operativo **oculta el hardware feo con interfaces hermosas**.
  - Administra los **recursos de hardware**:
    - Proporciona una **asignación** ordenada de los recursos a los programas.
    - Resuelve de **conflictos** entre programas o usuarios.
    - Se encarga de la **compartición** de recursos (temporal o espacial).

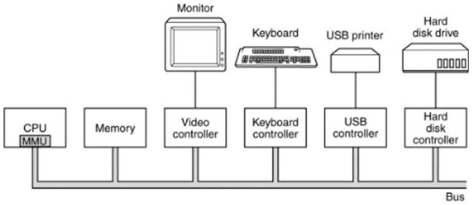


MODOS DE EJECUCIÓN DE SOFTWARE

- MODO NÚCLEO/KERNEL → el software tiene **acceso completo** a todo el **hardware** y puede ejecutar **todas las instrucciones máquina** que sea posible.
  - Es como se ejecuta el SO.
- MODO USUARIO → el software sólo tiene acceso a un **subconjunto** limitado de **instrucciones máquina**. Están prohibidas aquellas instrucciones máquina que afectan a la **protección de la memoria** y las que se encargan de la **E/S**.
  - Para acceder a servicios del SO, el programa de usuario debe realizar una **llamada al sistema** (syscall).
    - Las syscalls son TRAPS → instrucciones que cambian del modo usuario al modo kernel y le dan el control al SO.
      - ↳ Aparte de los syscalls, la mayoría de los demás traps son producidos por el hardware para advertir acerca de una situación excepcional.

REVISIÓN DEL HARDWARE

- El SO está **íntimamente ligado** al hardware de la computadora, por lo que para trabajar debe conocerlo muy bien.



EL PROCESADOR

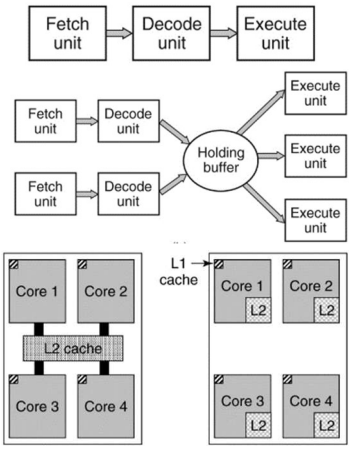
- El PROCESADOR o CPU se encarga de obtener instrucciones de memoria y ejecutarlas.
  - ↳ Cada CPU tiene un **repertorio de instrucciones** ensamblador que puede ejecutar.
  - El **ciclo** básico de ejecución de instrucciones (1 instrucción por ciclo de reloj) consiste en:
    1. **Obtener** instrucción de memoria.
    2. **Decodificar** instrucción.
    3. **Ejecutar** instrucción.
    4. **Repetir** para cada instrucción del programa.
- El **SO** debe controlar el **estado** del procesador y administrar su **uso**.

REGISTROS

- Las CPUs contienen varios REGISTROS en su interior en los que almacenan { variables clave, resultados temporales }
  - ↳ Si se accediese a memoria cada vez que se necesitasen se usaría mucho tiempo.
- Además de estos, hay otros **registros especiales** { Program Counter (**PC**). Stack Pointer (**SP**). Program Status Word (**PSW**) → contiene un bit que controla si se está en modo kernel o usuario. }

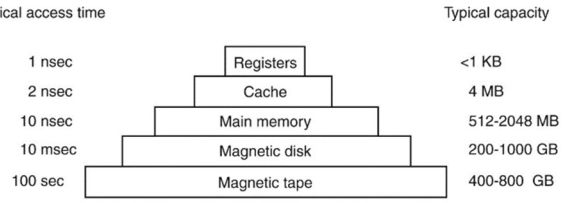
EJECUCIÓN DE MÁS DE UNA INSTRUCCIÓN POR CICLO DE RELOJ

- CANALIZACIÓN → CPU con **unidades separadas** para la obtención, decodificación y ejecución de instrucciones, para poder ejecutar **distintas etapas** de **distintas instrucciones** en un mismo instante.
- CPU SUPERESCALAR → CPU con **canalización** y **varias unidades de ejecución** que llevan a cabo diferentes labores
  - ↳ Varias instrucciones se obtienen y decodifican a la vez. Luego se vacían en un **búfer de contención** hasta que se puedan ejecutar. Cuando alguna unidad de ejecución queda libre, busca en el búfer para ver si contiene alguna instrucción que pueda manejar y la ejecuta.
- SISTEMAS MULTIHILAMIENTO → la CPU puede contener el estado de **varios hilos de ejecución** distintos y alternar entre ambos rápidamente.
  - Cada hilo aparece para el SO como una CPU separada denominada CPU VIRTUAL.
  - ↳ **No** ofrece un **paralelismo** real. En cada instante sólo hay un proceso en ejecución, pero este cambia muy rápido.
- SISTEMAS MULTICORE → sistemas con varias CPUs completas en su interior.
  - ↳ Estas CPUs pueden compartir **cache** o tener una cache para cada core.



LA MEMORIA

- La memoria ideal sería **rápida, grande y barata**. Como no existe ninguna tecnología que cumpla estos objetivos, el sistema de memoria se basa en una **jerarquía de capas**.
  - Las capas superiores tienen mayor velocidad, pero menor capacidad y más costo.



REGISTROS

- Son igual de rápidos que la CPU, así que **no hay ningún retraso** a la hora de usarlos.
- Tienen una **capacidad** de almacenamiento de $32 * 32$ o $64 * 64$, es decir, inferior a 1KB.

MEMORIA CACHÉ

- Se divide en **líneas caché** de un tamaño determinado.
- Cuando el programa necesita leer una palabra de memoria, la caché comprueba si la línea que se requiere está en caché:
  - Si está en la caché → **acierto de caché**, no hay que solicitar la palabra a la memoria principal.
  - Si no está en la caché → **fallo de caché**, hay que solicitar la palabra a la memoria principal, lo cual conlleva una **penalización temporal** grande.
- Puede haber varios **niveles** de caché.
  - Los niveles superiores son más próximos a la CPU y más rápidos, pero también más pequeños.

MEMORIA PRINCIPAL o RAM

- Memoria de acceso aleatorio (RAM) **volátil**.
- Todos los **fallos de la caché** se solucionan accediendo a la memoria principal.

MEMORIA ROM

- Memoria de acceso aleatorio **no volátil** cuyo contenido se **programa en fábrica** y **no** se puede **modificar** después.
- Es **rápida** y **económica**.

DISCO

- Es muchísimo **más grande** y **barato** que la RAM, pero también mucho más **lento** (pues es un **dispositivo mecánico**).
- Es **no volátil**, sirve para datos durante largos periodos de tiempo.
- Consiste en uno o varios **platos,** en los que se escribe la información en una serie de **círculos concéntricos,** que giran bajo **cabezas** conectadas a un **brazo mecánico**.
  - PISTA → región anular que puede leer cada una de las cabezas en una posición del brazo.
  - CILINDRO → conjunto de pistas que leen todas las cabezas para una posición dada del brazo.
  - SECTOR → cada una de las regiones, de tamaño fijo, que forman una pista. Los cilindros exteriores tienen más regiones que los interiores.

LOS DISPOSITIVOS DE ENTRADA Y SALIDA

- Constan de dos partes: un **dispositivo controlador/controladora** y el **dispositivo en sí**.
  - La CONTROLADORA es un chip o conjunto de chips que controlan físicamente el dispositivo.
    - A menudo son pequeñas **computadoras incrustadas** que se programan para hacer eso.

DRIVERS

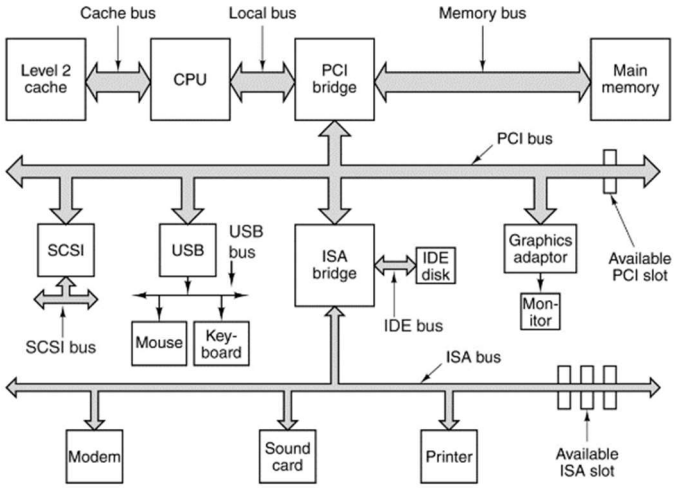
- El DRIVER es el software que se comunica con la controladora.
  - Hay un driver distinto para cada **tipo de controladora**.
- Para poder usarlo, debe colocarse en el SO de manera que se pueda ejecutar en **modo kernel**.
  - Se pueden cargar de forma **dinámica** → se coloca el driver en el kernel haciendo que el SO lo acepte mientras lo ejecuta e instala al instante.

REGISTROS DE LAS CONTROLADORAS

- Todas las controladoras tienen algunos **registros,** que sirven para comunicarse con ellas.
- Para **activar** la controladora, el driver recibe un comando del SO y lo traduce en los valores apropiados para escribir en los registros de la controladora.
- La **escritura** y **lectura** en estos registros se puede realizar de dos maneras (depende de la computadora):

EL BUS

- Consiste en un **conjunto de cables** eléctricos que conectan los distintos dispositivos de la **computadora**.
- Antes había **un único bus** que conectaba todos los dispositivos, pero esto acabó siendo inmanejable.
  Ahora hay **8 buses** (caché, local, memoria, PCI, SCSI, USB, IDE e ISA) y cada uno tiene una **velocidad de transferencia** y **función distintas**.



ARRANQUE DEL ORDENADOR

- La BIOS (Basic Input Output System) es un programa que se encuentra en una **RAM flash no volátil**.
  - Contiene **software de E/S de bajo nivel**, lo necesario para leer el teclado, escribir por pantalla, etc.

1. Cuando se arranca la computadora, la BIOS comienza su ejecución automáticamente.
   1.1. Realiza una comprobación del sistema (explora la RAM, los buses, los periféricos, etc.).
   1.2. Determina el dispositivo de arranque de una lista de dispositivos almacenada en memoria.
2. Se lee el primer sector del dispositivo de arranque, se coloca en memoria y se ejecuta.
   2.1. Determina la partición activa (la que contiene el SO).
   2.2. Lee de esta partición el SO y lo inicia.
3. El SO:
   3.1. Carga los drivers en el kernel.
   3.2. Inicializa sus tablas (de procesos, de memoria, de E/S, etc.).
   3.3. Crea procesos.
   3.4. Arranca el inicio de sesión o el GUI.

TIPOS DE SOs

SOs DE MAINFRAME
- Están orientados hacia el **procesamiento de muchos trabajos,** sobre todo de muchas **operaciones de E/S,** a la vez.
  - ➢ Se usan en servidores web de alto rendimiento.
  - ➢ UNIX.

SOs DE SERVIDORES
- Dan servicio a **varios usuarios** a la vez y les permiten compartir los **recursos** de **hardware** y **software**.
  - ➢ Linux, Windows, Solaris.

SOs DE MULTIPROCESADORES
- Son sistemas de **varias CPUS**.
  - ➢ Linux, Windows.

SOs DE COMPUTADORAS PERSONALES (PC)
- Su trabajo es proporcionar un buen soporte a **un sólo usuario**.
  - ➢ Linux, Windows, MacOS.

SOs DE COMPUTADORAS DE BOLSILLO (PDA)
- Proporcionan servicios de telefonía, fotografía digital, etc.
  - ➢ Android, IOS, Windows Phone.

SOs EMPOTRADOS (embedded)
- Tienen un **conjunto cerrado de aplicaciones** y no se pueden instalar nuevas.
  - ➢ Se usan en microondas, reproductores de música, etc.
  - ➢ QNX, VxWorks.

SOs EN TIEMPO REAL
- Su parámetro clave es el **tiempo**.
  - ➢ Se usan en fábricas.

CONCEPTOS BÁSICOS DE LOS SOs

Los procesos tienen su memoria dividida en 3 segmentos:
- Segmento de **texto** (código del programa).
- Segmento de **datos** (variables).
- Segmento de **pila**.

PROCESOS

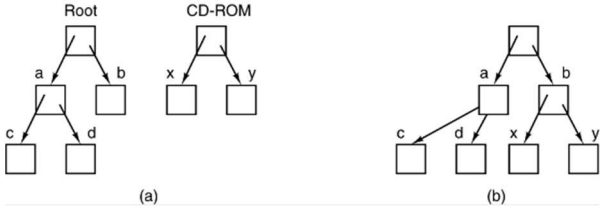
- Un PROCESO es una abstracción de un programa en ejecución que reúne toda la información necesaria para ejecutarlo.
  - Cada proceso tiene asociado un ESPACIO DE DIRECCIONES → lista de **ubicaciones de memoria** desde el 0 a un valor máximo en las que el proceso puede escribir y leer.
  - También tienen asociados otros recursos como registros (PC, SP etc.), lista de archivos abiertos, alarmas pendientes, etc.
- La ejecución de **varios procesos simultáneamente** se consigue haciendo que la CPU conmute muy rápidamente entre ellos (MULTIPROGRAMACIÓN).
  - Para que esto sea posible, cada cierto tiempo el SO debe detener la ejecución de un proceso y comenzar otro (CAMBIO DE CONTEXTO).
    - ↳ Durante su suspensión debe almacenarse toda la información del proceso para que cuando se vuelva a iniciar esté en el mismo estado que tenía cuando se realizó el cambio de contexto.
      - Esta información se almacena en la TABLA DE PROCESOS, que es un array de estructuras, una para cada proceso con existencia actual.
- Los procesos pueden **crear** uno o más procesos aparte. Los procesos creados son los HIJOS y los creadores son los PADRES.
  Estos a su vez pueden tener otros hijos, llegando a una **jerarquía de procesos** en forma de **estructura de árbol.**
- Los procesos a menudo necesitan **comunicarse y sincronizarse** entre sí. Esto es la COMUNICACIÓN ENTRE PROCESOS y se puede realizar por medio de SEÑALES.
- Cuando se inicia un proceso, este almacena el UID (User IDentification) del usuario que lo inició.
  - Cada persona que usa un sistema tiene su propia UID. Varios usuarios se pueden asociar en un grupo, que recibe una GID (Group IDentification).
    - ↳ Una UID notable es la de **superusuario**, que tiene poder especial y puede violar muchas reglas de protección.

ESPACIOS DE DIRECCIONES

- Los espacios de direcciones de los programas en ejecución se mantienen en la **memoria principal**.
  - ↳ Habitualmente, se colocan **varios** programas en memoria principal al mismo tiempo, por lo que se necesita un método para **evitar que interfieran** entre sí (o con el SO).
- Para **desacoplar** el espacio de direcciones de la memoria principal se usa la MEMORIA VIRTUAL → el SO mantiene una parte del espacio de direcciones en la memoria principal y otra en el disco, moviendo trozos de un lugar a otro cuando sea necesario.

ARCHIVOS

- Un DIRECTORIO es una manera de agrupar archivos.
- Un directorio puede contener **archivos** u otros **directorios**, dando lugar a una jerarquía en forma de **estructura de árbol**.
  - ↳ El DIRECTORIO RAÍZ es aquel que se encuentra en el nivel más alto de la jerarquía.
  - El NOMBRE DE RUTA especifica cada archivo dentro de la jerarquía de directorios.
    - Nombres de ruta **absolutos** (empiezan por '/')→ lista de directorios que deben recorrerse desde el directorio raíz para llegar al archivo.
    - Nombres de ruta **relativos** (no empiezan por '/')→ omiten la ruta hasta el directorio de trabajo actual.
- El SISTEMA DE ARCHIVOS MONTADO consiste en incluir otros **sistemas de archivos externos** a la jerarquía del sistema de archivos raíz del disco duro.
  - ↳ En principio no se puede acceder a los archivos de los sistemas de almacenamiento externos pues no existe manera de especificar sus nombres de ruta.
  - La syscall *mount* permite adjuntar el sistema de archivos externo al sistema de archivos raíz en donde el programa desea que esté.



PROTECCIÓN

- Las computadoras tienen grandes cantidades de información que los usuarios desean mantener confidencial. Es **responsabilidad del SO** administrar la seguridad del sistema.

▷ Protección de archivos en UNIX → código de protección binario de 9 bits divididos en 3 campos de 3 bits.

- Los campos son para el usuario, grupo y el resto.
- Los bits son para leer (r), escribir (w) y ejecutar (x).
- Un 1 indica que se tiene ese permiso y un 0 indica que no se tiene.

SHELL

○ El SHELL es el intérprete de comandos del SO.

- No forma parte del SO, es la interfaz con el usuario, pero utiliza muchas de sus características.
- Hay varios shells distintos: $sh, csh, ksh, bash$ ...

- La salida y entrada **estándar** son la terminal.
  - **Redireccionamiento** → se puede redirigir la entrada y salida estándar con los operandos < y >, respectivamente.
  - **Canalización** → la salida de un programa se puede usar como entrada para otro con el operando |.
  - **Comodines** → $\begin{cases} * \rightarrow \text{el shell lo sustituye por todas las posibles combinaciones de caracteres provenientes del directorio en cuestión.} \\ ? \rightarrow \text{igual que } * \text{ pero sólo para un único caracter.} \end{cases}$

```
/home/diego$ ls hu?o
hugo
/home/diego$ ls p??o
paco
/home/diego$ ls ???s
luis
```

```
/home/diego$ ls h*
hugo
/home/diego$ ls *is
luis
/home/diego$ ls *u*
hugo luis
/home/diego$ ls p*o
paco
/home/diego$
```

```
date >archivo
sort <archivo1 >archivo2
cat archivo1 archivo2 archivo3 | sort >/dev/lp
```

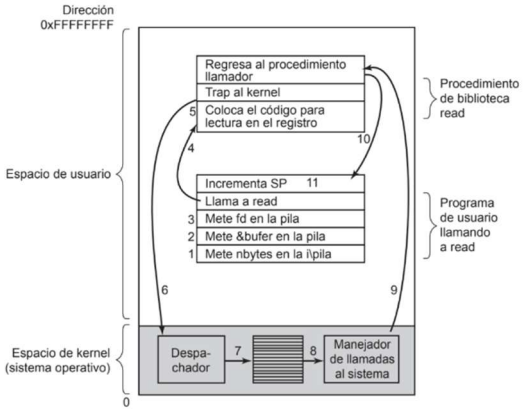
LLAMADAS AL SISTEMA

- Las syscall disponibles **varían de un sistema a otro**. Nosotros usaremos el estándar **POSIX**.
- La verdadera mecánica relacionada con la acción de emitir una syscall es muy **dependiente de la máquina** (y a menudo se expresa en **ensamblador**) → se proporciona al programador una **biblioteca de procedimientos** para poder realizar syscalls desde programas en C y otros lenguajes.

▷ EJEMPLO: $read(fd, buffer, nbytes)$:

- Argumentos → archivo, puntero al buffer, número de bytes a leer.
- Devuelve → número de bytes leídos o -1 si no se pudo realizar la llamada.
  - El número de error se coloca en una variable global llamada $errno$.

- Los programas siempre deben comprobar los resultados de una syscall para ver si ocurrió un **error**.



PASOS PARA REALIZAR UNA SYSCALL

1-3. El programa llamador mete los parámetros de la función en la pila.
4. El programa llamador llama al procedimiento de la biblioteca (que seguramente esté en ensamblador).
5. El procedimiento de biblioteca coloca el número de syscall en un lugar en el que el SO lo espera, como un registro.
6. El procedimiento de biblioteca ejecuta una trap para cambiar el modo usuario a kernel y empezar la ejecución en una dirección fija dentro del núcleo.
   - La instrucción trap no puede saltar a una dirección arbitraria, siempre a una fija.
7. El código de kernel examina el número de syscall y lo pasa al manejador de syscalls correspondiente, a través de una tabla de apuntadores a manejadores de syscalls indexada en base al número de la syscall.
8. Se ejecuta el manejador de syscalls.
9. Se devuelve el control (o no) al procedimiento de biblioteca, en la instrucción de después del trap.
10. El procedimiento de biblioteca regresa al programa de usuario.
11. El programa usuario limpia la pila, como después de la llamada a cualquier procedimiento.

LLAMADAS AL SISTEMA

Administración de procesos

Llamada	Descripción
pid = fork()	Crea un proceso hijo, idéntico al padre
pid = waitpid(pid, &statloc, opciones)	Espera a que un hijo termine
s = execve(nombre, argv, entornp)	Reemplaza la imagen del núcleo de un proceso
exit(estado)	Termina la ejecución de un proceso y devuelve el estado

Administración de archivos

Llamada	Descripción
fd = open(archivo, como, ...)	Abre un archivo para lectura, escritura o ambas
s = close(fd)	Cierra un archivo abierto
n = read(fd, bufer, nbytes)	Lee datos de un archivo y los coloca en un búfer
n = write(fd, bufer, nbytes)	Escribe datos de un búfer a un archivo
posicion = lseek(fd, desplazamiento, dedonde)	Desplaza el apuntador del archivo
s = stat(nombre, &buf)	Obtiene la información de estado de un archivo

Administración del sistema de directorios y archivos

Llamada	Descripción
s = mkdir(nombre, modo)	Crea un nuevo directorio
s = rmdir(nombre)	Elimina un directorio vacío
s = link(nombre1, nombre2)	Crea una nueva entrada llamada nombre2, que apunta a nombre1
s = unlink(nombre)	Elimina una entrada de directorio
s = mount(especial, nombre, bandera)	Monta un sistema de archivos
s = umount(especial)	Desmonta un sistema de archivos

Llamadas varias

Llamada	Descripción
s = chdir(nombredir)	Cambia el directorio de trabajo
s = chmod(nombre, modo)	Cambia los bits de protección de un archivo
s = kill(pid, senial)	Envía una señal a un proceso
segundos = tiempo(&segundos)	Obtiene el tiempo transcurrido desde Ene 1, 1970

Figura 1-18. Algunas de las principales llamadas al sistema POSIX. El código de retorno *s* es −1 si ocurrió un error. Los códigos de retorno son: *pid* es un id de proceso, *fd* es un descriptor de archivo, *n* es una cuenta de bytes, *posicion* es un desplazamiento dentro del archivo y *segundos* es el tiempo transcurrido. Los parámetros se explican

ESTRUCTURA DEL SO UNIX