



ARQUITECTURA Y SISTEMAS OPERATIVOS

CLASE 2

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS



BIBLIOGRAFIA

- Operating System Concepts. By Abraham, Silberschatz.
 - Capitulo I
 - Capitulo II
- Modern Operating Systems. By Andrew S. Tanenbaum.
 - Capitulo I

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS



¿QUE ES UN SISTEMA OPERATIVO?



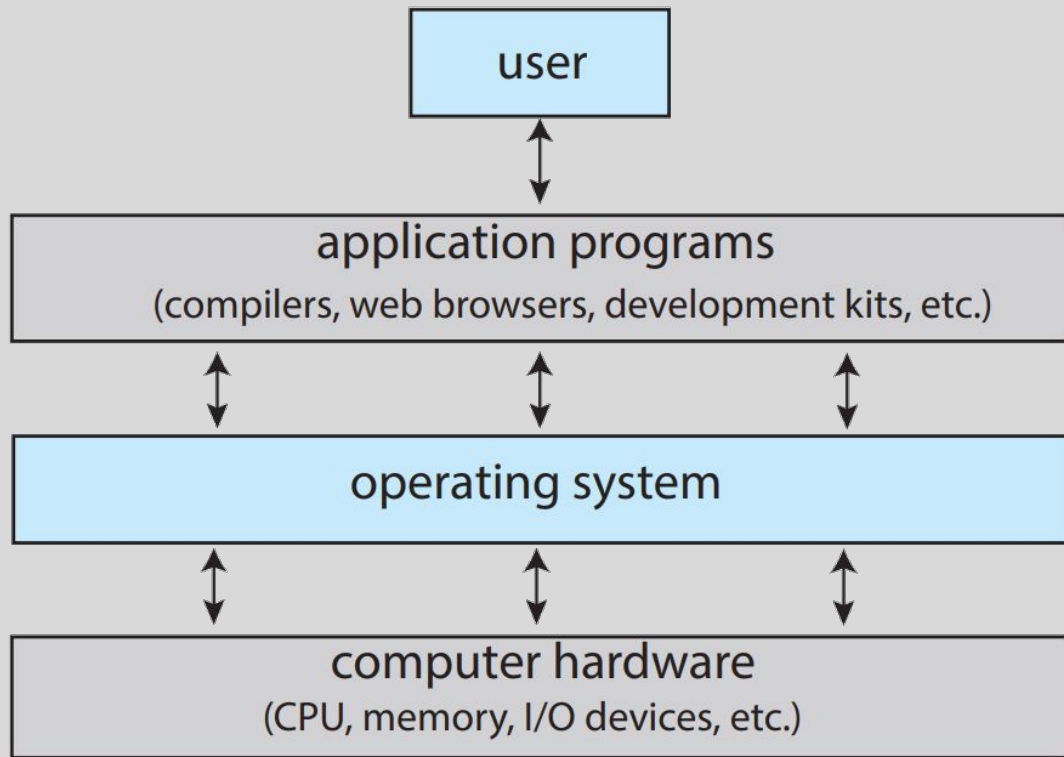
Un **sistema operativo** es un *software* que gestiona el *hardware* de una computadora.

Proporciona la *base* para los *programas de aplicación* y actúa como intermediario entre el usuario de la computadora y el hardware.

Para explorar el rol de un sistema operativo en un entorno informático actual, es importante comprender la organización y la arquitectura del hardware.

Esto incluye la CPU, la memoria y los dispositivos de E/S, así como el almacenamiento.

Una responsabilidad fundamental de un sistema operativo es asignar estos recursos a los programas.



¿QUE HACE UN SISTEMA OPERATIVO?

El *hardware* (la unidad central de procesamiento (CPU), la memoria y los dispositivos de entrada/salida (E/S)) **proporciona** los recursos informáticos básicos del sistema.

Los *programas de aplicación* **definen** cómo se **utilizan** estos recursos para resolver los problemas informáticos de los usuarios.

El **sistema operativo** controla el **hardware** y coordina su uso entre los **distintos programas de aplicación** para el o los usuarios.



¿QUE HACE UN SISTEMA OPERATIVO?

Desde el punto de vista del usuario, el objetivo es **maximizar el trabajo** (o *el ocio*) que realiza el usuario. En este caso, el sistema operativo está diseñado principalmente para la **facilidad de uso**, prestando *cierta* atención al **rendimiento** y la **seguridad**, y *ninguna* a la **utilización de recursos** (cómo se comparten los distintos recursos de hardware y software).

Desde la perspectiva de una computadora, el SO es el *software* más cercano al *hardware*. En este contexto, podemos considerar un SO como un **asignador de recursos**. Ante numerosas y posiblemente conflictivas solicitudes de recursos, el **sistema operativo** debe decidir cómo asignarlos a programas y usuarios específicos para que estos puedan operar el sistema de forma eficiente y justa.

DEFINIENDO SISTEMAS OPERATIVOS

Entonces...

¿Cómo podemos definir qué es un
sistema operativo?

En general, no existe una definición completamente adecuada y universalmente aceptada de sistema operativo.

Los sistemas operativos existen porque ofrecen una **solución razonable** para crear un sistema informático **utilizable**.

El objetivo fundamental de los sistemas operativos es **ejecutar programas** y facilitar la resolución de problemas del usuario.

Dado que el hardware por sí solo no es particularmente fácil de usar, se desarrollan programas de aplicación, que requieren ciertas operaciones comunes, tiempo de CPU, memoria y dispositivos de E/S.

Las funciones comunes de control y asignación de recursos se unen en un solo software: **el sistema operativo**.

DEFINIENDO SISTEMAS OPERATIVOS

Entonces...

¿Cómo podemos definir qué es un
sistema operativo?

La definición más común, y la que vamos a adoptar en este curso, es que el sistema operativo es el único programa que se ejecuta constantemente en la computadora, generalmente llamado **kernel**.

Además del kernel, existen otros dos tipos de programas:

- los **programas de sistema**, que están asociados con el sistema operativo, pero no necesariamente forman parte del kernel.
- los **programas de aplicación**, que incluyen todos los programas no asociados con el funcionamiento del sistema.

¿POR QUÉ ESTUDIAR SISTEMAS OPERATIVOS?

Porque si.

Porque es interesante.

Porque comprender los fundamentos de los sistemas operativos, **cómo** controlan el hardware de las computadoras y **qué** proporcionan a las aplicaciones no solo es esencial para quienes los desarrollan, sino también muy útil para quienes escriben programas en ellos y los usan.

Porque hay que aprobar.

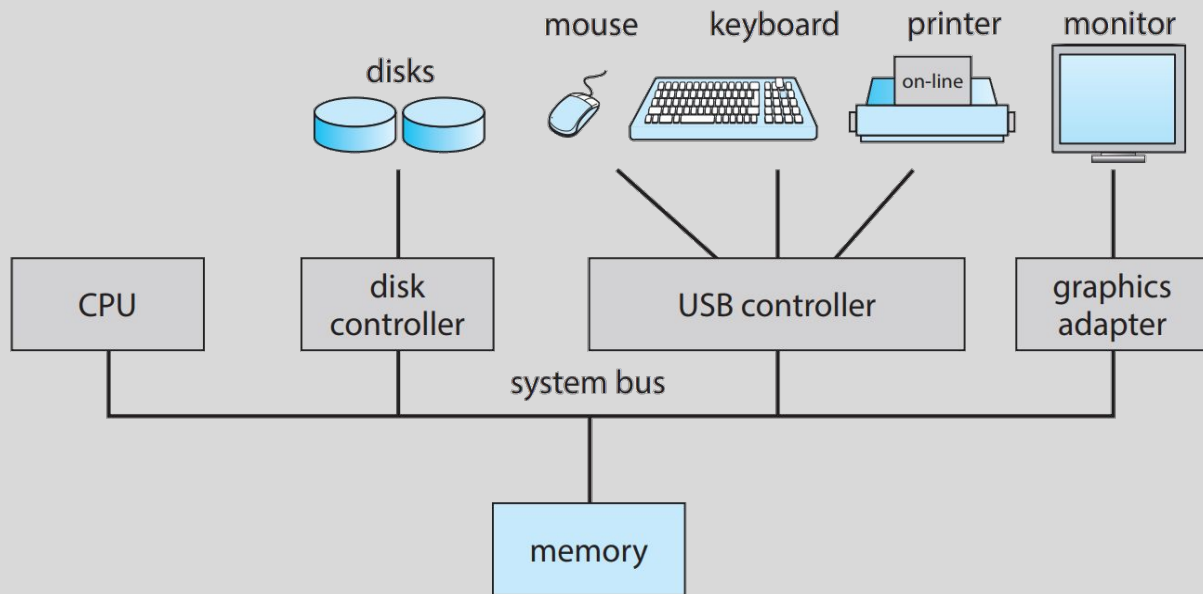
ORGANIZACIÓN DEL SISTEMA



ORGANIZACIÓN DE UNA COMPUTADORA

Una computadora está compuesta por uno o más CPUs, memoria y varios **controladores de dispositivos** (*device controllers*), conectados a través de un **bus** común que proporciona acceso entre los distintos componentes y la memoria compartida.

El **device controller** es responsable de mover los datos entre los dispositivos periféricos que controla y su almacenamiento local.

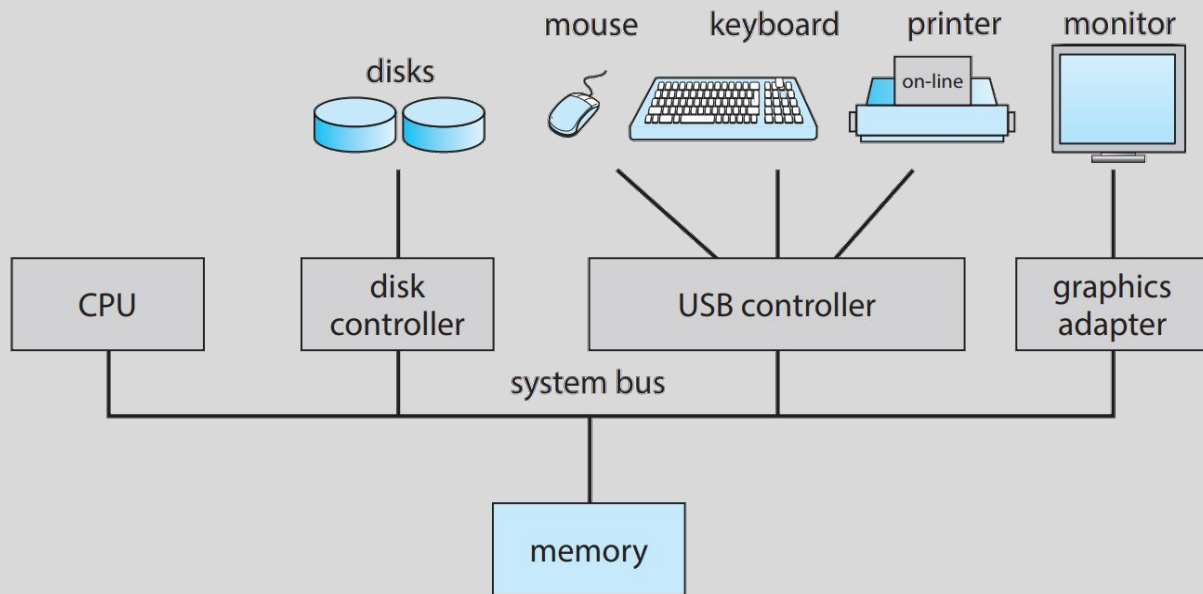


ORGANIZACIÓN DE UNA COMPUTADORA

Los SO tienen un **driver de dispositivo** (*device driver*) para cada controlador. Este driver conoce al controlador y proporciona al resto del SO una interfaz *uniforme* con el dispositivo.

La CPU y los *device controllers* pueden ejecutarse en **paralelo**, compitiendo por ciclos de memoria.

Para *garantizar* un acceso ordenado a la memoria compartida, un controlador de memoria sincroniza el acceso a la misma.

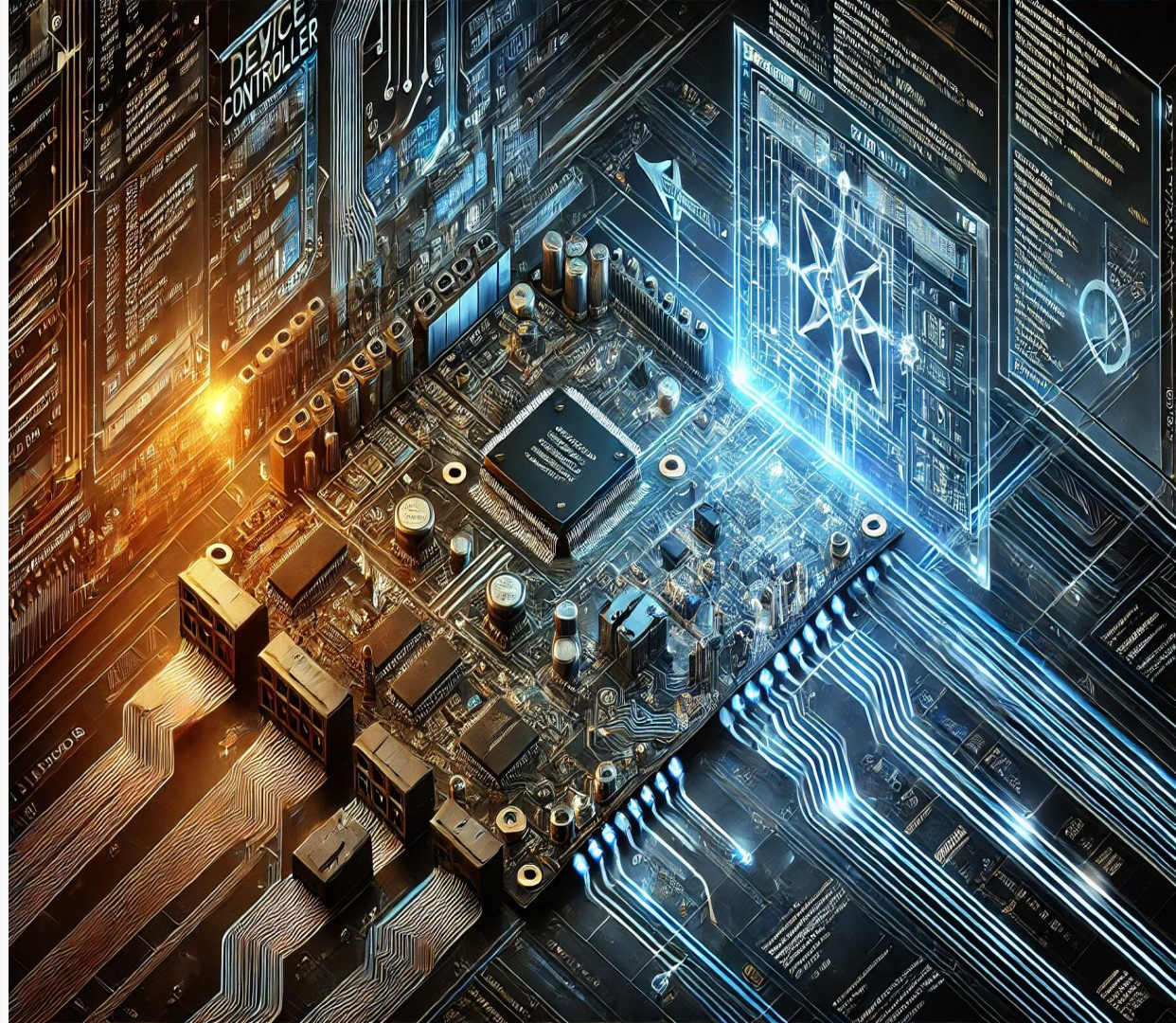


NOTA

Device controller es *hardware* físico que gestiona el funcionamiento real de un dispositivo (gestiona las transferencias de datos, la comunicación con el bus, etc.).

Device driver es *software* que proporciona al SO una interfaz estandarizada para interactuar con dicho hardware.

Esta separación ayuda a abstraer las especificaciones, por un lado del hardware, y por otro del SO, lo que facilita la administración y la interoperabilidad de los dispositivos.





CONCEPTOS DE SISTEMAS OPERATIVOS



PROCESOS Y ESPACIO DE DIRECCIONES

Un concepto clave en todos los sistemas operativos es el de **proceso**.

Un proceso es básicamente un programa en ejecución. Asociado a cada proceso se encuentra su espacio de direcciones, que es una lista de ubicaciones de memoria desde 0 hasta un máximo determinado, donde el proceso puede leer y escribir. El **espacio de direcciones** contiene el programa ejecutable, los datos del programa y su *stack*.

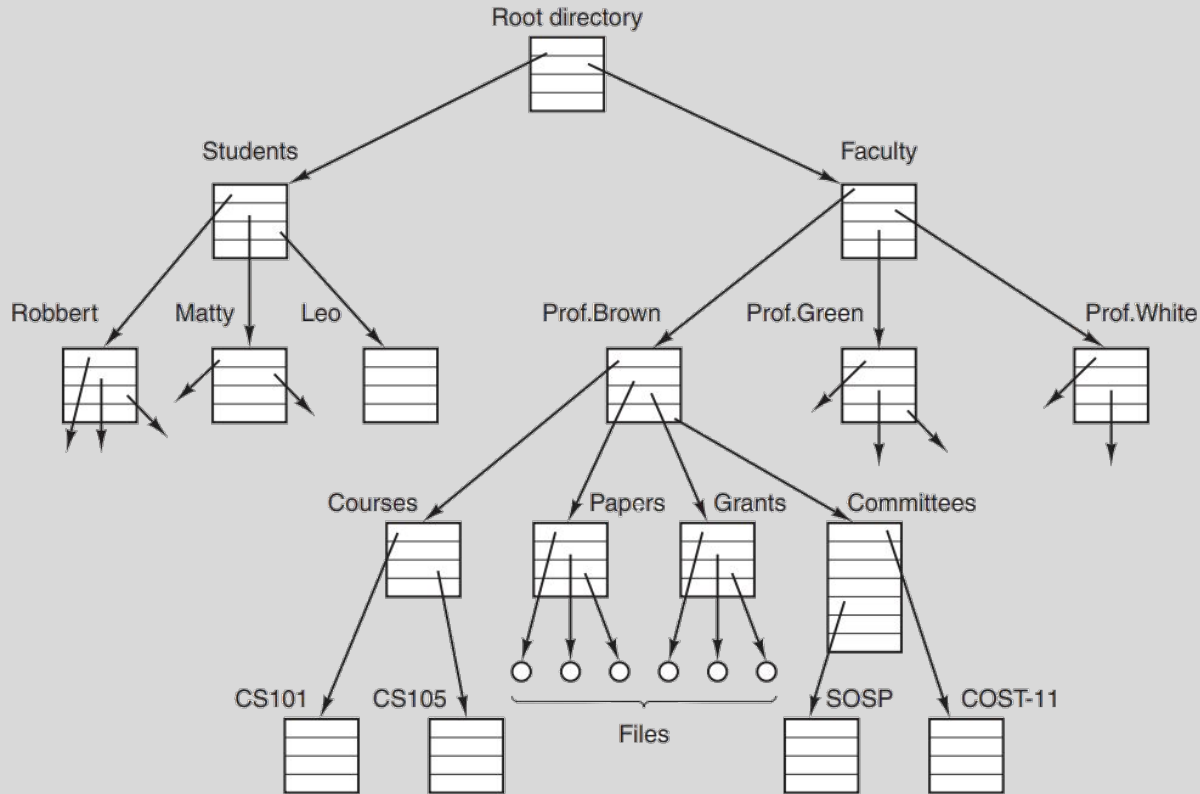
También se asocia a cada proceso un conjunto de recursos, que suelen incluir registros (incluido el PC), una lista de archivos abiertos, alarmas pendientes, listas de procesos relacionados y toda la demás información necesaria para ejecutar el programa. Un proceso es fundamentalmente un contenedor que mantiene toda la información necesaria para ejecutar un programa.

En muchos sistemas operativos, toda la información sobre cada proceso, salvo el contenido de su propio espacio de direcciones, se almacena en una **tabla** denominada **tabla de procesos**, que es una matriz de estructuras, una para cada proceso existente.

ARCHIVOS

Una función principal del SO es ocultar las particularidades de los discos y otros dispositivos de E/S, y presentar al programador o usuario un modelo abstracto, limpio, claro e independiente del dispositivo subyacente. El **archivo**.

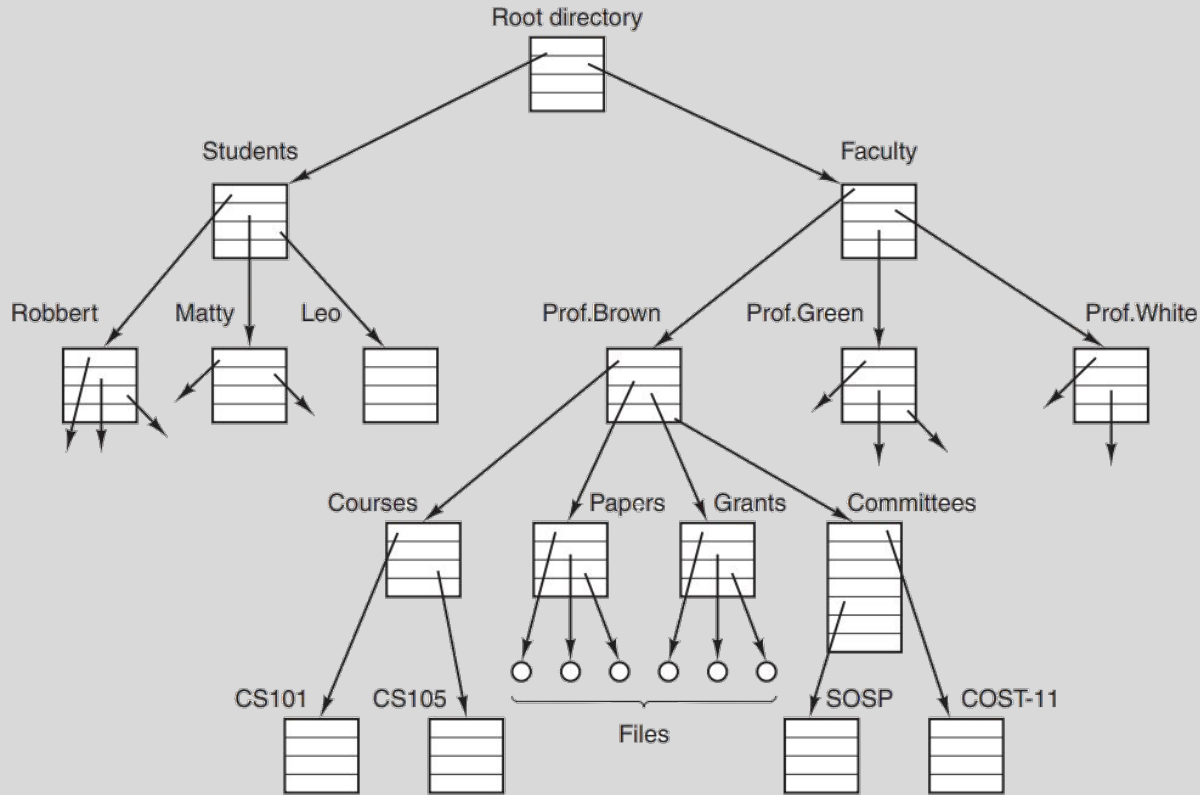
Para proporcionar un lugar donde guardar archivos, la mayoría de los SO utilizan el concepto de **directorio** como una forma de agruparlos.



ARCHIVOS

Cada archivo dentro de la jerarquía de directorios se puede especificar proporcionando su nombre de ruta (*path*) desde la parte superior de la jerarquía, el directorio raíz (*root*). Estos *path* absolutos consisten en la lista de directorios que se deben recorrer desde el directorio raíz para llegar al archivo, con barras diagonales (/) que separan los componentes.

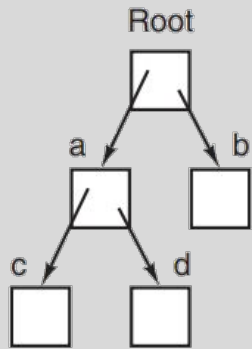
Antes de poder leer o escribir un archivo, debe abrirse, momento en el que se verifican los permisos. Si se permite el acceso, el sistema devuelve un pequeño entero llamado **descriptor de archivo** para usar en operaciones posteriores. Si se prohíbe el acceso, se devuelve un código de error.



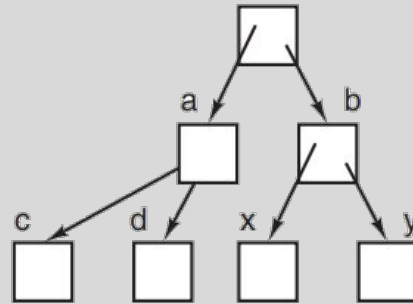
Ejemplo. El *path* para el archivo "CS101" es:
/Faculty/Prof.Brown/Courses/CS101

ARCHIVOS

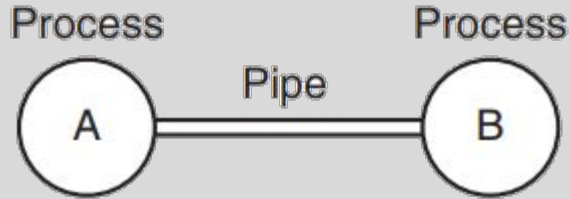
Otro concepto importante es el de *mounting*. Para proporcionar una forma de gestionar medios extraíbles, UNIX permite que el sistema de archivos del medio extraíble se conecte al árbol principal.



(a)



(b)



ARCHIVOS

Esta característica se relaciona tanto con procesos como con archivos: los *pipes*. Un pipe es un tipo de pseudo archivo que permite conectar dos procesos.

Cuando el proceso A desea enviar datos al proceso B, escribe en el pipe como si fuera un archivo de salida. El proceso B puede leer los datos leyendo desde el pipe como si fuera un archivo de entrada.

Así, la comunicación entre procesos en UNIX se asemeja mucho a la lectura y escritura de archivos.



ENTRADA / SALIDA

Cada sistema operativo cuenta con un subsistema de E/S para gestionar sus dispositivos de E/S.

Parte del software de E/S es independiente del dispositivo, es decir, se aplica a muchos o todos los dispositivos de E/S por igual.

Otras partes, como los *device controllers*, son específicas de cada dispositivo de E/S.



PROTECCION

Las computadoras contienen grandes cantidades de información que los usuarios suelen querer proteger y mantener confidencial (correos electrónicos, planes de negocios, declaraciones de impuestos, etc). El sistema operativo gestiona la seguridad del sistema para que, por ejemplo, solo los usuarios autorizados puedan acceder a los archivos.

En UNIX, los archivos se protegen asignando a cada uno un código de protección binario de 9 bits. El código de protección consta de tres campos de 3 bits: uno para el propietario, otro para los demás miembros del grupo del propietario y otro para el resto. Cada campo tiene un bit para acceso de lectura, acceso de escritura y acceso de ejecución. Estos 3 bits se conocen como bits *rwX*.

Por ejemplo, el código de protección *rwxr-x--x* significa que el propietario puede leer, escribir o ejecutar el archivo, los demás miembros del grupo pueden leer o ejecutar (pero no escribir) y todos los demás pueden ejecutar (pero no leer ni escribir). Para un directorio, la *x* indica permiso de búsqueda. Un guión indica que no se dispone del permiso correspondiente.



SHELL

Shell, es el intérprete de comandos de UNIX, y aunque no forma parte del sistema operativo, utiliza intensivamente muchas de sus funciones y, por lo tanto, sirve como un buen ejemplo de cómo se utilizan las llamadas al sistema (*system calls*).

También es la interfaz principal entre el usuario que se encuentra frente a su terminal y el sistema operativo, a menos que utilice una interfaz gráfica. La *shell* utiliza la terminal como entrada y salida estándar.

Si el usuario escribe:

```
$ date
```

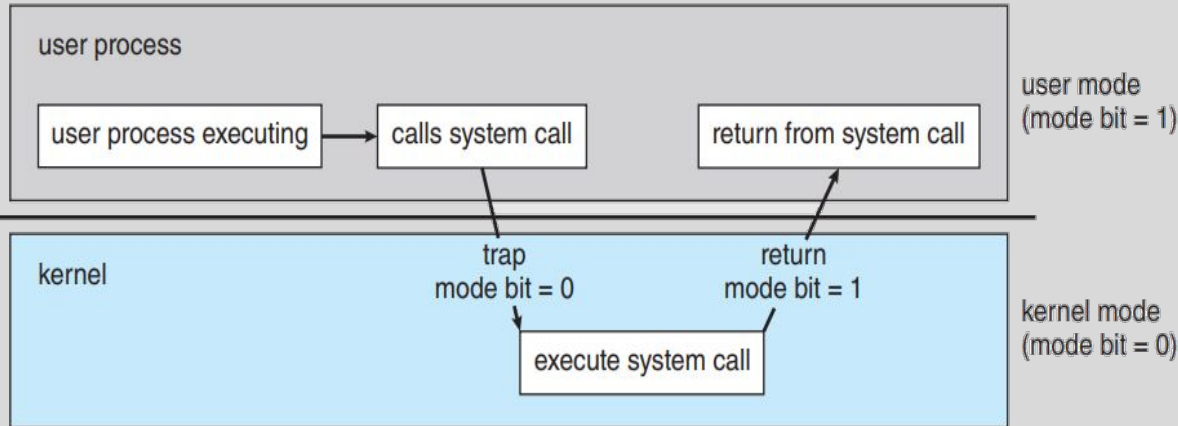
la *shell* crea un proceso hijo y ejecuta el programa `date` como hijo. Mientras el proceso hijo se ejecuta, la *shell* espera a que finalice. Cuando termina, la *shell* vuelve a escribir el *prompt* (\$) e intenta leer la siguiente línea de entrada.

DUAL-MODE

Dado que el SO y sus usuarios comparten los recursos de HW y SW, un SO diseñado adecuadamente debe garantizar que un programa incorrecto (o malicioso) no pueda hacer que otros programas, o el propio SO, fallen.

Para garantizar la ejecución adecuada del sistema, debemos poder distinguir entre la ejecución del código del SO y el código definido por el usuario.

Esto se logra mediante soporte de hardware que permita la diferenciación entre varios modos de ejecución.

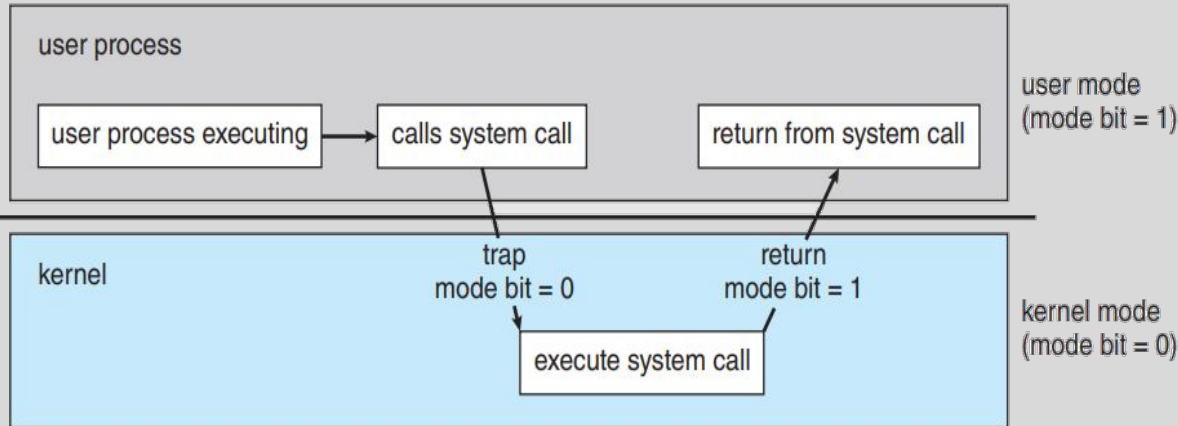


DUAL-MODE

Como mínimo, se necesitan dos modos de operación: modo usuario y modo kernel.

Para esto, se agrega un bit, llamado **bit de modo**, al HW de la computadora para indicar el modo actual: kernel (0) o usuario (1). Así, podemos distinguir entre una tarea que se ejecuta en nombre del SO y una que se ejecuta en nombre del usuario.

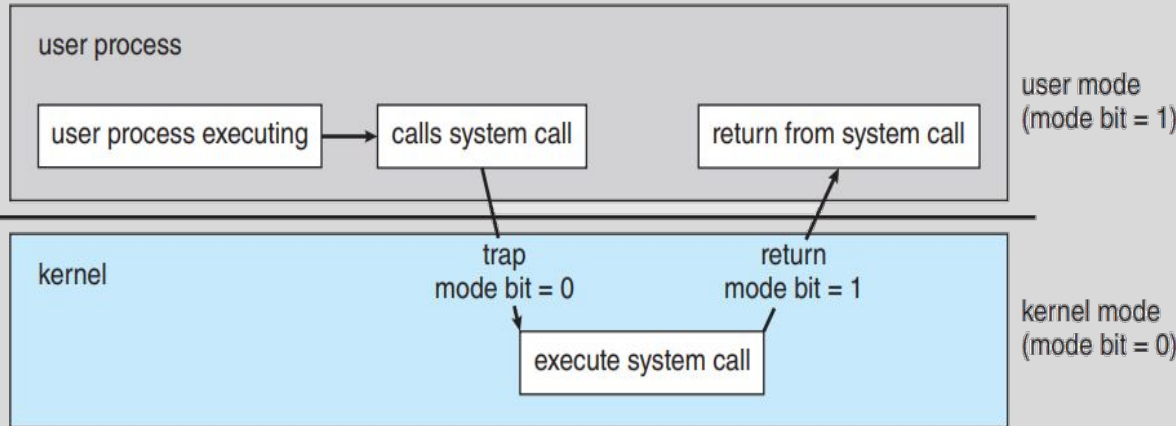
Cuando el sistema se está ejecutando una aplicación de usuario, está en modo usuario. Cuando la aplicación solicita un servicio del SO (a través de una **system call**), el sistema debe pasar del modo usuario al modo kernel para cumplir con la solicitud.



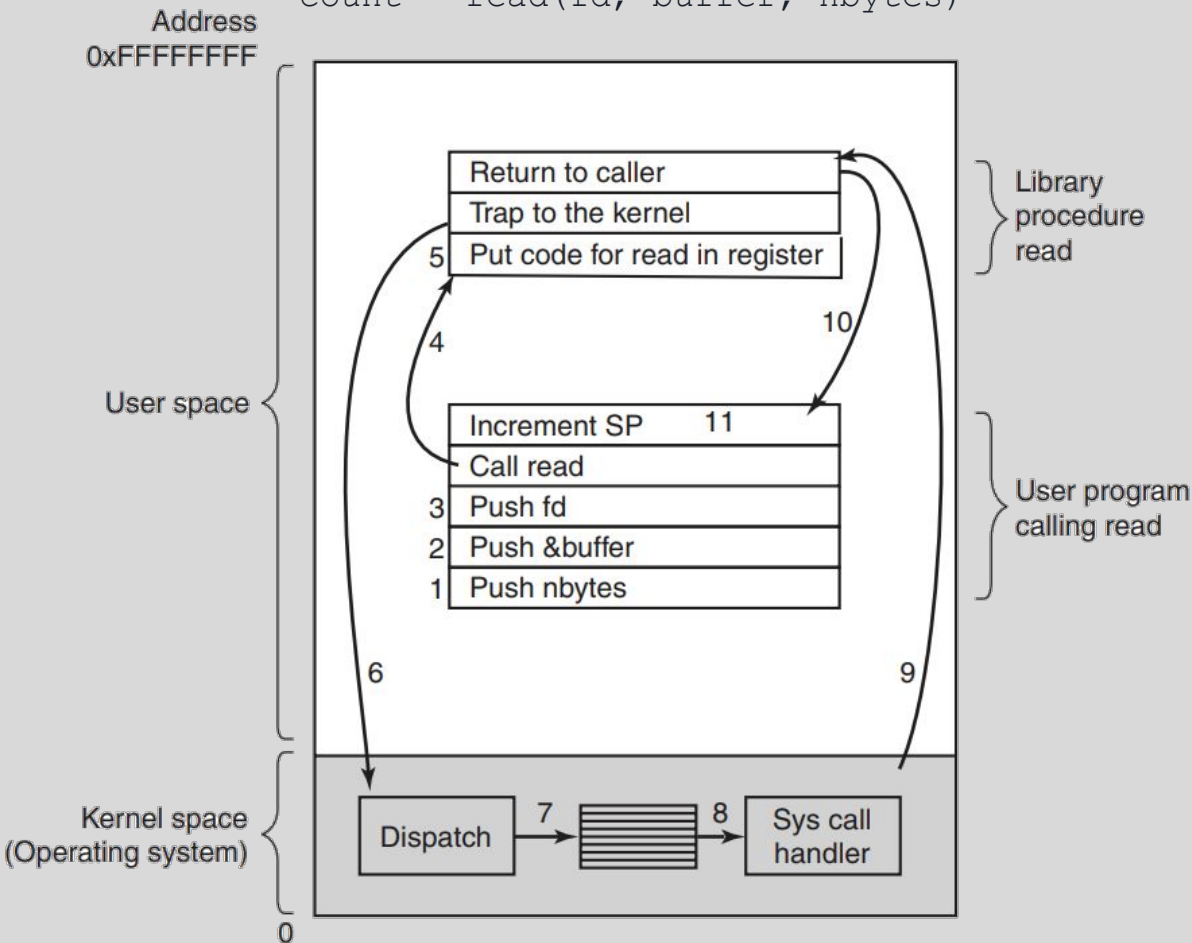
SYSTEM CALL

Las **system calls** proporcionan los medios para que un programa de usuario solicite al SO que realice tareas reservadas en nombre del programa de usuario. Una **system call** generalmente lleva la forma de un *salto* a una ubicación específica en la tabla de interrupciones.

Cuando se ejecuta una *syscall*, el HW la trata como una interrupción de SW. El control pasa a través de la tabla de interrupciones a una rutina de servicio en el SO, y el bit de modo se configura en modo kernel. La rutina de servicio es parte del SO. El kernel verifica que los parámetros son correctos y legales, ejecuta la solicitud y devuelve el control a la instrucción que sigue.



```
count = read(fd, buffer, nbytes)
```



SYSTEM CALL

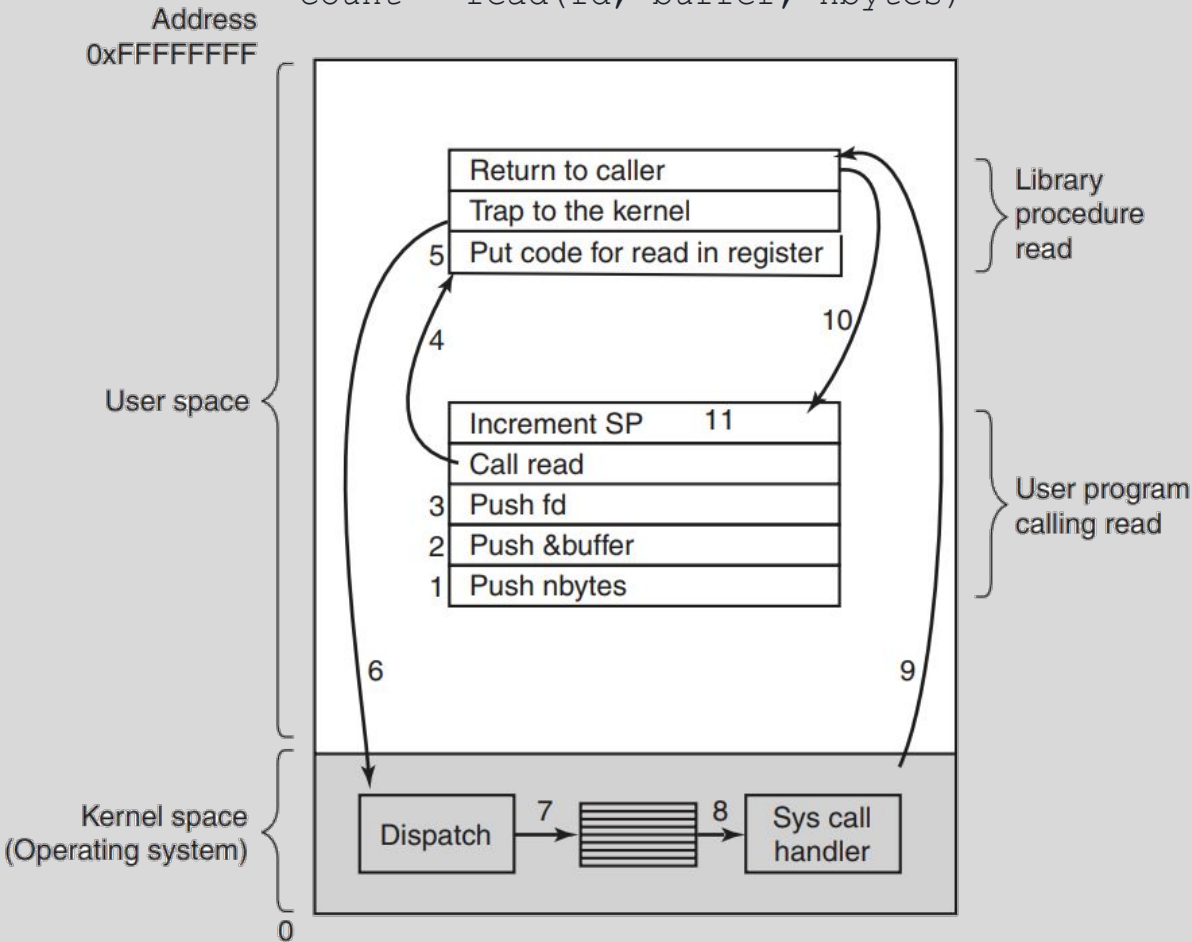
Las *system calls* se realizan en una serie de pasos.

El programa que realiza la llamada primero introduce los parámetros en la pila (1, 2 y 3).

A continuación, se realiza la llamada al procedimiento de la librería (4). Esto es, el programa que se encarga de realizar efectivamente la llamada al SO.

Este procedimiento, posiblemente escrito en lenguaje ensamblador, coloca el número de la *syscall* donde el SO la espera, en general en un registro (5).

```
count = read(fd, buffer, nbytes)
```



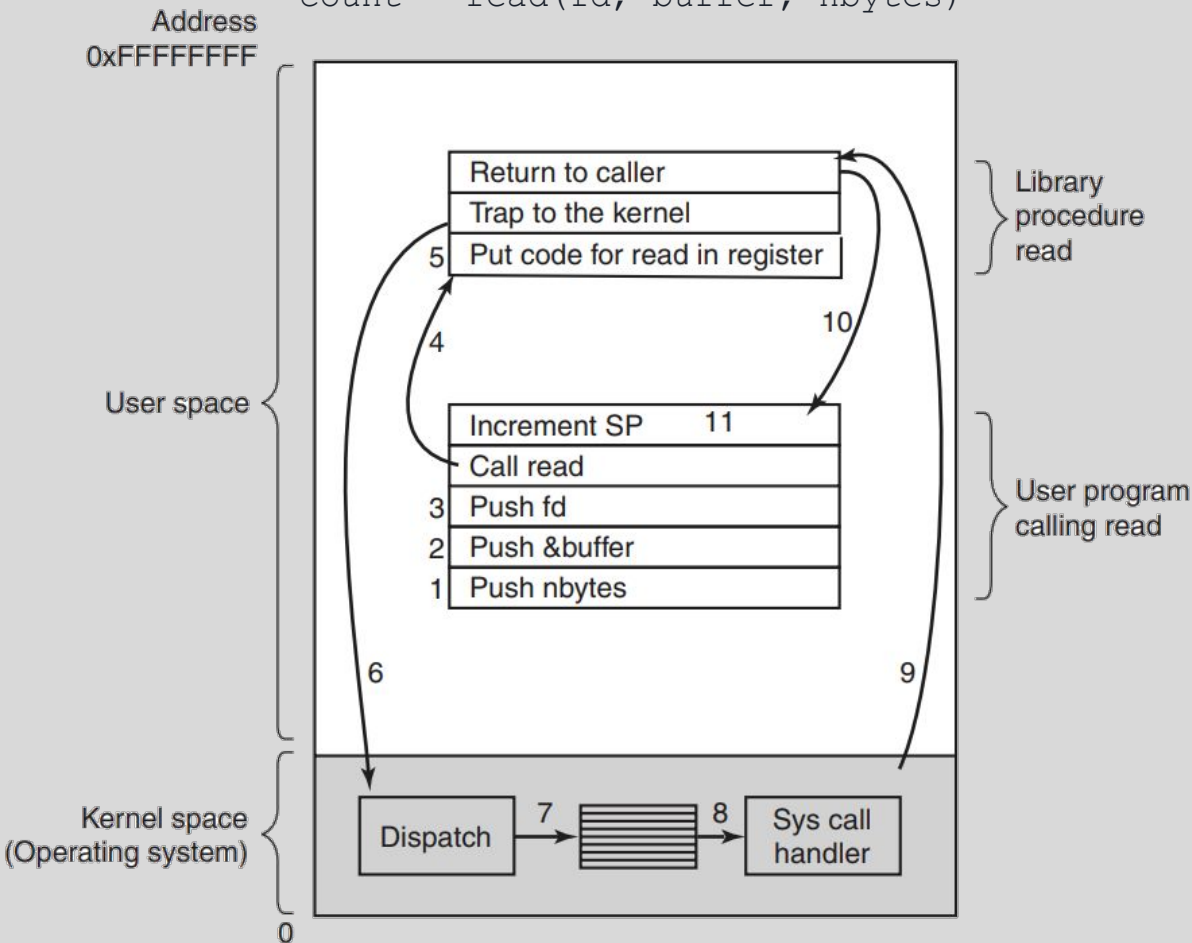
SYSTEM CALL

A continuación, ejecuta una instrucción TRAP para cambiar del modo usuario al modo kernel e iniciar la ejecución en una dirección fija dentro del kernel (6).

El código del kernel examina el número de la *syscall* y luego lo envía al *dispatcher* correcto, generalmente mediante una tabla de punteros a *syscall dispatchers* indexados según el número de la *syscall* (7).

En ese momento, se ejecuta el *dispatcher* (8).

```
count = read(fd, buffer, nbytes)
```



SYSTEM CALL

Una vez completado su trabajo, el control puede devolverse al procedimiento de la librería en el espacio de usuario mediante la instrucción siguiente a la instrucción TRAP (9).

Este procedimiento regresa al programa de usuario (10).

Para finalizar, el programa de usuario debe limpiar la *stack*, como lo hace después de cualquier llamada a procedimiento (11).



TIMER

Hay que asegurarse de que el SO mantenga el control sobre la CPU. No podemos permitir que un programa de usuario se bloquee en un bucle infinito o que falle al llamar a una *syscall* y nunca devuelva el control al SO.

Para esto, se usa un **timer**, que puede configurarse para interrumpir a la computadora tras un periodo específico de tiempo.

Esto es, el SO setea un contador, cada vez que el clock marca, el contador se decrementa. Cuando el contador llega a 0, se produce una interrupción. Si el temporizador interrumpe, el control se transfiere automáticamente al SO, que puede tratar la interrupción como un error fatal o conceder más tiempo al programa.

Antes de ceder el control al usuario, el SO se asegura de que el temporizador esté configurado para interrumpir.



ORGANIZACIÓN DEL SISTEMA

Funcionamiento básico

- Interrupciones, alertan a la CPU sobre eventos que requieren atención
- Estructura de almacenamiento
- Estructura de E/S



INTERRUPCIONES

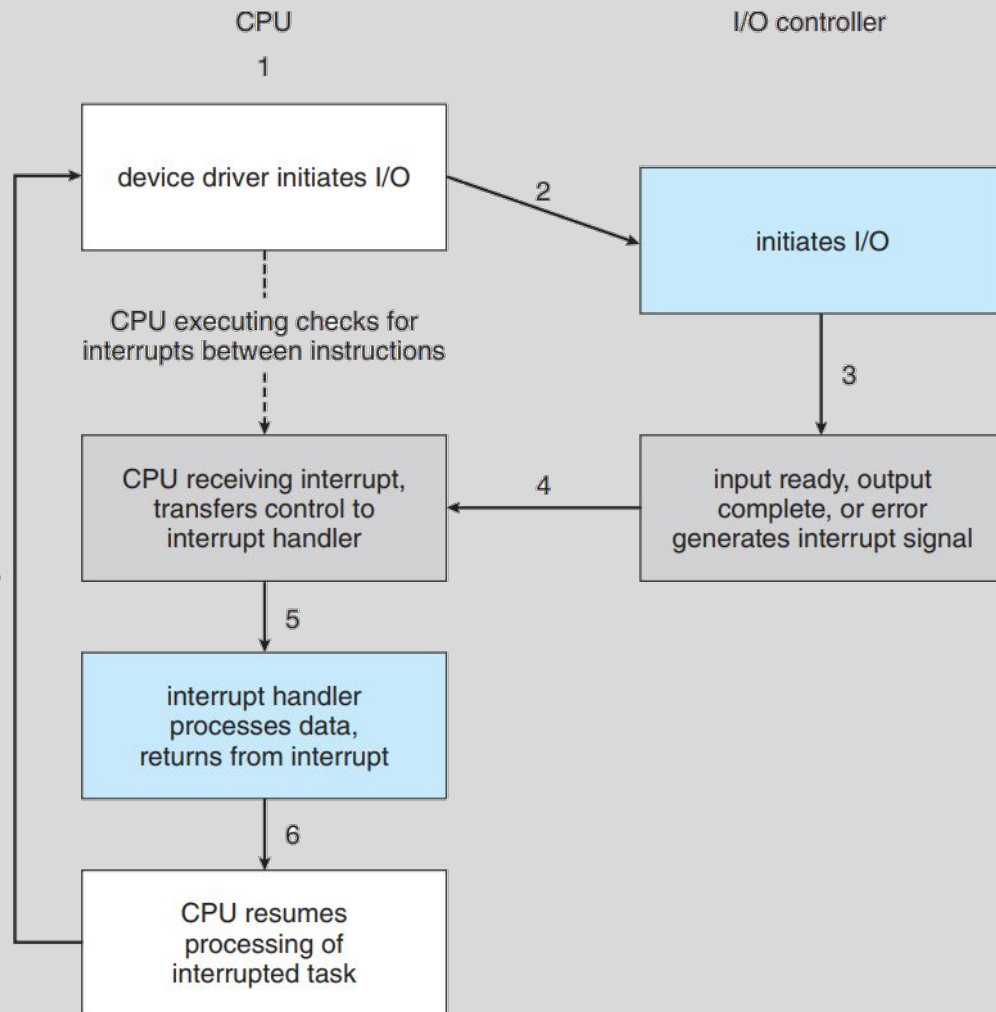
Consideren una operación típica: un programa que lee el valor ingresado por una tecla.

Para iniciar una operación de I/O, el *device driver* carga los registros correspondientes en el *device controller*.

El *device controller*, examina el contenido de estos registros para determinar la acción a realizar ("leer un carácter del teclado"). Luego inicia la transferencia de datos **desde** el dispositivo **hacia** su búfer local y, una vez completada la transferencia, el *device controller* informa al *device driver* que ha finalizado la operación.

Pero ¿cómo informa el *device controller* al *device driver* que ha finalizado la operación?

Mediante una **interrupción**.



INTERRUPCIONES

El hardware puede activar una interrupción en cualquier momento enviando una señal a la CPU.

Cuando la CPU se interrumpe, detiene lo que está haciendo y transfiere inmediatamente la ejecución a una ubicación fija que contiene la dirección de inicio de la rutina de servicio de la interrupción.

La rutina de servicio se ejecuta.

Si la interrupción necesita modificar el estado del procesador, debe guardar explícitamente el estado actual y luego restaurarlo antes de regresar el control.

Una vez atendida la interrupción, el cálculo interrumpido se reanuda como si la interrupción no se hubiera producido.



ESTRUCTURA DE ALMACENAMIENTO

La CPU solo puede cargar instrucciones desde la memoria, por lo que cualquier programa debe primero cargarse en la **memoria principal** (DRAM) para ejecutarse.

También se utilizan otros tipos de memoria. Por ejemplo, el primer programa que se ejecuta al encender la computadora, bootstrap program, se encarga de cargar el sistema operativo. Dado que la RAM es volátil, no podemos confiar en que contenga el programa de arranque.

Para este y otros fines, se utilizan memorias EEPROM y otras formas de firmware: almacenamiento con poca frecuencia de escritura y no volátil.

Siempre la memoria se presenta como una matriz de bytes, donde cada byte tiene su propia dirección. Y la interacción con el CPU se logra mediante una secuencia de instrucciones de **LOAD** o **STORE** en direcciones de memoria específicas.



ESTRUCTURA DE ALMACENAMIENTO

LOAD: mueve un **byte** o **word** de la memoria principal a un registro interno de la CPU.

STORE: mueve el contenido de un registro a la memoria principal.

La unidad básica de almacenamiento es el **bit**, y puede contener uno de dos valores: 0 o 1. Un **byte** consta de 8 bits, que es la porción de almacenamiento más práctica. En general, las computadoras no tienen una instrucción para mover un *bit*, pero sí una para mover un *byte*.

Idealmente, queremos que los programas y datos residan en la memoria principal de forma permanente. Aunque esto no es posible por dos razones:

1. La memoria principal suele ser demasiado pequeña para almacenar todos los programas y datos necesarios de forma permanente.
2. La memoria principal, es volátil: pierde su contenido al interrumpirse la alimentación.



ESTRUCTURA DE ALMACENAMIENTO

Por esto, la mayoría de los sistemas proporcionan **almacenamiento secundario**, como una extensión de la memoria principal. El principal requisito del almacenamiento secundario es que pueda almacenar grandes cantidades de datos de forma permanente.

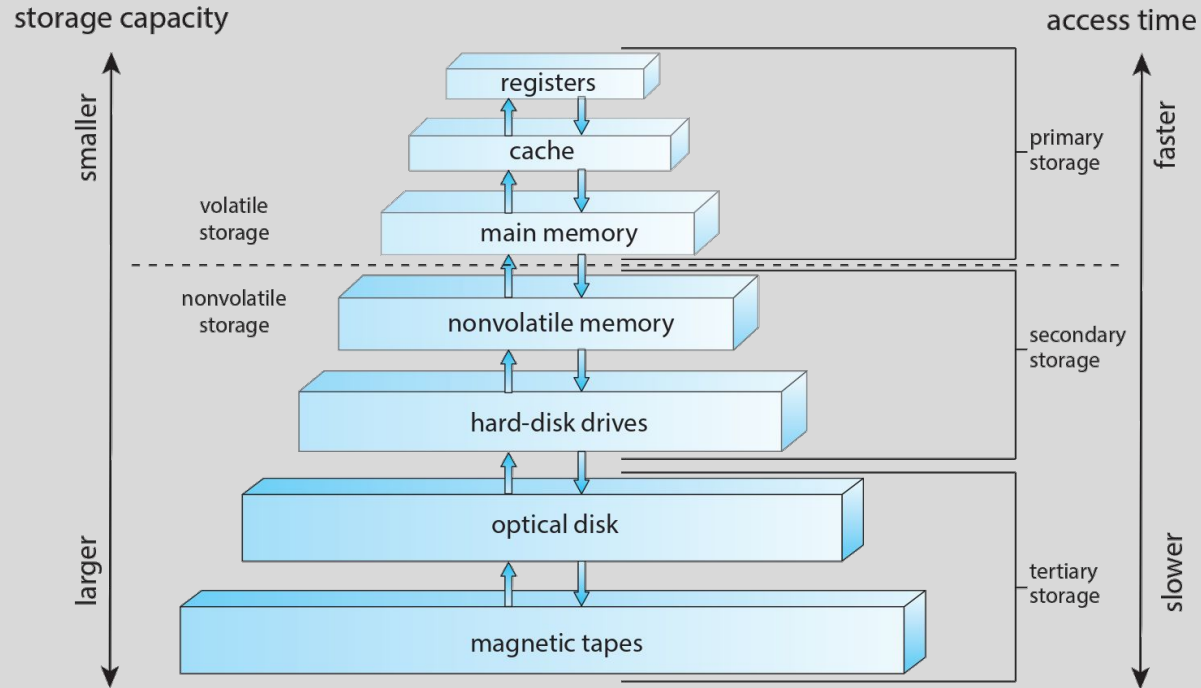
El almacenamiento secundario también es mucho más lento que la memoria principal.

También existen tipos de **almacenamiento terciario**, utilizado principalmente para Backups. Donde el costo por byte es mucho menor, con lo cual la capacidad de almacenamiento es alta, pero pagan el costo de ser mucho más lentas.

ESTRUCTURA DE ALMACENAMIENTO

Cada sistema de almacenamiento proporciona las funciones básicas de almacenar datos y conservarlos hasta su recuperación posterior. Las principales diferencias entre los distintos sistemas de almacenamiento residen en la **velocidad**, el **tamaño** y la **volatilidad**.

Como regla general, existe un equilibrio entre tamaño y velocidad, con una memoria más pequeña y más rápida más cerca de la CPU.





ESTRUCTURA DE E/S

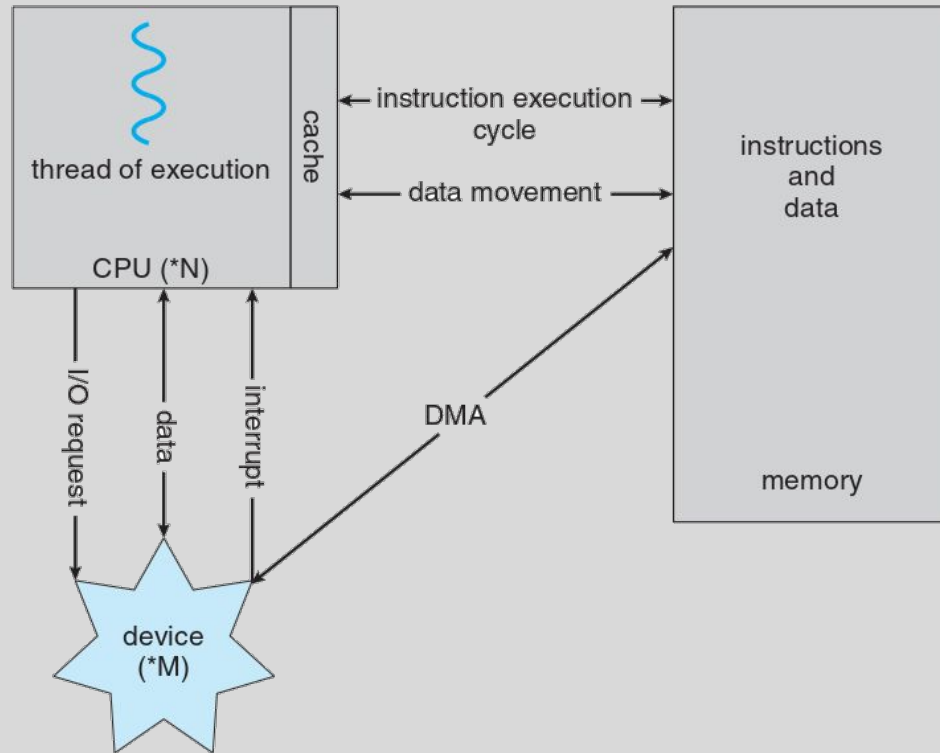
¿Cómo se gestiona la Entrada y Salida de información de un SO?

Una gran parte del código de un sistema operativo se dedica a la gestión de E/S, tanto por su importancia para la confiabilidad y el rendimiento del sistema, como también por la naturaleza variable de los dispositivos.

El tipo de E/S basada en interrupciones que mencionamos anteriormente, es adecuado para transferir pequeñas cantidades de datos, pero puede generar una alta sobrecarga cuando se utiliza para el movimiento masivo de datos.

Para solucionar este problema, se utiliza el acceso directo a memoria (DMA).

ESTRUCTURA E/S



Direct Memory Access (DMA): es la capacidad del *device controller* de transferir un bloque completo de datos directamente **hacia o desde** el dispositivo y la memoria principal, sin intervención de la CPU.

Solo se genera una interrupción por bloque para indicar al *device driver* que la operación se ha completado.

Mientras el *device controller* realiza estas operaciones, la CPU está disponible para realizar otras tareas.

ARQUITECTURA DEL SISTEMA





SINGLE PROCESSOR SYSTEMS

Los sistemas **Single-Processor** cuentan con una CPU principal es capaz de ejecutar un conjunto de instrucciones de **propósito general**, incluyendo instrucciones de procesos.

Estos sistemas también cuentan con otros procesadores de propósito especial. Pueden ser procesadores específicos para cada dispositivo, como controladores de disco, teclado y gráficos. Todos estos procesadores de propósito especial ejecutan un conjunto limitado de instrucciones y no ejecutan procesos. A veces, son gestionados por el propio SO, que les envía información sobre su próxima tarea y supervisa su estado.

Por ejemplo, un microprocesador controlador de disco recibe una secuencia de solicitudes de la CPU e implementa su propia cola y algoritmo de programación. Esta configuración libera a la CPU principal de la sobrecarga que supone la programación de disco.

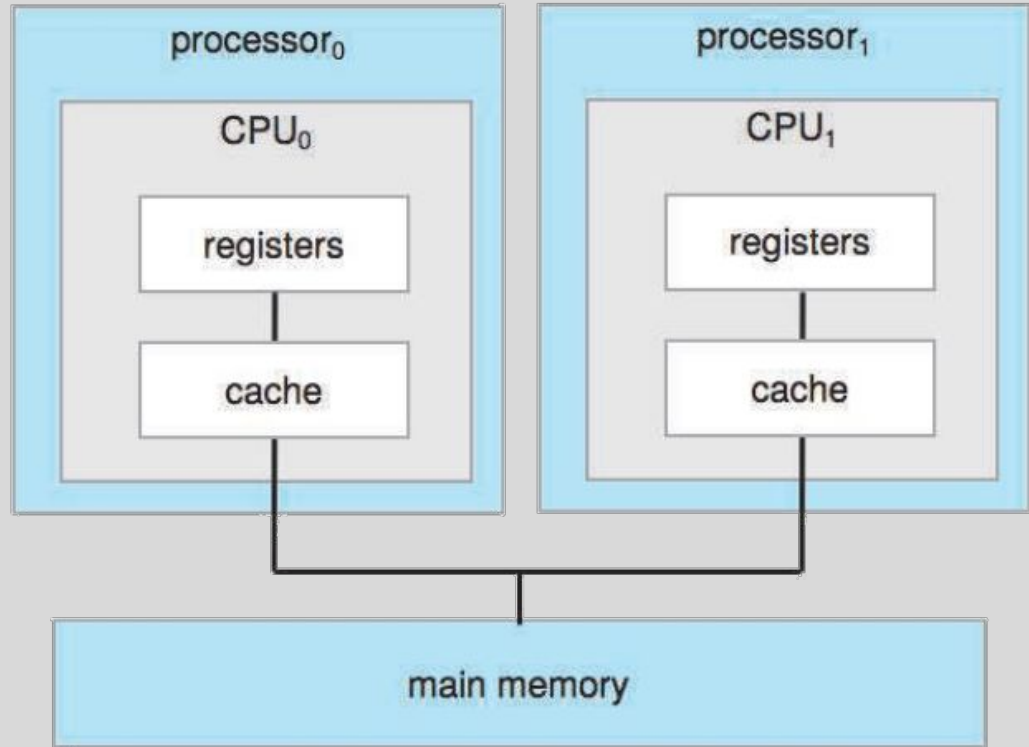
El uso de microprocesadores de propósito especial es común y no convierte un sistema de un solo procesador en uno multiprocesador. Si solo hay una CPU de **propósito general** con un solo núcleo de procesamiento, entonces el sistema es un sistema de un solo procesador.

MULTI PROCESSOR SYSTEMS

Antes, estos sistemas contaban con dos (o más) procesadores, cada uno con una CPU de **un solo** núcleo.

La principal ventaja de estos sistemas es el aumento del **throughput**. Es decir, al aumentar el número de procesadores, se espera realizar más trabajo en menos tiempo.

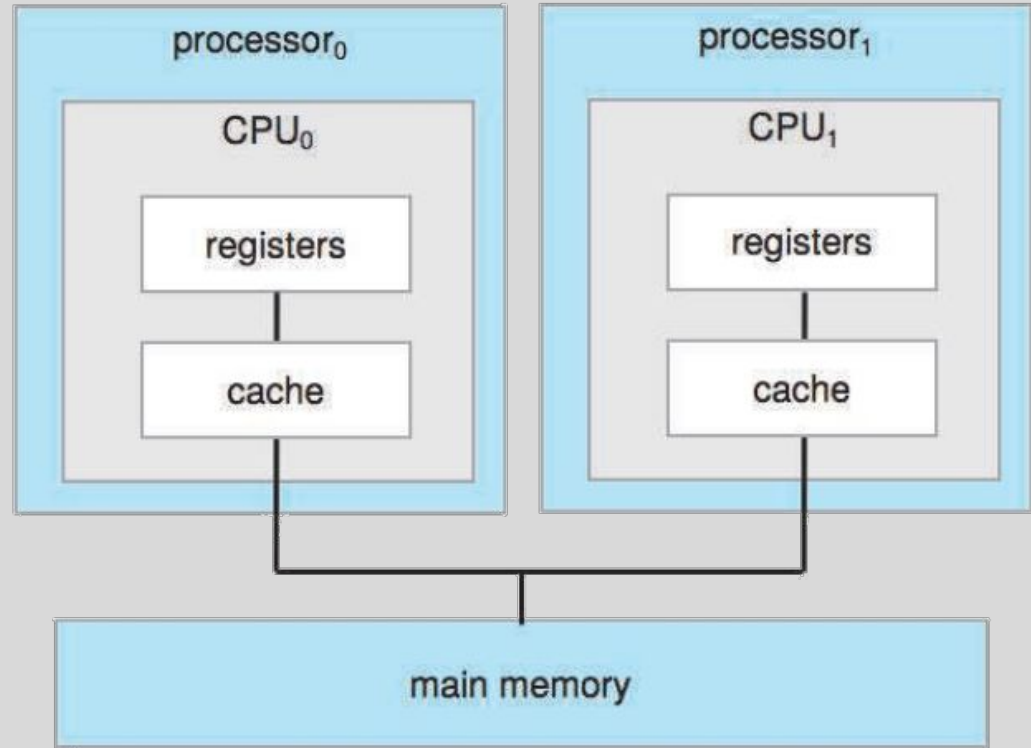
Sin embargo, la tasa de aumento con N procesadores **no** es N; **es menor** que N.



MULTI PROCESSOR SYSTEMS

Cuando varios procesadores cooperan en una tarea, se genera una cierta sobrecarga para mantener el correcto funcionamiento de todos los componentes, por ejemplo tareas de coordinación, locks de recursos, etc. Esta sobrecarga reduce la ganancia esperada de procesadores adicionales.

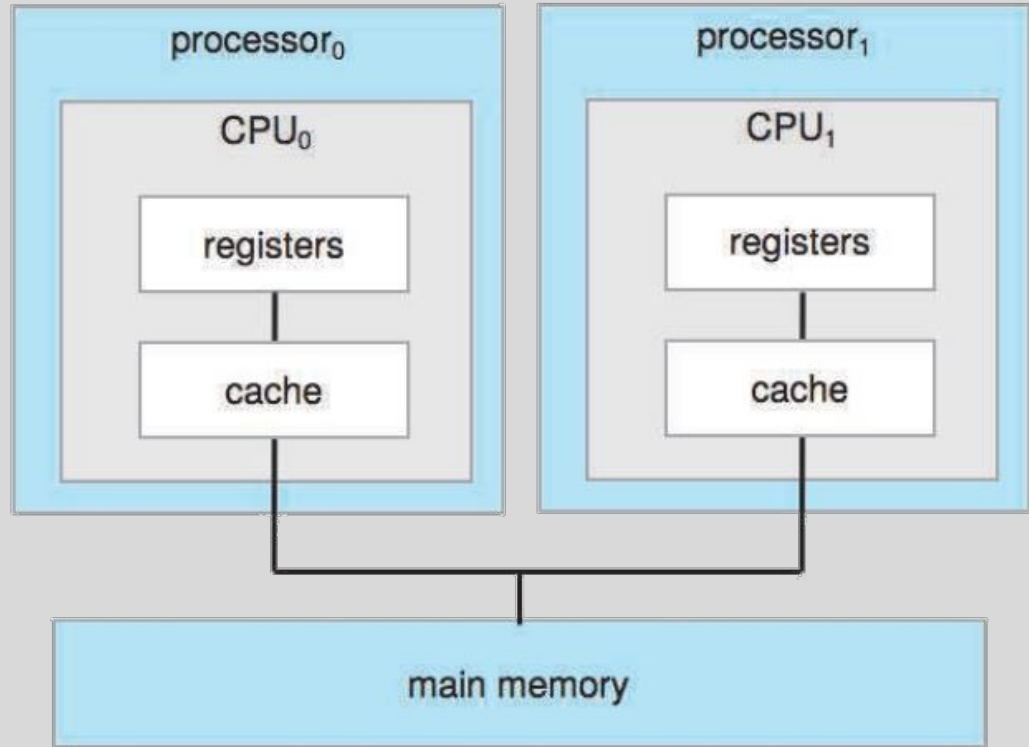
Estos sistemas utilizan multiprocesamiento simétrico (SMP), donde cada procesador realiza todas las tareas, incluyendo las funciones del SO y los procesos de usuario. Notar que cada procesador tiene su propio conjunto de registros y caché local. Y que todos comparten la memoria principal.



MULTI PROCESSOR SYSTEMS

La ventaja de este modelo es que muchos procesos pueden ejecutarse simultáneamente sin que el rendimiento se deteriore significativamente.

Sin embargo, como las CPU están separadas, una puede estar inactiva mientras otra está sobrecargada. Esto puede evitarse si los procesadores comparten ciertas estructuras de datos. Un sistema multiprocesador de este tipo permitirá que los procesos y recursos, como la memoria, se compartan dinámicamente entre los distintos procesadores y puede reducir la varianza de la carga de trabajo entre ellos.

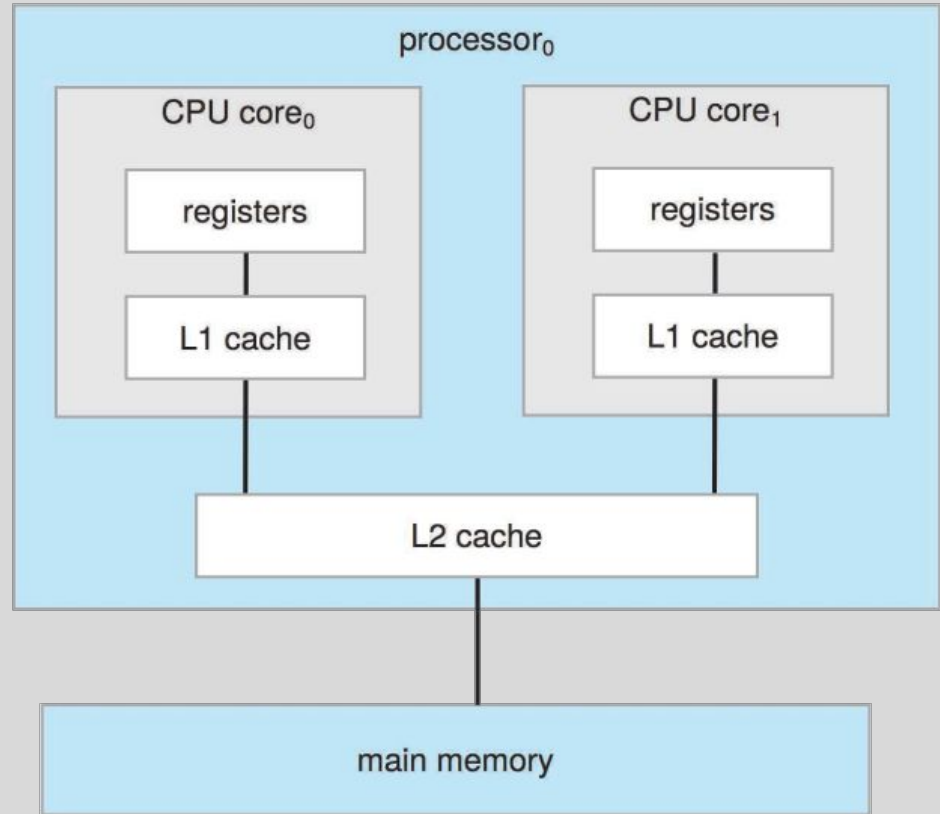


MULTI PROCESSOR SYSTEMS

Esto evolucionó con el tiempo y ahora tenemos sistemas **multinúcleo**, en los que varios núcleos de procesamiento residen en un solo chip. Estos sistemas suelen ser más eficientes porque la comunicación dentro del chip es más rápida que la comunicación entre chips.

En este diseño, cada núcleo tiene su propio conjunto de registros, así como su propia caché local, caché de nivel 1 o L1. La caché de nivel 2 (L2) es local en el chip, pero compartida por los dos núcleos de procesamiento.

Un procesador multinúcleo con N núcleos se presenta al sistema operativo como N CPU normales.

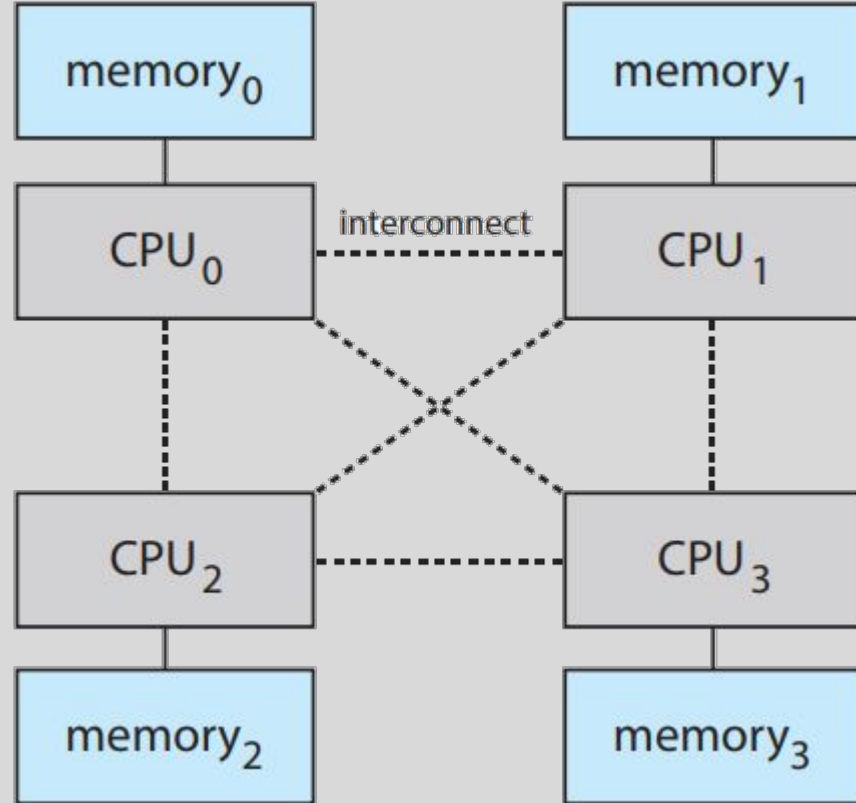


MULTI PROCESSOR SYSTEMS

Añadir CPU adicionales de esta manera es un concepto que no escala muy bien, esto convierte al bus del sistema en un cuello de botella y el rendimiento comienza a degradarse.

Un enfoque alternativo consiste en proporcionar a cada CPU su propia memoria local, a la que se accede mediante un bus local pequeño y rápido.

Las CPU están conectadas mediante una interconexión de sistema compartida, de modo que todas comparten un espacio de direcciones físicas. Este enfoque, se conoce como **non-uniform memory access (NUMA)**.

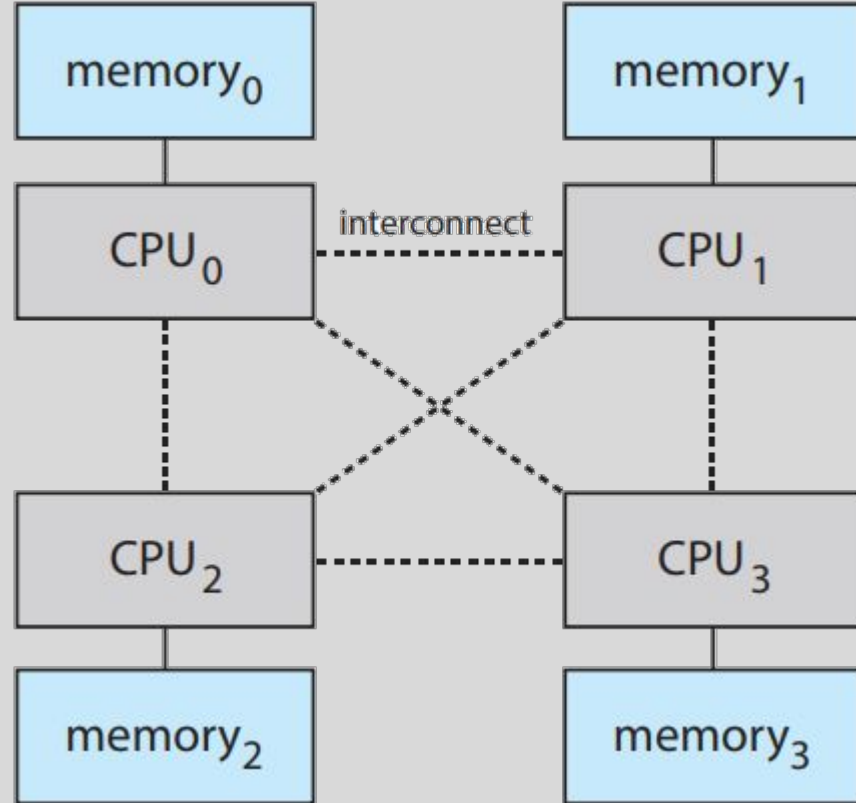


MULTI PROCESSOR SYSTEMS

Una posible desventaja de NUMA es el aumento de la latencia cuando una CPU debe acceder a la memoria no local.

Por ejemplo, la CPU0 no puede acceder a la memoria local de la CPU3 con la misma rapidez con la que accede a su propia memoria local.

De todas formas, los sistemas NUMA pueden escalar para admitir una gran cantidad de procesadores, por esto son cada vez más populares en sistemas de alto rendimiento.





OPERACIONES DE SISTEMAS OPERATIVOS



OPERACIONES DE SISTEMAS OPERATIVOS DESDE SU INICIO HASTA LA ESPERA DE UN EVENTO

Para que una computadora comience a funcionar, necesita un programa inicial o **bootstrap program**, que se almacena en el firmware del hardware de la computadora. Se encarga de inicializar todos los aspectos del sistema, desde los registros de la CPU hasta los device controllers y el contenido de la memoria, y debe saber cómo cargar el SO y cómo iniciar su ejecución. Para esto, debe localizar el **kernel** y cargarlo en la memoria.

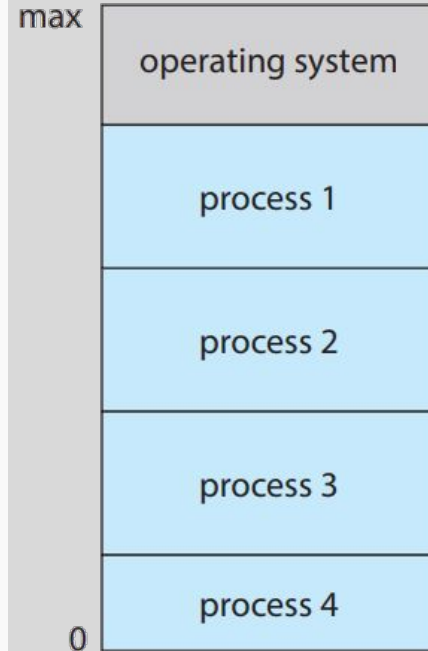
Una vez cargado y en ejecución, el **kernel** puede comenzar a proporcionar servicios al sistema y a sus usuarios. Algunos servicios son proporcionados fuera del kernel por programas del sistema, los **daemons** del sistema. Cuando terminan de cargar los daemons, el sistema se inicia completamente y espera a que ocurra algún evento. Los eventos casi siempre se indican mediante la ocurrencia de una interrupción. Algunas son generadas por HW (un acceso no válido a memoria) y otras por SW, una solicitud de un programa de usuario para que se utilice un servicio del SO mediante la ejecución de una operación especial denominada **system call**.

Así, básicamente un sistema operativo proporciona las operaciones necesarias para generar el entorno adecuado para la ejecución de los programas.

MULTIPROGRAMMING

Uno de los aspectos más importantes de los SO es la capacidad de ejecutar múltiples programas. La multiprogramación (**multiprogramming**) aumenta la utilización de la CPU al organizar los programas de forma tal que la CPU siempre tenga uno para ejecutar. Un programa en ejecución se denomina **proceso**.

La idea es la siguiente: el SO mantiene en memoria varios procesos disponibles para su ejecución simultáneamente, selecciona uno y comienza a ejecutarlo. Eventualmente, el proceso puede tener que esperar a que se complete alguna tarea, como una operación de E/S. En ese punto, el SO simplemente selecciona a otro proceso disponible y lo ejecuta. Cuando ese proceso necesita esperar, la CPU cambia a otro proceso, y así sucesivamente. Finalmente, el primer proceso termina de esperar y recupera la CPU. Mientras al menos un proceso necesite ejecutarse, la CPU nunca estará inactiva.





MULTITASKING

La multitarea (**multitasking**) es una extensión lógica de *multiprogramming*. Es la capacidad de un sistema operativo para gestionar la ejecución simultánea de múltiples tareas o procesos.

Esto se logra dividiendo el tiempo de la CPU en intervalos breves y asignándolos a cada tarea activa, alternando rápidamente entre ellas.

Esta conmutación rápida crea la ilusión de que las tareas se ejecutan al mismo tiempo.

Si varios procesos están listos para ejecutarse simultáneamente, el sistema debe elegir cuál se ejecutará a continuación. Esta decisión se conoce como **CPU scheduling**.



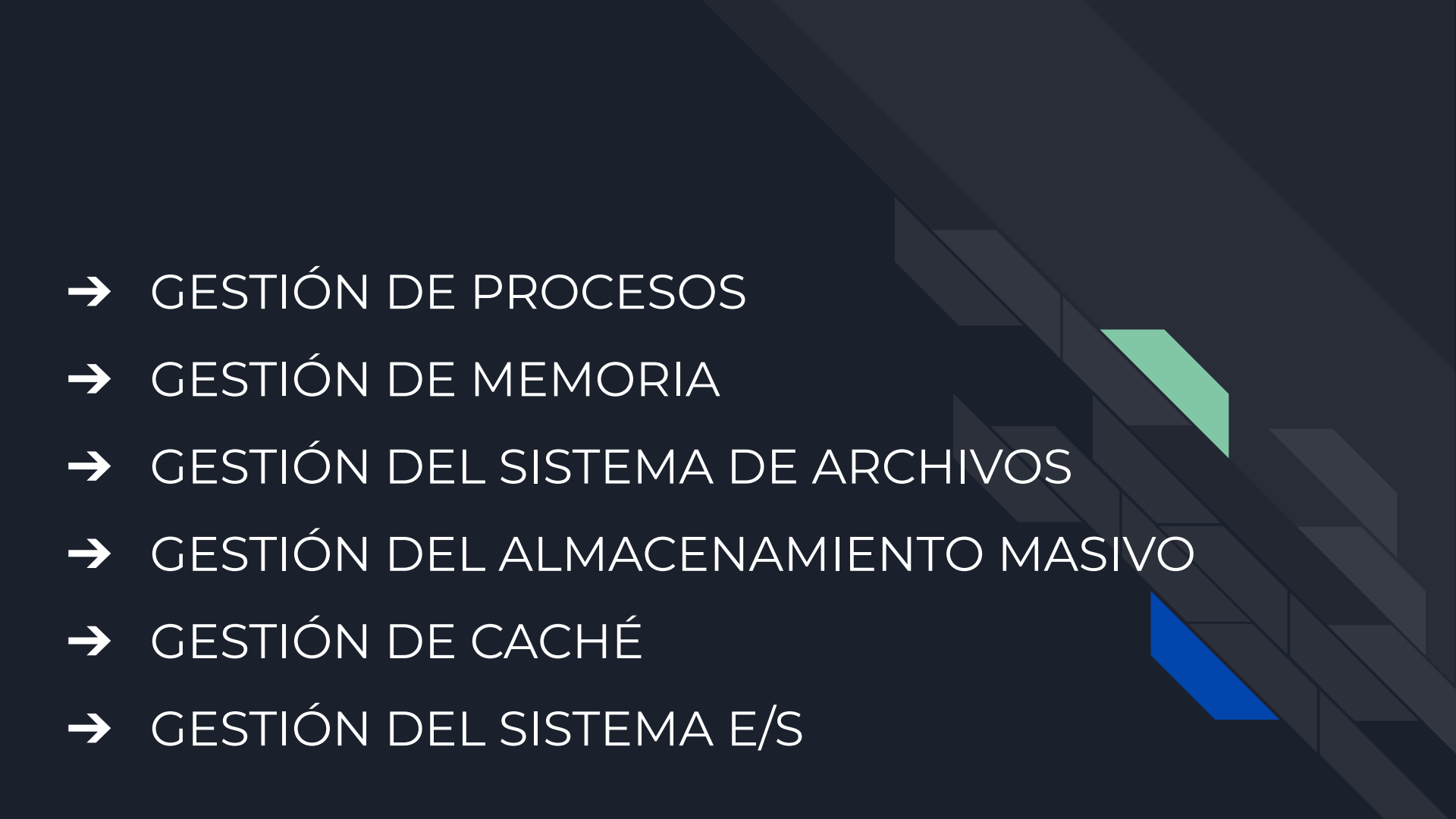
MULTIPROGRAMMING AND MULTITASKING

Ejecución: En *multiprogramming*, aunque varios programas residen en la memoria, solo uno se ejecuta en un momento dado. En *multitasking*, múltiples tareas pueden ejecutarse **aparentemente** de manera simultánea mediante la asignación temporal de la CPU.

Objetivo: *Multiprogramming* busca **maximizar la utilización de la CPU** manteniéndola ocupada la mayor parte del tiempo posible; *multitasking* pretende mejorar la capacidad de respuesta del sistema al usuario, permitiendo la ejecución concurrente de varias aplicaciones.

GESTIÓN DE RECURSOS



- 
- GESTIÓN DE PROCESOS
 - GESTIÓN DE MEMORIA
 - GESTIÓN DEL SISTEMA DE ARCHIVOS
 - GESTIÓN DEL ALMACENAMIENTO MASIVO
 - GESTIÓN DE CACHÉ
 - GESTIÓN DEL SISTEMA E/S



GESTIÓN DE PROCESOS

Un programa en ejecución, es un **proceso**. Y un proceso es la **unidad de trabajo** de un sistema. Se puede considerar un proceso como una instancia de un programa en ejecución, pero el concepto es más general.

Un proceso necesita ciertos **recursos**, tiempo de CPU, memoria, archivos, etc, para realizar su tarea. Estos recursos se asignan al proceso mientras se ejecuta. Y al finalizar, el SO recuperará los recursos reutilizables.

Es importante mencionar que un programa en sí mismo no es un proceso. Un programa es una entidad **pasiva**, mientras que un proceso es una entidad **activa**.

La ejecución de un proceso debe ser **secuencial**. La CPU ejecuta una instrucción del proceso tras otra, hasta que este se completa. En cualquier momento, se ejecuta como máximo **una** instrucción en nombre del proceso.



GESTIÓN DE PROCESOS

Un sistema consta de un conjunto de procesos, algunos son del SO y el resto son del usuario. Todos estos procesos pueden ejecutarse simultáneamente o en paralelo en varios núcleos de CPU.

El SO es responsable de las siguientes actividades relacionadas con la gestión de procesos:

- Crear y eliminar procesos de usuario y del sistema.
- Programar procesos e hilos en las CPU.
- Suspender y reanudar procesos.
- Proporcionar mecanismos para la sincronización de procesos.
- Proporcionar mecanismos para la comunicación entre procesos.



GESTIÓN DE MEMORIA

La memoria principal es un gran conjunto de bytes, cuyo tamaño varía entre cientos de miles y miles de millones, donde cada byte **tiene su propia dirección**.

La CPU lee instrucciones de la memoria principal durante el ciclo *fetch*, y lee y escribe datos de la memoria durante el ciclo **store**. Entonces, para que la CPU procese datos del disco, estos deben transferirse primero a la memoria mediante llamadas de E/S generadas por la CPU. Del mismo modo, para que un programa se ejecute, debe asignarse a direcciones absolutas y cargarse en memoria. Cuando el programa termina, su espacio de memoria se declara disponible y otro programa puede cargarse y ejecutarse.

Para mejorar tanto la utilización de la CPU como la velocidad de respuesta, las computadoras deben mantener varios programas en memoria, lo que crea la necesidad de gestionar la memoria. Existen muchos esquemas diferentes de gestión de memoria. Estos reflejan diversos enfoques, y la eficacia de cualquier algoritmo depende de la situación. Cada algoritmo requiere su propio soporte de hardware.



GESTIÓN DE MEMORIA

El sistema operativo es responsable de las siguientes actividades relacionadas con la gestión de memoria:

- Controlar qué partes de la memoria se están utilizando actualmente y qué proceso las está utilizando.
- Asignar y liberar espacio de memoria según sea necesario.
- Decidir qué procesos (o partes de procesos) y datos se moverán dentro y fuera de la memoria.



GESTIÓN DEL SISTEMA DE ARCHIVOS

El SO se basa en las propiedades físicas de sus dispositivos de almacenamiento para definir una unidad lógica de almacenamiento: el archivo. El SO mapea archivos a medios físicos y accede a ellos a través de los dispositivos de almacenamiento. Comúnmente, los archivos representan programas y datos. Y normalmente se organizan en directorios para facilitar su uso.

Las computadoras pueden almacenar información en varios tipos de medios físicos. Cada uno de estos medios tiene sus propias características y organización física. Estas propiedades incluyen: velocidad de acceso, capacidad, velocidad de transferencia de datos y método de acceso (secuencial o aleatorio).

Además, cuando existen varios usuarios, puede ser conveniente controlar quién puede acceder a un archivo y cómo lo hace (por ejemplo, leer, escribir, añadir).



GESTIÓN DEL SISTEMA DE ARCHIVOS

El SO es responsable de las siguientes actividades relacionadas con la gestión de archivos:

- Crear y eliminar archivos.
- Crear y eliminar directorios.
- Administrar las operaciones para la manipulación de archivos y directorios.
- Asignar archivos a almacenamiento masivo.
- Realizar copias de seguridad de archivos en medios de almacenamiento estables (no volátiles).



GESTIÓN DEL ALMACENAMIENTO MASIVO

Los SO deben proporcionar **almacenamiento secundario** para respaldar la memoria principal. La mayoría de los programas, se almacenan en estos dispositivos hasta que se cargan en memoria. Los programas utilizan a estos dispositivos como origen y destino de su procesamiento. La *performance* de una computadora depende mucho de la velocidad del subsistema de almacenamiento secundario y de los algoritmos que lo gestionan.

El sistema operativo es responsable de las siguientes actividades relacionadas con la gestión del almacenamiento secundario:

- Montaje y desmontaje.
- Gestión del espacio libre.
- Asignación de almacenamiento.
- Administración de discos.
- Particionado.
- Protección.



GESTIÓN DEL ALMACENAMIENTO MASIVO

También existen dispositivos de almacenamiento más lento, económico y de mayor capacidad que el almacenamiento secundario, denominados **almacenamiento terciario**. Cintas magnéticas, discos de CD, DVD y Blu-ray, son dispositivos de almacenamiento terciario típicos. Y si bien no es crucial para el rendimiento del sistema, aun así debe gestionarse. Algunos SO se encargan de esta tarea, mientras que otros lo dejan en manos de las aplicaciones.



GESTIÓN DEL ALMACENAMIENTO

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape



GESTIÓN DE CACHÉ

La información normalmente se guarda en sistemas de almacenamiento persistente (y "lento"), y a medida que se necesita, se copia temporalmente a un sistema de almacenamiento más rápido (memoria principal, caché). Así, cuando necesitamos cierta información, primero comprobamos si está en la caché. Si está, la utilizamos directamente. Si no, utilizamos la información de la fuente, guardando una copia en la caché asumiendo que la necesitaremos pronto.

Algunas cachés se implementan completamente en hardware. Por ejemplo, la caché de instrucciones, que almacena las instrucciones que se espera que se ejecuten a continuación. Sin esta caché, la CPU tendría que esperar varios ciclos mientras se recuperaba una instrucción de la memoria principal.

Las cachés exclusivas de HW quedan excluidas del curso ya que están fuera del control del SO.

Por ejemplo, la transferencia de datos de la caché a la CPU y a los registros suele ser una función de HW, sin intervención del SO. En cambio, la transferencia de datos del disco a la memoria sí suele estar controlada por el SO.



GESTIÓN GESTIÓN DEL SISTEMA E/S

Un SO es responsable de ocultar al usuario las peculiaridades de dispositivos de hardware específicos. Por ejemplo, cómo transferir un dato desde el disco hasta la memoria, o cómo capturar el valor de una tecla presionada y disparar la interrupción correspondiente. Así, solo el device controller conoce las peculiaridades del dispositivo específico al que está asignado.

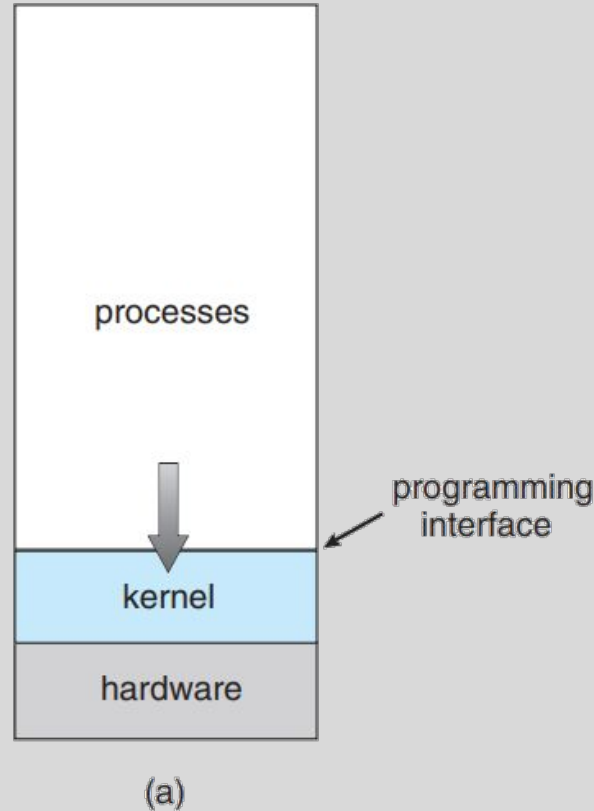
El subsistema de E/S consta de varios componentes:

- Un componente de gestión de memoria que incluye búfer, caché y spool.
- Una interfaz general entre dispositivos y controladores.
- Controladores para dispositivos de hardware específicos.

VIRTUALIZACIÓN

En términos generales, el SW de **virtualización** pertenece a una categoría que también incluye la **emulación**, que consiste en *simular* el HW de una computadora en SW. Esto es, cada instrucción que se ejecuta en el sistema origen, debe traducirse a la *función equivalente* en el sistema destino, lo que en general resulta en varias instrucciones.

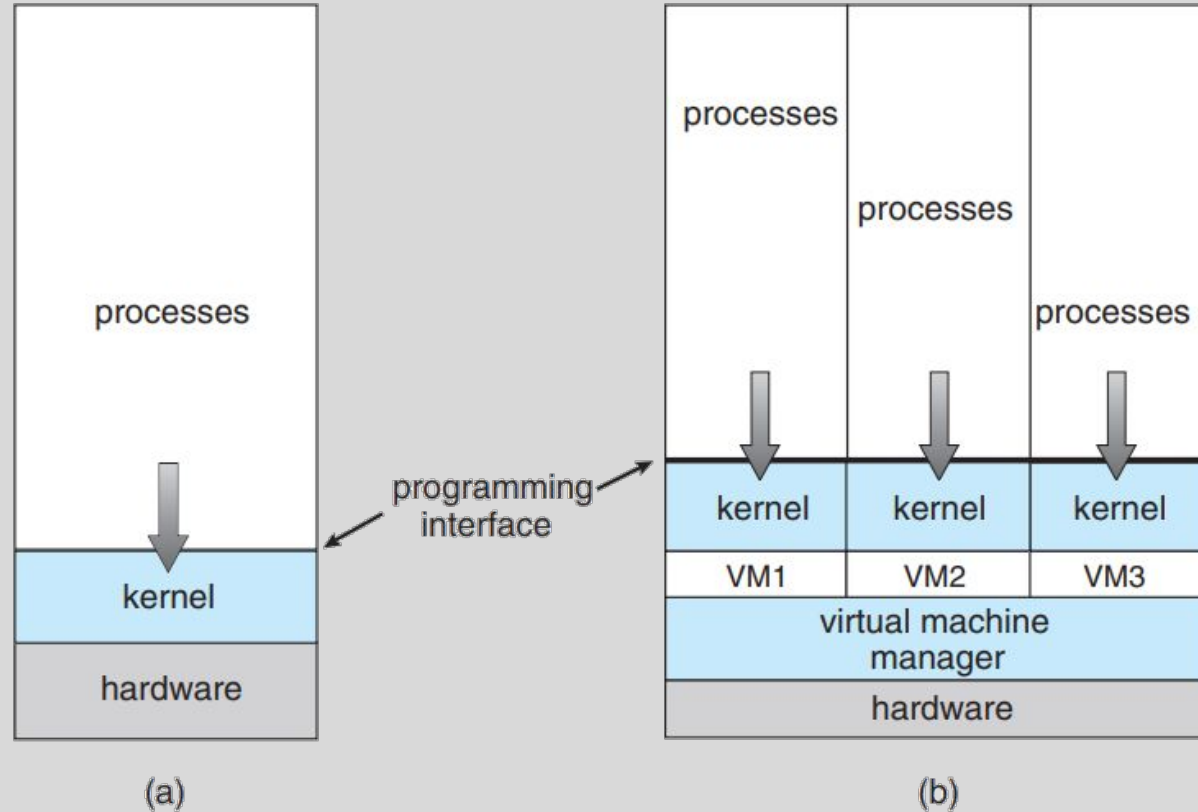
Esto es posible porque existe un principio de equivalencia entre HW y SW. Toda la lógica implementada en HW puede implementarse en SW y viceversa.



VIRTUALIZACIÓN

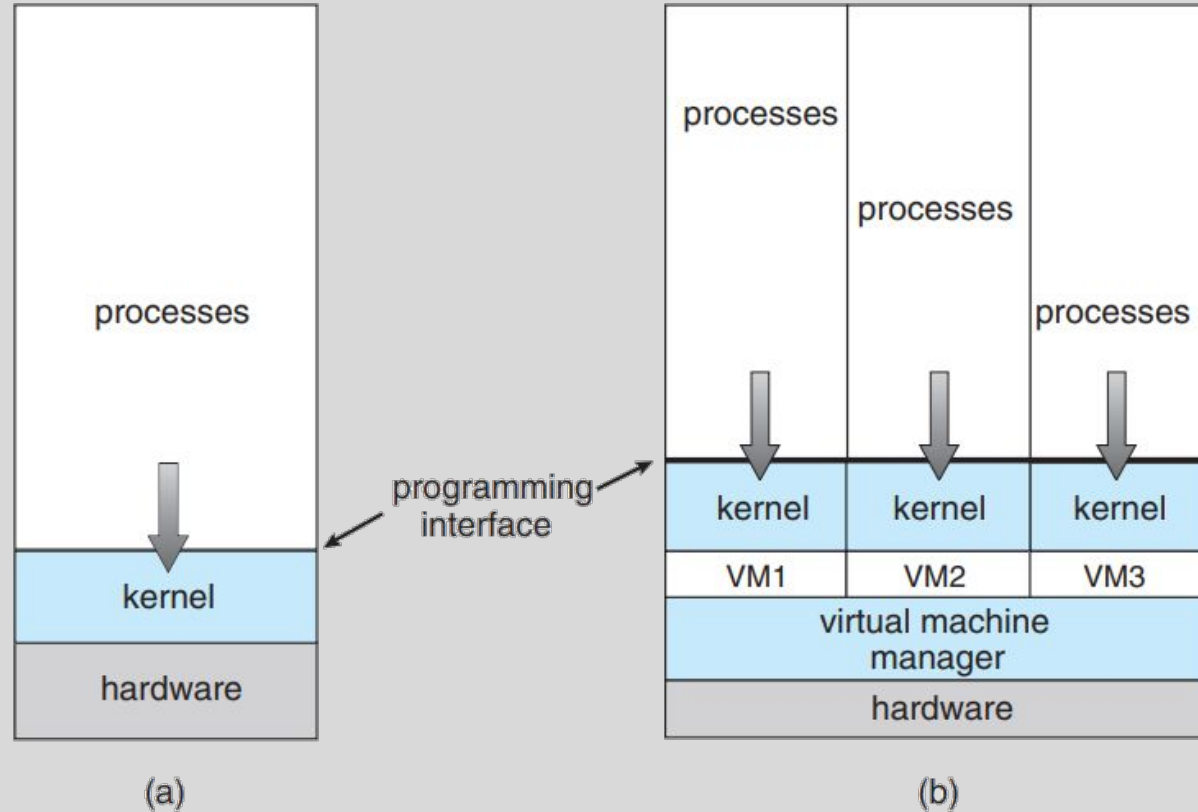
En cambio, en la virtualización, un SO compilado de forma nativa para una arquitectura de CPU concreta, se ejecuta dentro de otro SO también nativo para esa CPU.

Es una tecnología que permite abstraer el HW de una computadora (CPU, memoria, unidades de disco, tarjetas de interfaz de red, etc.) en componentes de SW, permitiendo que los SO se ejecuten como aplicaciones dentro de otros SO, y creando así la ilusión de que cada entorno se ejecuta en su propia PC.



VIRTUALIZACIÓN

La diferencia fundamental radica en que la *virtualización comparte* recursos de hardware, mientras que la *emulación simula* hardware que podría no existir físicamente en el sistema.



SISTEMAS DISTRIBUIDOS

En términos simples, una red es una ruta de comunicación entre dos o más sistemas. Las redes varían según los protocolos utilizados, las distancias entre nodos y los medios de transporte. Por ejemplo, TCP/IP es el protocolo de red más común y proporciona la arquitectura fundamental de Internet. Desde el punto de vista de un SO, solo se necesita: un protocolo de red, un dispositivo de interfaz (por ejemplo, una placa de red), un controlador para gestionarlo y un software para gestionar los datos.



SISTEMAS DISTRIBUIDOS

Un sistema distribuido es un conjunto de sistemas físicamente separados, posiblemente heterogéneos, que se conectan en red para proporcionar a los usuarios acceso a los recursos que mantiene.

El acceso a un recurso compartido aumenta la velocidad de computación, la funcionalidad, la disponibilidad de los datos y la fiabilidad.



Muchas Gracias

Jeremías Fassi

Javier E. Kinter

