Introducción
1

Sistemas operativos y distribuidos

Gustavo Distel
gd@cs.uns.edu.ar

DCIC - UNS

Introducción
Contenido

- ¿Sistemas operativos? y... ¿distribuidos?
- Organización de una computadora.
- Arquitectura del sistema informático.
- Operaciones del sistema operativo.
- Administración de recursos.
- Seguridad y protección.
- Virtualización.
- Estructuras de datos del *kernel.*
- Entornos informáticos.
- Sistemas operativos libres y de código abierto.

¿Sistemas operativos?

- Es un intermediario entre el usuario de una computadora y su *hardware*.
- Proporciona un entorno en el cual los usuarios puedan ejecutar programas de una manera práctica y eficiente.
- Es *software* que administra el *hardware* de una computadora.
- Provee las bases para los programas de aplicación.

¿Sistemas operativos?

Se encuentran en infinidad de dispositivos y en una gran variedad de formas en las que pueden llevar a cabo sus tareas

¿Sistemas operativos?

- Son de gran tamaño, complejos y generalmente se desarrollan y usan durante muchos años.
- Ej: el código fuente de Linux tiene 17 millones de líneas de código.
- En número de libros sería: si cada libro tiene 1000 páginas y cada página tiene 50 líneas → ¡alrededor de 170 libros!

¿Qué hace un sistema operativo?

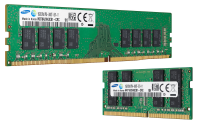
Un sistema informático puede dividirse en cuatro componentes:

- Usuarios
- Programas de aplicación: definen la forma en que se emplean los recursos para resolver los problemas informáticos.
- Sistema operativo: controla y coordina el uso del *hardware* entre los diversos programas de aplicación.
- *Hardware*: proporciona los recursos básicos de cómputo al sistema.

¿Sistemas operativos?

Vista del usuario → Varía de acuerdo con la interfaz que utilice.

Vista de la computadora → Íntimamente relacionado con el hardware. El SO es un asignador de recursos.



Definición de Sistemas Operativos

- No hay una definición aceptada.
- Una podría ser: lo que un proveedor envía cuando se solicita o compra "un sistema operativo", aunque las características varían mucho de un sistema a otro.
- Otra definición común es: un sistema operativo es aquel programa que se ejecuta en todo momento en la computadora (*kernel*). Todo lo demás son:
  - Programas del sistema (viene con el sistema operativo, pero no es parte del *kernel*), o
  - Programas de aplicación (todos los programas que no están asociados con el sistema operativo).

¿Por qué estudiar Sistemas Operativos?

- Aunque hay muchos profesionales de Informática, solo un pequeño porcentaje de ellos participa en la creación o modificación de un sistema operativo. ¿Por qué, entonces, estudiar los SO y cómo funcionan?
  - Porque, como casi todo el código se ejecuta sobre un SO, el conocimiento de cómo funcionan los sistemas operativos es crucial para una programación adecuada, eficiente, efectiva y segura.
    - Programación concurrente.
    - Administración de recursos.
    - Análisis de rendimiento.
    - Interfaces.
    - Lidiar con un *software* grande.
    - Y mucho más...

Y... ¿Sistemas Distribuidos?

- Colección de sistemas informáticos físicamente separados y posiblemente heterogéneos, que están conectados en red para proporcionar a los usuarios acceso a los recursos que el sistema mantiene.
  - El acceso a un recurso compartido aumenta la velocidad de computación, la funcionalidad, la disponibilidad de datos y la confiabilidad.
  - Los sistemas distribuidos dependen de la red para su funcionamiento
- Red: ruta de comunicación entre dos o más sistemas.
  - Protocolo TCP/IP y redes LAN, WAN.
- Un sistema operativo de red actúa de forma autónoma, es consciente de la red y puede comunicarse con otras computadoras en red.
- Un sistema operativo distribuido proporciona un entorno menos autónomo; las diferentes computadoras generan la ilusión de que un solo sistema operativo controla la red.

¿Por qué estudiar Sistemas Distribuidos?

Ejemplos

Redes de telecomunicación:
- Redes telefónicas y redes celulares
- Redes informáticas (ej.: Internet)
- Redes inalámbricas de sensores

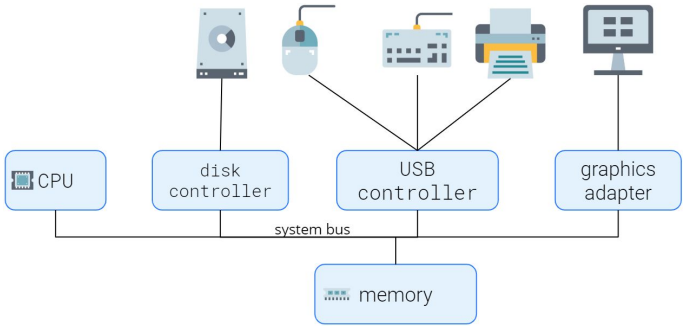
Aplicaciones de red:
- *World Wide Web* y redes *peer-to-peer*
- Juegos *online* multijugador masivos y comunidades de realidad virtual
- Bases de datos distribuidas y sistemas de gestión de bases de datos distribuidas
- Sistemas de archivos de red
- Sistemas de procesamiento de información distribuidos (ej.: sistemas bancarios y sistemas de reserva de líneas aéreas)

Control de procesos en tiempo real:
- Sistemas de control de aviones
- Sistemas de control industrial

Computación paralela:
- Computación científica (Computación en clúster y *grid computing*, proyectos de computación voluntaria)
- Representación distribuida en gráficos de computadora

Organización de una computadora

- Un sistema informático de propósito general consiste en una o más *CPUs* y varios controladores de dispositivos conectados a través de un *bus* común, que proporciona acceso entre los componentes y la memoria compartida.

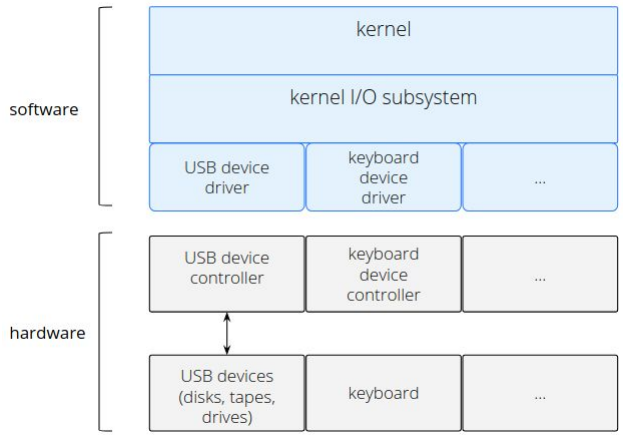


Organización de una computadora

- Cada *device controller* (controlador del dispositivo) está a cargo de un tipo específico de dispositivo (ej.: una unidad de disco, dispositivo de audio, pantalla gráfica).
- Un *device controller* mantiene algunos *buffers* locales y un conjunto de registros de propósito especial. A su vez es responsable de mover los datos entre los dispositivos periféricos que controla y su *buffer* local.
- Normalmente, los SO tienen un *device driver* por cada *device controller*, el cual lo interpreta y proporciona al resto del SO una interfaz uniforme con el dispositivo.
- La *CPU* y los *device controller* 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.

Organización de una computadora

Estructura del kernel I/O



Organización de una computadora

Interrupciones

- Consideremos un programa que realiza una E/S
  - El *device driver* carga los registros apropiados en el *device controller*.
  - El *device controller* examina el contenido de estos registros para determinar qué acción tomar (por ej.: leer carácter).
  - El *device controller* inicia la transferencia de datos desde el dispositivo a su *buffer* local.
  - Una vez que se completa la transferencia de datos, el *device controller* informa al *device driver* que ha finalizado su operación.
  - Luego el *device driver* le da control a otras partes del sistema operativo (devolviendo los datos o un puntero a los datos si la operación fue una lectura).
- ¿Cómo informa el controlador al *device driver* que ha finalizado su operación?
  - A través de una interrupción.

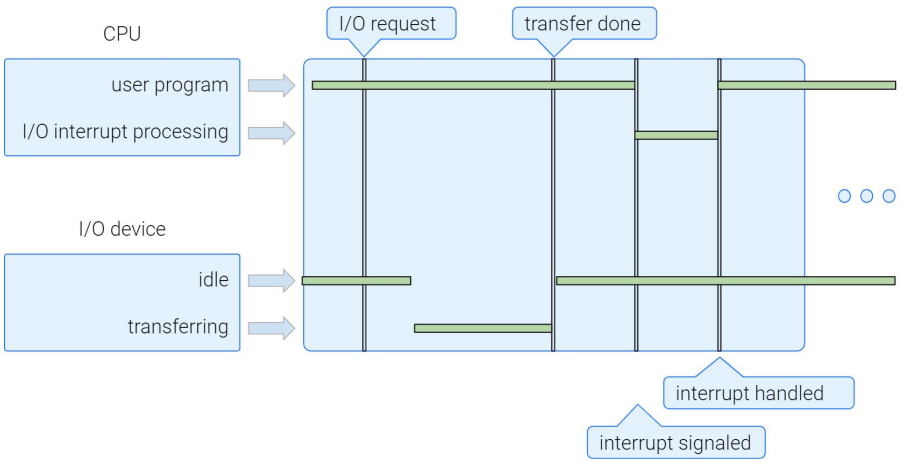
Organización de una computadora

Interrupciones

- El *hardware* puede desencadenar una interrupción en cualquier momento enviando una señal a la *CPU*, generalmente a través del *bus* del sistema.
- Cuando se interrumpe la *CPU*, se detiene lo que está haciendo e inmediatamente transfiere la ejecución a una ubicación fija.
- La ubicación fija contiene la dirección de inicio donde se encuentra la rutina de servicio para la interrupción.
- La rutina de servicio de interrupción se ejecuta y al finalizar la *CPU* reanuda el cálculo interrumpido.

Organización de una computadora

Interrupciones - Línea de tiempo



Organización de una computadora

Interrupciones

- Dado que ocurren con mucha frecuencia deben manejarse rápidamente.
- Para proporcionar la velocidad necesaria se puede usar una tabla de punteros para las rutinas de interrupción.
- La rutina de interrupción se llama indirectamente a través de la tabla, sin necesidad de una rutina intermedia.
- La tabla de punteros se almacena en poca memoria (primeras 100 ubicaciones). Estas ubicaciones contienen las direcciones de las rutinas de servicio de interrupción para los distintos dispositivos.
- Esta tabla de direcciones se denomina vector de interrupción y se indexa por un número único, asignado con la solicitud de interrupción, para proporcionar la dirección de la rutina de servicio de interrupción.
- Windows y Unix lo realizan de esta manera.

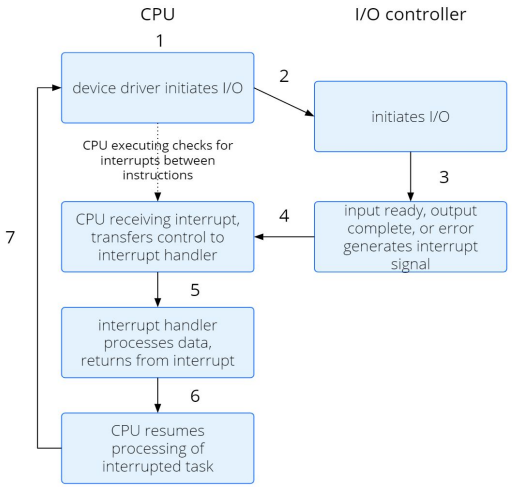
Organización de una computadora

Interrupciones - Implementación

- El hardware de la *CPU* tiene un cable llamado línea de solicitud de interrupción (*interrupt-request line*), que la *CPU* detecta luego de ejecutar cada instrucción.
- Cuando la *CPU* detecta una señal, lee el número de interrupción y salta a la rutina del manejador de interrupciones utilizando ese número como un índice en el vector de interrupción.
- Luego comienza la ejecución en la dirección asociada con ese índice.
- El controlador de interrupciones: guarda cualquier estado que cambiará durante su operación, determina la causa de la interrupción, realiza el procesamiento necesario, realiza una restauración de estado y ejecuta una instrucción de `retorno_de_interrupción` para devolver a la *CPU* al estado de ejecución antes de la interrupción.

Organización de una computadora

Ciclo - manejo de interrupción de E/S



Organización de una computadora

Interrupciones - Implementación

- En un SO moderno necesitamos funciones de manejo de interruptores más sofisticadas:
  - La posibilidad de demorar el manejo de interrupciones durante un procesamiento crítico.
  - Una forma eficiente de despachar al manejador de interrupciones.
  - Interrupciones de varios niveles, para que el SO pueda distinguir entre interrupciones de prioridad alta y baja y pueda responder con el grado de urgencia adecuado.
- En el *hardware* de una computadora moderna, estas tres características son proporcionadas por la *CPU* y el *hardware* del controlador de interrupción.
- La mayoría de las *CPU* tienen dos líneas de solicitud de interrupción.
  - Interrupción no enmascarable: reservada para eventos tales como errores de memoria no recuperables.
  - Interrupción enmascarable: puede ser desactivada por la *CPU* antes de la ejecución de secuencias de instrucciones críticas que no deben interrumpirse.

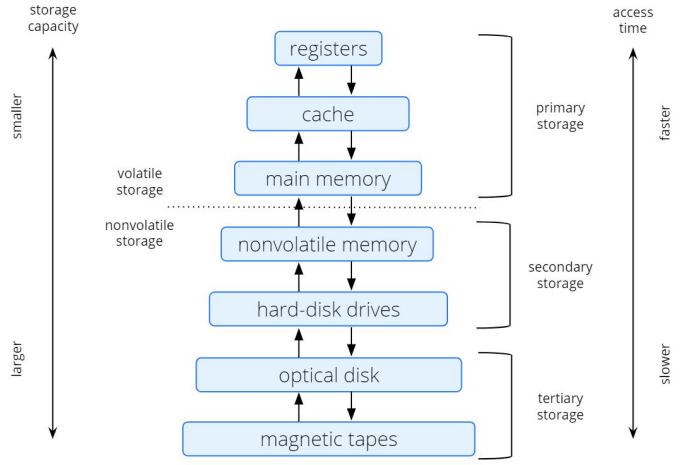
Organización de una computadora

Estructura de almacenamiento

- Memoria principal (*RAM*):
  - La *CPU* carga instrucciones sólo desde la memoria, por lo tanto cualquier programa debe cargarse primero en memoria para ejecutarse (ciclo de instrucción-ejecución)
  - La memoria principal suele ser pequeña para almacenar todos los programas y datos necesarios de forma permanente. Y, a su vez, es volátil.
- Memoria secundaria:
  - Extensión de la memoria principal, ejemplo: HDD/SSD/flash memory/Cinta.
  - Almacena grandes cantidades de datos de forma permanente.
- La amplia variedad de sistemas de almacenamiento se puede organizar en una jerarquía de acuerdo con la capacidad de almacenamiento y el tiempo de acceso.
- Regla general: existe una relación entre el tamaño y la velocidad (memoria más pequeña y más rápida cerca de la *CPU*). Y existe otra variable a considerar: el costo.

Organización de una computadora

Jerarquía de dispositivos de almacenamiento



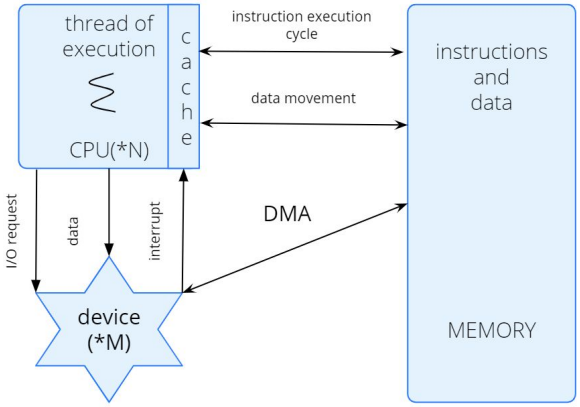
Organización de una computadora

Estructura E/S

- La E/S controlada por interrupción es apropiada para mover pequeñas cantidades de datos, pero puede generar una sobrecarga cuando se utiliza para el movimiento de datos como NVS I/O.
- Para esto se utiliza el acceso directo a la memoria (DMA).
  - Después de configurar *buffers*, punteros y contadores para el dispositivo de E/S, el controlador del dispositivo transfiere un bloque completo de datos directamente desde o hacia el dispositivo y la memoria principal, sin la intervención de la CPU.
- Sólo se genera una interrupción por bloque para indicar al controlador de dispositivo que la operación se ha completado, en lugar de una interrupción por byte.
- Mientras el controlador del dispositivo realiza estas operaciones, la *CPU* está disponible para realizar otros trabajos.

Organización de una computadora

Estructura E/S



How a modern computer system works.

Arquitectura del sistema informático

Algunas definiciones

- *CPU*: *hardware* que ejecuta las instrucciones.
- Procesador (*Processor*): chip físico que contiene una o más *CPU*.
- Núcleo (*Core*): unidad de cómputo básica de la *CPU*.
- Múltiples núcleos (*Multicore*): incluye varios cores en la misma *CPU*.
- Multiprocesador (*Multiprocessor*): incluye varios procesadores.
- Aunque actualmente la mayoría los sistemas son multinúcleo, se usa el término general *CPU* para referirse a una sola unidad computacional de un sistema de cómputo y *core* y *multicore* cuando nos referimos específicamente a uno o más *cores* en una *CPU*.

Arquitectura del sistema informático

Sistemas de un solo procesador

- Antiguamente, la mayoría de las computadoras utilizaban un único procesador que contenía una *CPU* con un solo *core*.
- El *core* es el componente que ejecuta instrucciones y tiene registros para almacenar datos localmente. La *CPU* con su *core* es capaz de ejecutar un conjunto de instrucciones de propósito general, incluidas las instrucciones de los procesos.
- Estos sistemas también tienen otros procesadores de propósito especial, que pueden venir en forma de procesadores específicos del dispositivo (controladores de disco, teclado y gráficos).
- Estos procesadores de propósito especial pueden ser o no administrados por el sistema operativo y no convierten un sistema de un único procesador en multiprocesador.

Arquitectura del sistema informático

Sistemas multiprocesador

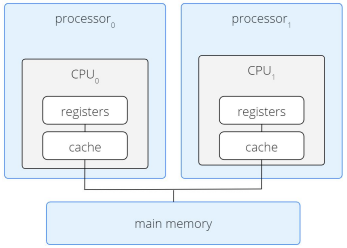
- En las computadoras actuales, desde dispositivos móviles hasta servidores, predominan los sistemas multiprocesador .
- Tradicionalmente, estos sistemas tienen dos (o más) procesadores, cada uno con una *CPU* de un solo *core*.
- La principal ventaja de los sistemas multiprocesador es el aumento del rendimiento (*throughput*): al aumentar el número de procesadores, se espera hacer más trabajo en menos tiempo.
- Sin embargo, la relación de aceleración con N procesadores no es igual a N sino menor que N. Cuando varios procesadores cooperan en una tarea, se incurre en cierta cantidad de gastos generales para que todas las piezas funcionen correctamente.
- Esta sobrecarga, sumada a la disputa por los recursos compartidos, reduce la ganancia esperada a partir de los procesadores adicionales.

Arquitectura del sistema informático

Sistemas multiprocesador

- Los sistemas multiprocesador más comunes utilizan el multiprocesamiento simétrico (SMP): cada procesador de *CPU* del mismo nivel realiza todas las tareas (incluidas funciones del SO y procesos de usuario).

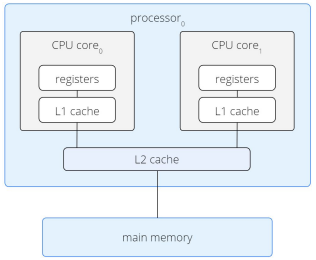
- Ventaja: muchos procesos pueden ejecutarse simultáneamente (N procesos pueden ejecutarse si hay N *CPUs*) sin que el rendimiento disminuya.
- Desventaja: dado que las *CPU* están separadas, una puede estar inactiva y la otra sobrecargada, resultando en ineficiencias. Éstas pueden evitarse si los procesadores comparten ciertas estructuras de datos.



Arquitectura del sistema informático

Sistemas multiprocesador

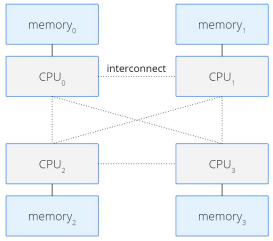
- La definición de multiprocesador ha evolucionado y actualmente incluye sistemas *multicore*, en los que múltiples *cores* informáticos residen en un solo chip.
- Los sistemas *multicore* pueden ser más eficientes que múltiples chips con *cores* únicos ya que la comunicación en un chip es más rápida que la comunicación entre chips.
- Un chip con múltiples núcleos utiliza mucha menos energía que los múltiples chips de un solo núcleo, un problema de consideración para dispositivos móviles y portátiles.
- Diseño *dual-core* con dos *cores* en el mismo chip:
  - cada núcleo tiene registros y caché L1.
  - La caché L2 es local al chip pero es compartido por los dos núcleos de procesamiento.



Arquitectura del sistema informático

NUMA (Non-Uniform Memory Access)

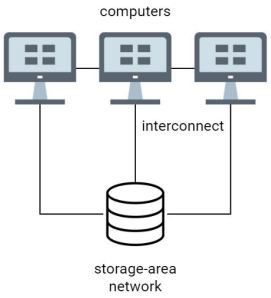
- Agregar *CPU* adicionales a un sistema multiprocesador aumentará la potencia de cálculo; sin embargo, la *escalability* no es correcta, ya que si agregamos demasiadas *CPUs*, el *bus* del sistema se convierte en un cuello de botella y el rendimiento comienza a degradarse.
- Un enfoque alternativo es proporcionar a cada *CPU* (o grupo de *CPUs*) su propia memoria local a la que se accede a través de un *bus* local pequeño y rápido. Las CPU se conectan mediante una interconexión de sistema compartida (todas las CPU comparten un espacio de direcciones físicas).
- Ventaja: cuando una *CPU* accede a su memoria local, no solo es rápida, sino que no existe disputa sobre la interconexión del sistema.
- Los sistemas *NUMA* escalan mejor a medida que se agregan más procesadores.
- Inconveniente: mayor latencia cuando una *CPU* debe acceder a la memoria remota a través de la interconexión del sistema, creando una posible penalización de rendimiento



Arquitectura del sistema informático

Cluster

- Se diferencian de los sistemas multiprocesador en que están compuestos por dos o más sistemas individuales (nodos) unidos entre sí por una red.
- Un *cluster* comparte almacenamiento y está vinculado a través de una red de área local (*LAN*).
- Se utiliza para brindar un servicio de alta disponibilidad, que continuará incluso si falla uno o más sistemas del *cluster*.
  - En general, se obtiene alta disponibilidad al agregar un nivel de redundancia en el sistema.
  - La alta disponibilidad proporciona mayor confiabilidad, crucial en muchas aplicaciones.
- Algunos sistemas son tolerantes a fallas, ya que pueden sufrir una falla de cualquier componente individual y seguir funcionando.
- La tolerancia a fallas requiere un mecanismo para permitir que la falla sea detectada, diagnosticada y, si es posible, corregida.



Arquitectura del sistema informático

Cluster

- Un *cluster* puede estructurarse de forma asimétrica o simétrica.
  - ***Cluster* asimétrico**: una máquina está en modo de espera activa mientras que la otra ejecuta las aplicaciones. La máquina de espera activa solo monitorea el servidor activo y si falla, la máquina de espera activa se convierte en el servidor activo.
  - ***Cluster* simétrico**: dos o más *hosts* ejecutan aplicaciones y se monitorean entre sí. Esta estructura es más eficiente ya que utiliza todo el hardware disponible.
- Los *cluster* también se pueden utilizar para proporcionar entornos informáticos de alto rendimiento.
- Dichos sistemas pueden proporcionar una potencia computacional significativamente mayor que los sistemas de un único procesador o incluso que los SMP, ya que pueden ejecutar una aplicación simultáneamente en todos los equipos del *cluster*.
- Sin embargo, la aplicación debe haberse escrito específicamente para aprovechar el *cluster*.

Operaciones del sistema operativo

- Para que una computadora comience a funcionar, cuando se enciende o reinicia, necesita tener un programa inicial para ejecutarse: *bootstrap*.
- Por lo general, se almacena en hardware (en el firmware). Inicializa todos los aspectos del sistema, desde los registros de la *CPU* hasta los controladores de dispositivos y los contenidos de la memoria.
- El *bootstrap* debe ubicar el *kernel* del sistema operativo, cargarlo en la memoria y ejecutarlo.
- Una vez que el *kernel* se carga y se ejecuta, puede comenzar a proporcionar servicios al sistema y a sus usuarios.
- Algunos servicios se proporcionan fuera del *kernel,* por medio de programas del sistema que se cargan en la memoria en el momento del arranque; éstos se convierten en *daemons* del sistema, que se ejecutan todo el tiempo que el kernel se está ejecutando.

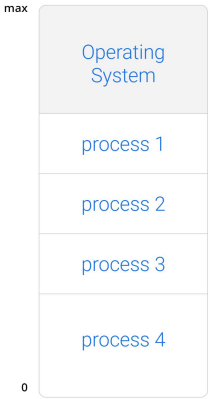
Operaciones del sistema operativo

- En Linux, el primer programa del sistema es "*systemd*", que inicia muchos otros *daemons*.
- Una vez que se completa esta fase, el sistema se inicia por completo y espera que ocurran eventos.
- Los eventos casi siempre son la ocurrencia de una interrupción.
- Otra forma de interrupción es un *trap* (o una excepción), que es una interrupción generada por *software* causada por un error (por ejemplo, división por cero o acceso de memoria no válido) o por una solicitud específica de un programa de usuario de un servicio del sistema.
- Esta solicitud se realiza ejecutando una operación especial denominada llamada del sistema (*system call*).

Operaciones del sistema operativo

Multiprogramación y multitarea

- La multiprogramación aumenta la utilización de la *CPU* organizando los programas para que la *CPU* siempre tenga uno para ejecutar. En un sistema multiprogramado, un programa en ejecución se denomina proceso.
- El SO mantiene varios procesos en la memoria, selecciona y comienza a ejecutar a uno.
- Es posible que el proceso deba esperar a que se complete alguna tarea.
- En un sistema no multiprogramado la CPU quedaría inactiva, mientras que en uno multiprogramado el SO simplemente cambia a otro proceso 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 haya un proceso por ejecutarse, la CPU nunca estará inactiva.



Operaciones del sistema operativo

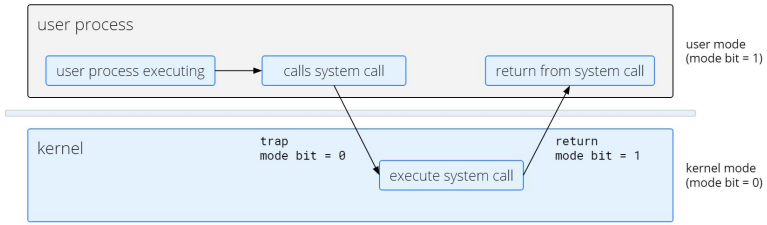
Multiprogramación y multitarea

- La multitarea es una extensión lógica de la multiprogramación. En los sistemas multitarea, la *CPU* ejecuta varios procesos alternando entre ellos, pero al hacerlo con mucha frecuencia, proporciona al usuario un tiempo de respuesta rápido.
- Cuando se ejecuta un proceso, normalmente es por un tiempo corto antes de que finalice o realice una E/S.
- La E/S puede ser interactiva, es decir, la salida va a una pantalla y la entrada proviene de un teclado, *mouse* o pantalla táctil.
- Dado que la E/S interactiva normalmente se ejecuta a "velocidad de personas", puede tardar mucho tiempo en completarse.
- En lugar de dejar que la *CPU* permanezca inactiva mientras se lleva a cabo esta entrada interactiva, el sistema operativo cambiará rápidamente la *CPU* a otro proceso.

Operaciones del sistema operativo

Operación de modo dual y multimodo

- Para garantizar la correcta ejecución del sistema, debe ser posible distinguir entre la ejecución del código del SO y el código definido por el usuario.
- Se necesitan como mínimo dos modos de operación separados: modo de usuario y modo *kernel*. Se agrega un *bit*, llamado *bit* de modo, al *hardware* de la computadora para indicar el modo actual: *kernel* (0) o usuario (1).
- El *bit* de modo permite distinguir entre una tarea del SO y una del usuario.
- Cuando una aplicación de usuario solicita un servicio del sistema operativo a través de una llamada del sistema, este debe pasar del modo usuario al modo *kernel*.



Operaciones del sistema operativo

Modo dual y multimodo

- El *hardware* permite que se ejecuten instrucciones privilegiadas sólo en modo *kernel*. Si se intenta ejecutar una instrucción privilegiada en modo usuario, el *hardware* no la ejecuta sino que la considera ilegal y se produce un *trap* al SO.
- Ejemplos: cambiar al modo *kernel,* el control de E/S, de temporizador y de interrupciones.
- El control inicial reside en el SO, donde las instrucciones se ejecutan en modo *kernel*. Cuando se otorga el control a una aplicación de usuario, se establece el modo usuario. Finalmente, el control vuelve al SO a través de una interrupción, un *trap* o una llamada de sistema.
- Las llamadas de sistema proporcionan los medios para que un programa de usuario le solicite al SO que realice tareas reservadas al SO.
- Cuando se ejecuta una llamada de sistema, el *hardware* generalmente lo trata como una interrupción de *software*.
    - El control pasa a través del vector de interrupción a una rutina de servicio en el SO, y el *bit* de modo se establece en modo *kernel*.
    - La rutina del servicio de llamada de sistema es una parte del SO.

Operaciones del sistema operativo

Timer

- Es necesario asegurarse de que el SO mantenga el control sobre la *CPU*. No se puede permitir que un programa de usuario se atasque en un bucle infinito o que no pueda llamar a los servicios del sistema y nunca devolver el control al SO.
- Para lograr este objetivo puede usarse un temporizador, el cual se configura para que interrumpa a la computadora después de un período determinado.
- El SO establece el contador y cada vez que el reloj hace tic el contador disminuye; cuando llega a 0 se produce una interrupción.
- Antes de entregar el control al usuario, el SO asegura que el temporizador esté configurado para producir interrupciones.
- Si el temporizador interrumpe, el control se transfiere automáticamente al SO, que puede considerar a la interrupción como un error fatal o puede dar al programa más tiempo.
- Las instrucciones que modifican el contenido del temporizador son privilegiadas.

Administración de recursos

Gestión de procesos

- Hasta el momento consideramos a un proceso como una instancia de un programa en ejecución, pero veremos que el concepto es más general.
- Un proceso necesita ciertos recursos para realizar su tarea (tiempo de *CPU*, memoria, archivos y dispositivos de E/S). Normalmente, estos recursos se asignan mientras el proceso se está ejecutando. Cuando el proceso finaliza, el SO reclama los recursos reutilizables.
- Un programa en sí mismo no es un proceso, sino que es una entidad pasiva (como el contenido de un archivo almacenado en el disco), mientras que un proceso es una entidad activa.
- Aunque dos procesos pueden asociarse al mismo programa, se consideran dos secuencias de ejecución separadas.
- Un proceso es la unidad de trabajo en un sistema, por lo tanto un sistema consiste en un conjunto de procesos, algunos de los cuales son procesos del SO y el resto son procesos del usuario.

Administración de recursos

Gestión de procesos

- El SO es responsable de las siguientes actividades relacionadas con la gestión de procesos:
  - Crear y eliminar procesos de usuario y de sistema.
  - Planificar procesos e hilos en las *CPUs*.
  - Suspender y reanudar procesos.
  - Proporcionar mecanismos para la sincronización de procesos.
  - Proporcionar mecanismos para la comunicación de procesos.

Administración de recursos

Gestión de memoria

- La memoria principal es generalmente el único gran almacenamiento al que la *CPU* puede acceder directamente. Las instrucciones deben estar en memoria para que la *CPU* las ejecute.
- Para mejorar tanto la utilización de la CPU como la velocidad de respuesta de la computadora a sus usuarios, las computadoras de propósito general deben mantener varios programas en la memoria, por lo que necesita administrar la memoria.
- El SO es responsable de las siguientes actividades relacionadas con la administración de memoria:
  - Realizar un seguimiento de qué partes de la memoria se están utilizando actualmente y qué proceso está utilizando cada una.
  - Asignar y desasignar espacio de memoria según sea necesario.
  - Decidir qué procesos (o partes de procesos) y datos se moverán dentro y fuera de la memoria.

Administración de recursos

Gestión del sistema de archivos

- El SO proporciona una vista lógica y uniforme del almacenamiento de información. Se abstrae de 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 estos archivos a través de los dispositivos de almacenamiento.
- Un archivo es una colección de información definida por su creador. Comúnmente, los archivos representan datos y programas (tanto en forma de fuente como de objeto).
- Cuando varios usuarios tienen acceso a los archivos, se debe controlar qué usuario puede acceder a un archivo y cómo (por ejemplo, leer, escribir, agregar).

Administración de recursos

Gestión del sistema de archivos

- El SO es responsable de las siguientes actividades relacionadas con la administración de archivos:
  - Crear y borrar archivos.
  - Crear y eliminar directorios para organizar archivos.
  - Proveer primitivas para la manipulación de archivos y directorios.
  - Mapear archivos en almacenamiento masivo.
  - Realizar copias de seguridad de archivos en medios de almacenamiento estables (no volátiles).

Administración de recursos

Gestión de almacenamiento masivo

- El sistema informático debe proporcionar almacenamiento secundario para poder realizar una copia de la memoria principal.
- Los principales medios de almacenamiento son dispositivos *HDD* y *NVM* para programas y datos.
- La mayoría de los programas, incluidos compiladores, navegadores web, procesadores de texto y juegos, se almacenan en estos dispositivos hasta que se cargan en la memoria.
- El almacenamiento secundario se utiliza con frecuencia, por lo cual debe ser usado eficientemente. Igualmente, hay almacenamientos que son más lentos y de menor costo (y a veces de mayor capacidad) como las unidades de cinta.

Administración de recursos

Gestión de almacenamiento masivo

- El SO es responsable de las siguientes actividades relacionadas con la administración del almacenamiento secundario:
  - Montaje y desmontaje.
  - Gestión del espacio libre.
  - Asignación de almacenamiento.
  - Planificación de discos.
  - Particionamiento.
  - Protección.

Administración de recursos

Gestión de *caché*

- A medida que es utilizada la información de la memoria ppal., se va copiando de forma temporal en un sistema de almacenamiento más rápido, el *caché*.
  - Cuando se necesita la información, primero se verifica si está en el *caché*, si está se utiliza directamente desde el *caché*.
  - Si no está en el *caché*, se busca en la fuente y se coloca una copia en el *caché* bajo el supuesto de que se necesitará pronto.
- El movimiento de información entre niveles de una jerarquía de almacenamiento puede ser explícito o implícito, dependiendo del diseño del *hardware* y del *software* del sistema operativo de control. Por ejemplo:
  - entre *caché* y registros → *hardware*.
  - entre disco y memoria → SO.
- En un entorno distribuido la situación se vuelve aún más compleja. En este entorno, varias copias (o réplicas) del mismo archivo pueden guardarse en diferentes computadoras.

Administración de recursos

Gestión de sistema de E/S

- Uno de los propósitos de un SO es ocultar las peculiaridades de los dispositivos de *hardware* específicos del usuario.
- Por ejemplo, en UNIX, las peculiaridades de los dispositivos de E/S están ocultas de la mayor parte del SO por el subsistema de E/S, que consta de varios componentes:
  - Componente de gestión de memoria, que incluye almacenamiento en *buffer*, almacenamiento en *caché* y cola de impresión.
  - Interfaz general *device-driver*.
  - *Drivers* para dispositivos de *hardware* específicos.
- Solo el *device-driver* conoce las peculiaridades del dispositivo específico al que está asignado.

Protección y seguridad

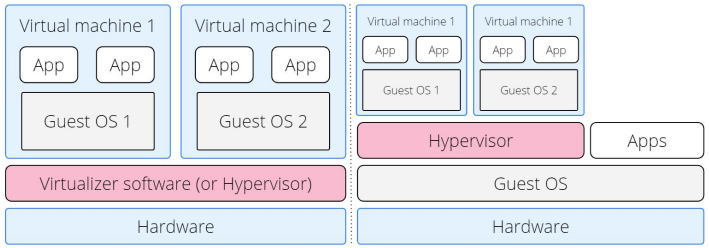
- Un sistema con múltiples usuarios y procesos debe regular el acceso a los datos. Es decir, los recursos sólo pueden ser operados por aquellos procesos que hayan obtenido la autorización del SO.
- Por ejemplo, el *hardware* de memoria garantiza que un proceso sólo pueda ejecutarse dentro de su propio espacio de direcciones. El *timer* garantiza que ningún proceso pueda obtener el control de la *CPU* sin renunciar al control al finalizar.
- La protección es cualquier mecanismo para controlar el acceso de los procesos o usuarios a los recursos definidos por un sistema informático.
- La protección puede mejorar la confiabilidad al detectar errores latentes en las interfaces entre los subsistemas de componentes.
- La detección temprana de errores de interfaz a menudo puede prevenir la contaminación de un subsistema sano por otro subsistema con mal funcionamiento. Además, un recurso no protegido no puede defenderse contra el uso (o mal uso) por parte de un usuario no autorizado o incompetente.

Protección y seguridad

- Un sistema puede tener una protección adecuada pero ser propenso a fallas y permitir un acceso inapropiado. Por ejemplo, si se roba la información de autenticación a un usuario, sus datos pueden ser copiados o eliminados, aunque la protección de archivos y memoria estén funcionando.
- Es un trabajo de seguridad defender a un sistema de ataques externos e internos.
- Dichos ataques abarcan una amplia gama que incluye: virus y gusanos, ataques de denegación de servicio (que usan todos los recursos del sistema y mantienen a los usuarios legítimos fuera del sistema), robo de identidad y robo del servicio (uso no autorizado de un sistema).
- En algunos sistemas, la prevención de algunos de estos ataques es una función del SO, mientras que en otros es una política o depende de *software* adicional.
- Debido al aumento alarmante en los incidentes de seguridad, las características de seguridad del SO son un área de investigación e implementación en rápido crecimiento.

Virtualización

- La virtualización permite abstraer el *hardware* de una computadora (*CPU*, memoria, disco, interfaz de red, etc.) en varios entornos de ejecución diferentes, creando así la ilusión de que cada entorno separado se está ejecutando en su propia computadora.
- Estos entornos pueden verse como diferentes SO individuales (por ej.: Windows, LinuX, macOS) que pueden ejecutarse al mismo tiempo y pueden interactuar entre sí.
- Permite que SOs se ejecuten como aplicaciones dentro de otros SOs.



Type 1 Bare metal

Type 2 Hosted

Virtualización

- **Emulación:** simular hardware de computadora en software, generalmente utilizada cuando el tipo de *CPU* de origen es diferente al tipo de *CPU* de destino.
  - Cada instrucción de máquina que se ejecuta de forma nativa en el sistema de origen debe traducirse a la función equivalente en el sistema de destino, lo que con frecuencia resulta en varias instrucciones de destino (costo en *performance*).
  - Algunas ventajas:
    - Posibilidad de usar otros sistemas.
    - Entornos de prueba.
- **Virtualización:** el SO es compilado de forma nativa para la *CPU*, ejecutando SOs invitados que también son compilados de forma nativa.
  - Algunas Ventajas:
    - Reducción de costos.
    - Seguridad.
    - Administración eficiente de la información.

Estructuras de datos del *kernel*

- Entre las estructuras de datos fundamentales, utilizadas extensivamente en SO, se encuentran:
  - Listas
    - Listas simplemente enlazadas.
    - Listas doblemente enlazadas.
    - Listas enlazadas circulares.
  - Pilas
  - Colas
  - Árboles
    - Árboles binarios.
    - Árboles rojo-negro.
  - *Hashs*
  - *Bitmaps*

Entornos informáticos

Tradicional

- *PCs* de propósito general conectadas a una red.
- Las tecnologías *web* y el aumento del ancho de banda de la *WAN* están ampliando los límites de estos entornos, otorgando a los usuarios domésticos y a las empresas mayor acceso a más datos.
- Las empresas crean sitios *web*, los cuales proporcionan acceso a sus servidores internos.
- Las computadoras de red (*thin clients*), que son esencialmente terminales que entienden la computación basada en la *web*, se utilizan en lugar de las estaciones de trabajo tradicionales, donde se desea mayor seguridad o un mantenimiento más sencillo.
- Las *notebooks* pueden sincronizarse con las *PCs* para permitir un uso muy portátil de la información de la empresa.
- Los dispositivos móviles también pueden conectarse a redes inalámbricas y a redes de datos de celulares para usar el sitio *web* de la empresa, así como otros recursos *web*.
- Evolución del escritorio

Entornos informáticos

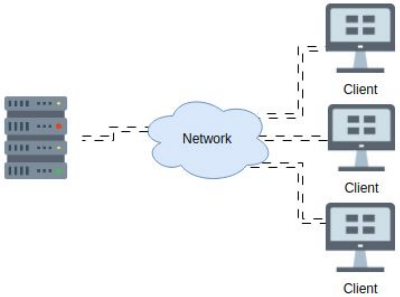
Móvil

- Computación en *smartphones* y *tablets* → características físicas distintivas de ser portátiles y ligeros.
- Puede resultar difícil realizar una distinción de funcionalidad entre una notebook y un *smartphone*.
- Incluso tienen funcionalidades que no está disponible o que no son prácticas en una computadora de escritorio o portátil como GPS, acelerómetros y giroscopios.
- Los dispositivos móviles suelen utilizar redes *wireless* o redes de datos de celulares.
- Limitadas capacidades de memoria y la velocidad de procesamiento de los dispositivos móviles en relación a una *PC*.
- El consumo de energía es un factor a considerar, los dispositivos móviles usan procesadores más pequeños, más lentos y con menos núcleos de procesamiento que los procesadores de las computadoras de escritorio y portátiles.
- Actualmente, dos SO dominan la computación móvil: *Apple iOS* y *Google Android*.

Entornos informáticos

cliente-servidor

- Los servidores satisfacen las solicitudes generadas por los clientes.
- Es un sistema distribuido especializado:
- Servidor de cálculo: proporciona una interfaz a la cual un cliente puede enviar una solicitud para realizar una acción. En respuesta, el servidor ejecuta la acción y envía los resultados al cliente.
  - Un ejemplo de este sistema es un servidor que ejecuta una base de datos en respuesta a la solicitud de datos de los clientes.

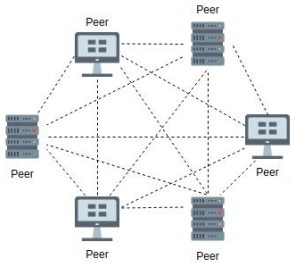


- Servidor de archivos: proporciona una interfaz del sistema de archivos, en el cual los clientes pueden crear, actualizar, leer y eliminar archivos.
  - Un ejemplo de tal sistema es un servidor *web* que entrega archivos a clientes en sus navegadores *web*.

Entornos informáticos

peer-to-peer

- Los nodos se consideran pares (*peer*), y cada uno puede actuar como cliente o servidor, dependiendo de si está solicitando o proporcionando un servicio.
- Los servicios se proporcionan por varios nodos distribuidos, lo cual representa una ventaja sobre los sistemas cliente-servidor tradicionales.
- Para ser parte de una red *peer-to-peer*, un nodo primero debe unirse a la red. Una vez unido, comienza a proporcionar servicios y también a solicitarlos.
- Dos maneras para determinar qué servicios están disponibles:
  - Cualquier nodo que desee un servicio específico primero realiza una búsqueda para determinar qué nodo proporciona dicho servicio.
  - No utilizar un servicio de búsqueda centralizado sino que, un *peer* actúa como cliente y debe descubrir qué nodo proporciona un servicio deseado transmitiendo una solicitud del servicio a todos los demás nodos de la red.



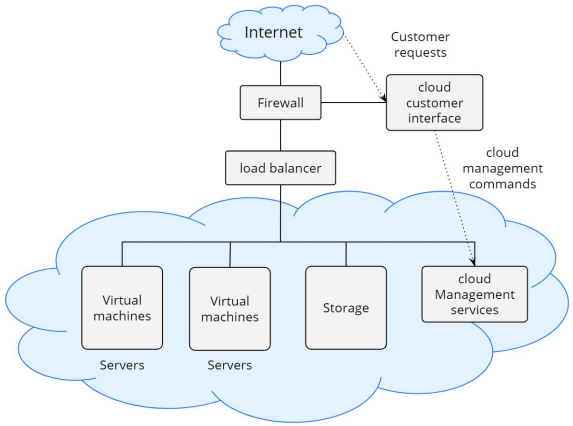
Entornos informáticos

cloud

- Ofrece cómputo, almacenamiento e incluso aplicaciones como un servicio a través de una red.
- Extensión lógica de la virtualización dado que usa la virtualización como base.
- Tipos de *cloud computing*:
  - Nube pública, privada o híbrida.
  - *Software* como servicio (*SaaS*): una o más aplicaciones (ej.: como procesadores de texto u hojas de cálculo) disponibles a través de Internet.
  - Plataforma como servicio (*PaaS*): una pila de *software* lista para el uso de la aplicación a través de Internet (ej.: un servidor de base de datos).
  - Infraestructura como servicio (*IaaS*): servidores o almacenamiento disponibles en Internet (ej.: almacenamiento disponible para hacer copias de respaldo de datos).

Entornos informáticos

cloud



Una nube pública que proporciona IaaS. Tenga en cuenta que tanto los servicios en la nube como la interfaz de usuario en la nube están protegidos por un firewall.

Entornos informáticos

Sistemas embebidos en tiempo real

- Quizás sea la forma más frecuente de computadoras que existen; desde motores y robots hasta hornos de microondas.
- Tareas específicas y sistemas primitivos con funciones limitadas.
- Por lo general, tienen poca o ninguna interfaz de usuario, dado que solo monitorean y administran dispositivos de *hardware*, como motores de automóviles y brazos robóticos.
- Pueden ser computadoras de propósito general que ejecutan un SO estándar con una aplicación dedicada y también pueden ser dispositivos de *hardware* con circuitos integrados específicos que realizan sus tareas sin SO.
- Sistemas operativos en tiempo real:
  - Se usa cuando se han establecido requisitos específicos de tiempo en la operación de un procesador o en el flujo de datos.
  - Ejemplos: Sistemas de experimentos científicos, sistemas de imágenes médicas, sistemas de control industrial y ciertos sistemas de visualización.

Sistemas operativos Free y Open-Source

- El estudio de los SO se ha simplificado gracias al *free software* y al *open-source*. Disponibles en formato de código fuente en lugar de como código binario compilado.
- El *free software* y el *open-source software* son dos ideas diferentes defendidas por distintos grupos de personas.

https://www.fsf.org/
https://www.gnu.org/philosophy/open-source-misses-the-point.es.html

https://www.fsf.org/
https://opensource.org/osd

- El *free software* no sólo hace que el código fuente esté disponible, sino que también tiene licencia para permitir su uso, redistribución y modificación sin costo. El *software open-source* no ofrece necesariamente tales licencias. Por lo tanto, aunque todo *free software* es *open source*, algunos *softwares open-source* no son "*free*".
- *GNU/Linux* es el sistema operativo open-source más famoso, con algunas distribuciones *free* y otras solamente *open-source*.
- *Microsoft Windows* es un ejemplo bien conocido del enfoque de código cerrado. *Windows* es un software propietario: *Microsoft* lo posee, restringe su uso y protege cuidadosamente su código fuente.