Sistemas Operativos I

Módulo I

GENERALIDADES DE SISTEMAS OPERATIVOS

1

Temas del Módulo I

- **Definición de Sistema Operativo.**
  - Modelo de un sistema de computación.
  - Definición de Sistema Operativo. Funciones de los sistemas operativos:
    - Como interfaz usuario/hardware (máquina extendida).
    - Como gestor de recursos.
- **Tipos de Sistemas Operativos.**
  - Clasificación general.
  - Mainframe, Servidor, Multiprocesador.
  - Computadora personal (PC), Computadoras de mano.
  - Embebidos.
  - Tiempo real.
- **Llamadas al sistema (system calls).**
  - Para administración de procesos.
  - Para administración de archivos.
  - Para administración de directorios.
  - Para usos misceláneos.
  - La API Win32.
- **Estructura del Sistema Operativo.**
  - Monolíticas.
  - Estructuradas.

2

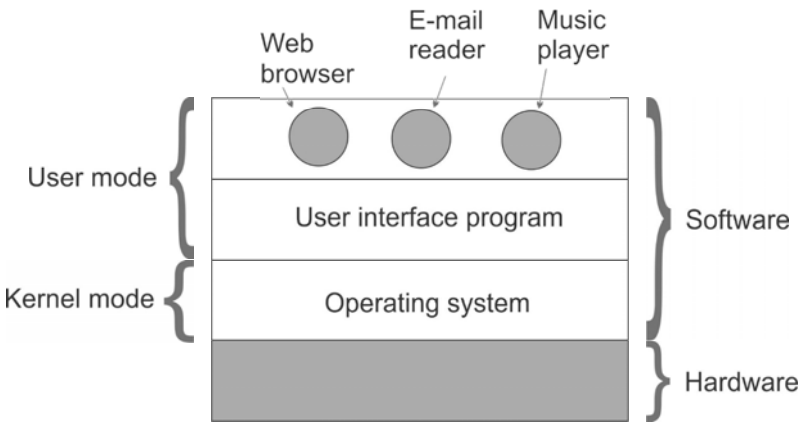
Definición de Sistema Operativo

- **Modelo de un sistema de computación.**
- Vimos ya los componentes principales de una computadora: CPU, memoria principal, subsistema de E/S y buses. Ahora tenemos que “hacer funcionar” ese hardware.
- Si cada programador tuviera que entender cómo funcionan, en detalle, cada uno de estos componentes, simplemente no se hubiera escrito nunca un programa de usuario.
- Más aún, administrar estos componentes y lograr que sean utilizados de manera óptima, es un desafío todavía mayor.
- Es por eso que las computadores se equipan con una capa de software llamada **sistema operativo**, cuya función es proveer a los programas de usuario un modelo simple y transparente de la computadora, además de administrar los componentes mencionados.

3

Definición de Sistema Operativo

- **Modelo de un sistema de computación.** (*cont.*)
- Se hace necesario, entonces, definir un nuevo modelo de computadora, el cual incluya el SO y los programas de usuario:



4

Definición de Sistema Operativo

- **Definición de Sistema Operativo.**
- ¿Software que ejecuta sobre el hardware en modo kernel?
- No es del todo cierto que ejecuta en modo kernel…
- Al menos no todo se ejecuta en modo kernel (*continuará…*).
- Para conocer con más exactitud qué es un SO conviene, más bien, estudiar sus funciones en una computadora, las cuales son básicamente dos funciones independientes entre sí:
- **Funciones del Sistema Operativo:**
  - Como **interfaz** usuario/hardware (máquina extendida).
  - Como **gestor** de recursos.

5

Definición de Sistema Operativo

- **Funciones del Sistema Operativo.** (*cont.*)
- Como **interfaz** usuario/hardware (máquina extendida).
- Imaginemos que tenemos que hacer funcionar un disco SATA. Existe un libro (Anderson, 2007) que describe la primera versión de la interfaz, con todo lo que un programador debe saber para hacer funcionar el disco, en unas 450 páginas.
- A menos que nuestro trabajo sea el hacer funcionar estos discos, ningún programador quiere tener que lidiar con éste nivel de programación. En lugar de esto, utilizaríamos un *driver* de disco, el cual provee la interfaz para poder leer y escribir bloques de datos en el disco. Un SO contiene muchos drivers para dispositivos de E/S.
- Aún así, éste nivel de programación sigue siendo “bajo”. Es por eso que los SO brindan otra capa de abstracción: el archivo. Todos los SO incorporan esta capa.

6

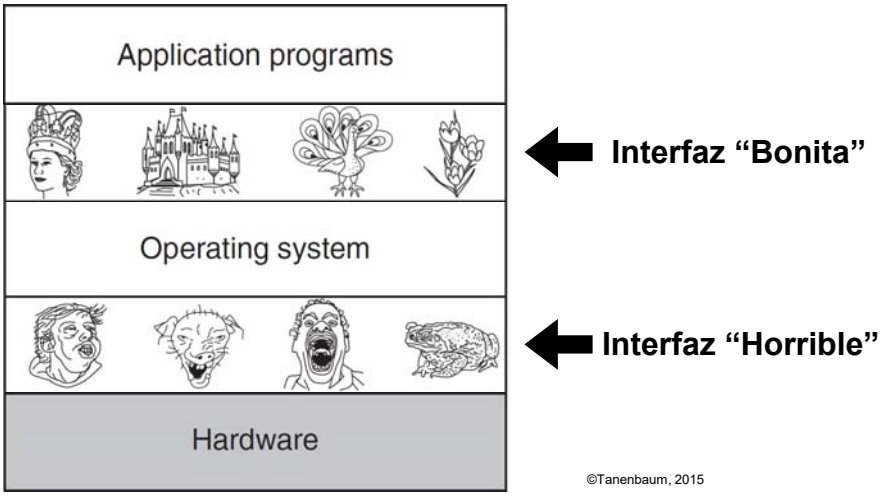
Definición de Sistema Operativo

- **Funciones del Sistema Operativo.** (*cont.*)
- Como **interfaz** usuario/hardware (máquina extendida). (*cont.*)
- La clave, entonces, son las abstracciones. La tarea del SO es crear buenas abstracciones a fin de brindar al usuario una interfaz “amigable” y “bonita” para poder manejar el hardware.
- Volviendo al ejemplo del archivo, ésta capa de abstracción permite crear, leer y escribir archivos en un disco, sin tener que conocer los detalles de funcionamiento de éste último.
- Cabe destacar que incluso de un mismo tipo de dispositivo de E/S existen varios fabricantes, cada uno con su forma de construirlos. El SO también tiene que “ocultar” este detalle al usuario.
- De este modo, el SO convierte, a través de las sucesivas capas de abstracción, lo “feo” del hardware en algo “lindo” de programar y manejar.

7

Definición de Sistema Operativo

- **Funciones del Sistema Operativo.** (*cont.*)
- Como **interfaz** usuario/hardware (máquina extendida). (*cont.*)



8

Definición de Sistema Operativo

- **Funciones del Sistema Operativo.** (*cont.*)
- Como **gestor** de recursos.
- Las capas de abstracciones creadas por el SO representan una "vista desde arriba" del nuevo modelo de computadora.
- Si hacemos una "vista desde abajo", tenemos hardware que quiere ser utilizado por cada programa y por cada usuario de una computadora.
- En este sentido, el SO debe proveer una forma ordenada y controlada de asignar los recursos de la computadora. Estos recursos son: el CPU, memoria principal, dispositivos de E/S.
- Esto significa que cada usuario/programa recibirá por parte del SO el uso de los recursos para su funcionamiento.
- Para ello, el SO utiliza una técnica llamada **multiplexación** (compartir), la cual puede ser en **tiempo** y **espacio**.

9

Definición de Sistema Operativo

- **Funciones del Sistema Operativo.** (*cont.*)
- Como **gestor** de recursos. (*cont.*)
- Cuando un recurso es multiplexado en tiempo, los programas y usuarios toman "**turnos**" para utilizarlo (Módulo IV), esto es, primero lo utiliza un programa y pasado un cierto tiempo, lo usa otro, y así sucesivamente.
- Por ejemplo, si sólo se dispone de un CPU y se tienen varios programas para ejecutar, el SO primero asigna el CPU a un programa; después de haber ejecutado "lo suficiente", otro programa hace uso del CPU, y luego otro, hasta que eventualmente se retorna al primero.
- Otro ejemplo de multiplexación en tiempo es el compartir una impresora. Cuando se tiene una única impresora y varios programas para imprimir, el SO decide quién la utiliza primero y quién será el siguiente.

10

Definición de Sistema Operativo

- **Funciones del Sistema Operativo.** (*cont.*)
- Como **gestor** de recursos. (*cont.*)
- La otra forma de compartir los recursos es el multiplexado en espacio. En lugar de tomar "turnos", los programas y usuarios toman **partes** del recurso.
- Por ejemplo, la memoria principal se puede dividir entre varios programas en ejecución, de modo que cada uno puede estar residente al mismo tiempo (más detalles en el Módulo V).
- Otro recurso que se multiplexa en espacio es el disco. Este puede almacenar los archivos de múltiples usuarios a la vez. De modo que, la asignación de espacio en ese disco y llevar la cuenta de quién lo utiliza, es tarea del SO.

11

Tipos de Sistemas Operativos

- **Clasificación general.**
- Dijimos que los SO se definen de manera más adecuada según sus funciones. También podemos clasificarlos según sus funciones, pero también es muy importante destacar aspectos como su interacción con el usuario.
- Podemos clasificarlos, entonces, en dos grandes categorías:
  - **Según su interacción con el usuario:** esto es, si el SO permite o no que el usuario intervenga activamente en el envío de comandos o la realización de tareas.
  - **Según el propósito de su uso:** es decir, si están pensados para realizar cualquier tipo de tarea, o bien un único conjunto acotado de tareas.

12

Tipos de Sistemas Operativos

- **Clasificación general.** (*cont.*)
- Respecto de la interacción con el usuario, un SO será:
  - **Interactivo:** cuando el usuario interviene activamente.
  - **De procesamiento por lotes (batch):** cuando el usuario sólo interactúa para iniciar una tarea y no participa del procesamiento, sino que al final recibe los resultados.
- Respecto del propósito de su uso, un SO será de:
  - **Propósitos generales:** si permiten realizar cualquier tipo de tarea, por ejemplo, los SO de escritorio, donde se pueden utilizar editores de texto, planillas de cálculo, navegadores de Internet, juegos, etc..
  - **Propósitos específicos:** se diseñan para ejecutar un conjunto pequeño de tareas, como por ejemplo, una estación meteorológica, que sólo mide variables como temperatura, presión atmosférica, etc..
- A continuación, estudiaremos algunos tipos de sistemas operativos, teniendo en cuenta todas estas clasificaciones, destacando sus funciones principales.

13

Tipos de Sistemas Operativos

- **Mainframe.**
- Son los de más alto nivel. Están orientados a procesar muchos trabajos a la vez, en donde la mayoría requiere **grandes cantidades de operaciones de E/S**.
- Típicamente ofrecen tres tipos de servicios:
  - **Procesos batch (lotes):** son trabajos rutinarios sin un usuario interactivo presente. Por ejemplo, la generación de reportes de ventas de una cadena de tiendas.
  - **Procesamiento de transacciones:** manejan un gran número de pequeñas solicitudes (cientos o miles por segundo). Por ejemplo, el sistema de reserva de pasajes de una aerolínea.
  - **Tiempo compartido:** permite a múltiples usuarios remotos ejecutar tareas simultáneas, como por ejemplo, consultas en una gran base de datos.
- Estas funciones están estrechamente relacionadas; los SO de mainframe usualmente realizan todas ellas.

14

Tipos de Sistemas Operativos

- **Servidor.**
- Un nivel más abajo están estos SO; éstos ejecutan sobre servidores, cuyo hardware **equivale a una PC "grande"** (procesador más veloz, más memoria principal y varios discos para almacenamiento secundario con algún método de redundancia), o incluso en mainframes.
- Ofrecen servicios a múltiples usuarios de una red, y permiten compartir recursos de hardware y software, tales como:
  - **Impresoras:** cada usuario accede a un impresora conectada a la red.
  - **Archivos:** ya sea mediante carpetas compartidas o SGBD.
  - **Servicios web:** tanto aplicaciones internas como expuestas en Internet.
- Un uso típico de estos SO es en los ISP (Internet Service Provider), en los que se almacenan las páginas web y se manejan las solicitudes de los clientes que se conectan a ellas.

15

Tipos de Sistemas Operativos

- **Multiprocesador.**
- Una manera de aumentar el poder de cómputo es aumentar la cantidad de procesadores de una computadora. Según cómo se conectan y cómo se comparten los procesadores, éstos pueden llamarse (más detalles en SO II):
  - **Computadoras paralelas.**
  - **Multicomputadores.**
  - **Multiprocesadores.**
- Actualmente, se disponen de procesadores multinúcleo (multicore) para PC, lo que permite a los SO para computadoras de escritorio y notebooks trabajar al menos con versiones de menor escala de multiprocesamiento.
- Los SO multiprocesador existen y se desarrollan desde hace tiempo (30 años o más); sin embargo, la limitación actual es el desarrollo de aplicaciones que utilicen al máximo sus ventajas.

16

Tipos de Sistemas Operativos

- **Computadoras personales (PC).**
- Un escalón más abajo están los SO para PC. Actualmente dan soporte de multiprogramación, a menudo con docenas de programas que se cargan desde el “booteo”.
- Su función es proveer un buen apoyo a las tareas de un único usuario (no confundir con el soporte multiusuario de los SO vistos anteriormente).
- Se utilizan para tareas variadas:
  - Procesamiento de texto, planillas de cálculos, etc. (ofimática).
  - Juegos.
  - Acceso a Internet.
- Son los de mayor difusión y los más utilizados.

17

Tipos de Sistemas Operativos

- **Computadoras de mano.**
- Continuando “hacia abajo” en la escala de los SO, tenemos aquellos para computadoras de mano. Al principio, estas computadoras se conocían como **PDA** (Personal Digital Assistant), y estaban diseñadas para caber en una mano.
- Los ejemplos más modernos y mejor conocidos son los SO para teléfonos inteligentes (smartphones) y tabletas (tablets).
- La mayoría de ellos ya tiene soporte para multinúcleos y manejan aplicaciones y dispositivos complejos tales como:
  - GPS.
  - Cámaras HD y 4K.
  - Sensores de proximidad y aceleración.
  - Juegos con alta calidad gráfica.
- El mercado está prácticamente dominado por Android de Google y iOS de Apple, aunque hay algunos competidores.

18

Tipos de Sistemas Operativos

- **Sistemas operativos embebidos.**
- Los sistemas embebidos con computadoras que controlan dispositivos, y no son pensados como computadoras en sí, sino que normalmente no aceptan software instalado por usuarios.
- Ejemplos típicos de estos son lo que encontramos en dispositivos tales como:
  - Microondas.
  - Televisores (no necesariamente "Smart").
  - Reproductores de DVD y Blu-ray.
  - Reproductores de MP3.
- La diferencia fundamental entre éstos y los SO de mano es que nunca ejecutarán software no confiable, por lo que no necesitan protección entre aplicaciones, lo que lleva a una simplificación en su diseño.

19

Tipos de Sistemas Operativos

- **Sistemas operativos de Tiempo Real.**
- Estos SO se caracterizan por tener el tiempo como parámetro clave. Por ejemplo, en una fábrica de automóviles, si una pieza tiene que pasar por un robot de soldadura, y pasa más tarde de lo que debe, la pieza puede quedar arruinada.
- Si la acción **debe** ocurrir en un momento preciso (en un intervalo de tiempo), se tiene un sistema de tiempo real duro. Aplicaciones típicas se dan en procesos industriales, control de navegación, balística, etc.
- Por otro lado, si **se pierden acciones** ocasionalmente, **aunque sea indeseable**, y **no se produce** ningún **daño** grave ni permanente, se tiene un sistema de tiempo real suave. Ejemplos de éstos son los sistemas digitales de audio o multimedia en general. Los smartphones también son sistemas de tiempo real suave. (más detalles en SO II)

20

Tipos de Sistemas Operativos

- **Conclusiones.**
- La mayoría de los SO son interactivos. Aún los embebidos, aunque no se pueda instalar aplicaciones, tienen participación del usuario en la mayoría de los casos.
- La mayoría de los SO actuales manejan varios procesadores. Desde los servidores hasta los Smartphone y Smart TV, tienen procesadores de hasta ocho núcleos.
- Los SO de tiempo real son, en general, de uso específico, típicamente en ambientes industriales. (Se verán en SO II).
- Los Smartphone son sistemas de tiempo real, del tipo suave.
- Todos los SO ejecutan sobre una arquitectura de hardware que cuenta mínimamente con un CPU, memoria principal, dispositivos de E/S y buses.
- Todos los tipos de SO cumplen las funciones vistas: interfaz hardware/usuario, y gestión de recursos.

21

Llamadas al sistema (system calls)

- Las llamadas al sistema son **rutinas del SO**, las cuales pueden acceder al hardware. Son invocadas cada vez que un programa de usuario requiere de éste acceso, por lo que estas constituyen una interfaz programa/SO.
- Esto significa que las llamadas al sistema forman una **capa de abstracción** utilizada por los programas de usuario. Esta capa se denomina **API (Application Programming Interface)**.
- Esta interfaz varía de un SO a otro, aunque los conceptos subyacentes tienden a ser similares.
- Si bien la mecánica de una llamada al sistema es altamente dependiente del hardware, esto es, son programadas en lenguaje ensamblador (*assembler*, o lenguaje de máquina), normalmente se cuenta con librerías de procedimientos para poder hacer llamadas al sistema desde programas escritos en lenguaje C.

22

Llamadas al sistema (system calls)

- Vamos a suponer una computadora con un único procesador. En ésta se puede ejecutar sólo un programa a la vez.
- Si un programa de usuario se encuentra ejecutando, en modo usuario, y necesita un servicio del SO, por ej., leer los datos de un archivo, entonces tendrá que ejecutar una instrucción que genera una interrupción de programa, llamada **trap**.
- De esta manera se transfiere el control del CPU al SO. Éste inspecciona la solicitud de la llamada al sistema analizando los parámetros.
- Luego ejecutará la llamada al sistema y al finalizar, devuelve el control del CPU al programa que la invocó desde la instrucción siguiente a la llamada.
- Desde cierto punto de vista, una llamada al sistema equivale a una llamada a un procedimiento, con la diferencia que éstas ejecutan en **modo kernel**.

23

Llamadas al sistema (system calls)

- En gran medida, los servicios ofrecidos por las llamadas al sistema determinan la mayoría de lo que el SO tiene hacer, ya que la administración de recursos es mínima, al menos en PC comparada con máquinas con múltiples usuarios.
- Los servicios incluyen tareas como la creación y terminación de procesos (Módulo II), creación, eliminación, lectura y escritura de archivos, administración de directorios (carpetas), y realizar operaciones de E/S.
- Veremos a continuación algunas llamadas al sistema en POSIX:
  - Para administración de procesos.
  - Para administración de archivos.
  - Para administración de directorios.
  - Para usos misceláneos.

24

Llamadas al sistema (system calls)

- **System calls para administración de procesos.**
- La principal llamada al sistema para administración de procesos **fork**.
- Con ésta llamada es la única manera de crear un nuevo proceso. Crea una duplicado exacto del proceso que invoca la system call, incluyendo sus descriptores de archivo (Módulo V), registros, etc.
- Luego de la creación, el proceso original (llamado padre) y el nuevo (llamdo hijo) toman "caminos separados". Esto es, aunque tienen las mismas variables, los cambios en uno no se reflejan en el otro.
- La llamada a fork retorna un valor entero, el cual es cero en el proceso hijo (en su entorno de variables) y es igual al PID (Process IDentifier) en el proceso padre. (más detalles en Módulo II).

25

Llamadas al sistema (system calls)

- **System calls para administración de procesos.** (*cont.*)
- Otras llamadas al sistema para administración de procesos son:
- **waitpid(pid, &statloc, options):** espera la terminación de un proceso hijo. Lo habitual es que cuando se crea un proceso, éste ejecuta un código diferente del proceso padre, por lo que esta system call permite al proceso padre esperar a que termine la ejecución, ya sea de un proceso hijo en particular o de cualquier proceso hijo que esté en ejecución.
- **execve(name, argv, environp):** reemplaza la imagen central de un proceso. Esta llamada es la que permite cambiar el código que debe ejecutar un proceso creado por *fork*, el cual es especificado por el parámetro *name*. Los argumentos *argv* y *environp* corresponden a los parámetros del código a ejecutar y el entorno de ejecución, respectivamente.

26

Llamadas al sistema (system calls)

- **System calls para administración de procesos.** (*cont.*)
- **exit(status):** termina la ejecución de un proceso y retorna el valor *status*. Este valor corresponde a un identificador de tipo numérico que indica la condición con la que se termina la ejecución. Puede tomar valores entre 0 y 255. Todos los procesos deberían invocarla al finalizar su ejecución.

27

Llamadas al sistema (system calls)

- **System calls para administración de archivos.**
- Si bien hay muchas llamadas al sistema para administración de archivos, veremos las más básicas y fundamentales. Éstas se utilizan para manejar archivos individuales.
- **open(file, how, …):** abre un archivo para lectura, escritura, o ambas. El parámetro *file* especifica el archivo que se quiere abrir, ya sea a través de un camino absoluto o bien, relativo al directorio de trabajo actual. El parámetro *how* indica el modo de apertura, el cual puede ser:
  - Sólo lectura.
  - Sólo escritura.
  - Lectura y escritura.
  - Creación, si el archivo no existe.

28

Llamadas al sistema (system calls)

- **System calls para administración de archivos.** (*cont.*)
- **close(fd):** cierra un archivo abierto. Tan simple como eso, el parámetro es similar al de la system call open que especifica el nombre del archivo.
- **read(fd, buffer, nbytes):** lee datos de un archivo y los almacena en un buffer. El parámetro *fd* nuevamente indica el archivo, este caso, a leer; *buffer* indica el comienzo del sector de memoria donde se almacenará los datos leídos, y *nbytes*, la cantidad de Bytes a ser leídos del archivo.

  El uso típico de esta system call es para leer un archivo en forma secuencial, esto es, desde el comienzo del archivo, y Byte por Byte. Sin embargo es posible acceder al archivo en forma aleatoria utilizando otra system call complementaria, que veremos más adelante.

29

Llamadas al sistema (system calls)

- **System calls para administración de archivos.** (*cont.*)
- **write(fd, buffer, nbytes):** escribe datos de un buffer en un archivo. Al igual que *read*, *fd* indica el archivo, este caso, a ser escrito; *buffer* indica el comienzo del sector de memoria donde se toman los datos a escribir, y *nbytes*, la cantidad de Bytes a ser escritos en el archivo.
  También es usada para escribir en un archivo en forma secuencial, pero al igual que *read*, también es posible hacer escrituras en lugares aleatorios del archivo.
- **lseek(fd, offset, whence):** para realizar la lectura o escritura, el SO mantiene un puntero hacia el archivo. Este puntero se desplaza Byte a Byte secuencialmente, pero también es posible moverlo a posiciones aleatorias con ésta system call. El parámetro *offset* indica la posición en el archivo al que se quiere mover el puntero, mientras que *whence* indica desde dónde se moverá: el comienzo, la posición actual o el final del archivo.

30

Llamadas al sistema (system calls)

- **System calls para administración de archivos.** (*cont.*)
- **stat(name, &buffer):** obtiene la información de estado de un archivo. Permite conocer información acerca del tamaño, fecha de última modificación, modo, y otros datos.

31

Llamadas al sistema (system calls)

- **System calls para administración de directorios.**
- Estas llamadas al sistema están más relacionadas con directorios y con el sistema de archivos como un todo, más que en archivos individuales. Destacamos las siguientes:
- **mkdir(name, mode):** crea un nuevo directorio. El parámetro *name* indica el nombre que tendrá el directorio, y el parámetro *mode*, el modo de creación. Éste último define quién es el propietario del directorio, permisos de lectura, escritura, etc.
- **rmdir(name):** elimina un directorio vacío. El parámetro *name* indica el nombre del directorio a eliminar.
- **link(name1, name2):** crea una nueva entrada llamada *name2*, la cual apunta a *name1*. El propósito de esta llamada es lograr que un mismo archivo aparezca con uno o más nombres, incluso en diferentes directorios. Esto permite que los cambios hechos en un directorio se reflejen instantáneamente en todos aquellos en los que se encuentra "linkeado".

32

Llamadas al sistema (system calls)

- **System calls para administración de directorios.** (*cont.*)
- **unlink(name):** elimina la entrada llamada *name*. Se utiliza para eliminar un link simbólico a un archivo. Si es el último link del archivo, éste último se elimina cuando deja de estar en uso por algún proceso.
- **mount(special, name, flag):** permite acoplar dos sistemas de archivos en uno sólo. El primer parámetro indica el sistema de archivos que se quiere acoplar; el segundo parámetro, el sistema de archivos que "recibe" al primero; el tercer parámetro indica si se acopla para lectura, escritura, o ambas.
- **umount(special):** desacopla un sistema de archivos. El parámetro indica el sistema de archivos a desacoplar.

33

Llamadas al sistema (system calls)

- **System calls para usos misceláneos.**
- Existen otras llamadas al sistema para usos variados, que amplían las tipos de tareas que realiza el SO. Veremos las siguientes:
- **chdir(dirname):** cambia el directorio de trabajo. El directorio de trabajo es aquel donde se referenciarán cada acción que se realiza, como por ejemplo, creación de archivos, directorios, etc. Esta llamada al sistema elimina la necesidad de escribir caminos absolutos muy largos todo el tiempo.
- **chmode(name, mode):** cambia los bits de protección de un archivo. El parámetro *name* indica el nombre del archivo al que se quiere cambiar los bits de protección; el parámetro *mode*, los nuevos valores que tomarán estos bits.
- **kill(pid, signal):** envía una señal a un proceso. El *pid* indica el identificador de proceso, es decir, el proceso receptor de la señal; el segundo parámetro indica qué señal recibirá el proceso.

34

Llamadas al sistema (system calls)

- **System calls para usos misceláneos.** (*cont.*)
- **time(&seconds):** retorna la cantidad de segundos transcurridos desde las 0hs del 1 de enero de 1970. En computadoras con tamaño de palabra de 32bits, el máximo valor retornado es 2^{32}-1, lo que permite calcular fechas de hasta 136 años desde la fecha de inicio.

35

Llamadas al sistema (system calls)

- **La API Win32.**
- Vimos que en POSIX hay casi una relación de uno a uno en las llamadas al sistema y las funciones de librería para invocarlas. La system call *read* y la función de librería *read* de C, es un claro ejemplo de esto.
- Con Windows la situación es bastante diferente. Para empezar, las llamadas al sistema y las librerías de funciones están muy desacopladas. Microsoft ha definido un conjunto de procedimientos llamado API Win32, que los programadores deben utilizar para obtener servicios del SO.
- Al desacoplar la interfaz de las llamadas al sistema, Microsoft tiene la capacidad de modificar las llamadas al sistema sin invalidar programas existentes.
- Sin embargo, el conjunto de llamadas al sistema varía de versión a versión, ampliándose con las más nuevas.

36

Llamadas al sistema (system calls)

- **La API Win32.** (*cont.*)
- La cantidad de llamadas a la API Win32 es muy grande (en el orden de miles). Y aunque muchas de ellas invocan a llamadas al sistema, un gran número ejecutan en modo usuario. Incluso lo que en una versión era una llamada al sistema, en otra versión es una función de librería en espacio de usuario.
- Veremos algunas de las llamadas a la API Win32, en especial las que se correspondan con la funcionalidad de POSIX:
- **CreateProcess:** realiza el trabajo combinado de *fork* y *execve* de POSIX. Tiene muchos parámetros para especificar las propiedades del proceso creado. Windows no tiene una jerarquía de procesos como POSIX, por lo que no existe el concepto de proceso padre y proceso hijo; una vez creado un proceso, el creador y el creado son iguales.

37

Llamadas al sistema (system calls)

- **La API Win32.** (*cont.*)
- **WaitForSingleObjet:** se utiliza para esperar un evento; se pueden esperar muchos eventos posibles. Si el parámetro indica un proceso, entonces el proceso llamador espera a que el proceso especificado termine.
- **ExitProcess:** se utiliza para terminar un proceso en ejecución.
- En las diapositivas siguientes veremos un resumen de las llamadas a la API Win32 con funcionalidad equivalente en POSIX.

38

Llamadas al sistema (system calls)

- **La API Win32.** (*cont.*)
- **Para manejo de procesos:**

POSIX	Win32	Descripción
fork	CreateProcess	Crea un nuevo proceso
waitpid	WaitForSingleObjet	Puede esperar a que un proceso termine
execve	(ninguno)	CreateProcess = fork + execve
exit	ExitProcess	Termina la ejecución

- **Para manejo de archivos:**

POSIX	Win32	Descripción
open	CreateFile	Crea un nuevo archivo o abre uno existente
close	CloseHandle	Cierra un archivo
read	ReadFile	Lee datos de un archivo
write	WriteFile	Escribe datos en un archivo
lseek	SetFilePointer	Desplaza el puntero del archivo
stat	GetFileAttributesEx	Obtiene varios atributos de un archivo

39

Llamadas al sistema (system calls)

- **La API Win32.** (*cont.*)
- **Para manejo de directorios:**

POSIX	Win32	Descripción
mkdir	CreateDirectory	Crea un nuevo directorio
rmdir	RemoveDirectory	Elimina un directorio vacío
link	(ninguno)	Win32 no soporta los enlaces
unlink	DeleteFile	Destruye un archivo existente
mount	(ninguno)	Win32 no soporta el montaje
umount	(ninguno)	Win32 no soporta el montaje

- **Para manejos varios:**

POSIX	Win32	Descripción
chdir	SetCurrentDirectory	Cambia el directorio de trabajo actual
chmod	(ninguno)	Win32 no soporta la seguridad (aunque NT sí)
kill	(ninguno)	Win32 no soporta las señales
time	GetLocalTime	Obtiene la hora actual

40

Estructura del Sistema Operativo

- Hasta ahora sólo hemos visto cómo es el SO “desde afuera”, en términos de sus funciones y sus aplicaciones. Veremos ahora cómo es el SO “por dentro”.
- Para cumplir con las funciones vistas anteriormente, los SO tienen implementada una estructura, la cual contiene todas estas funcionalidades.
- Las estructuras de un SO pueden clasificarse como:
- **Monolíticas.**
- **Estructuradas.**
  - Por capas.
  - Micro-kernel.
  - Cliente-servidor.

41

Estructura del Sistema Operativo

- **Estructura Monolítica.**
- Esta es la más común de las estructuras. El SO completo se ejecuta como un único programa en modo kernel. Se escribe como un conjunto de procedimientos, enlazados en un único gran programa binario ejecutable.
- Cuando se utiliza esta técnica, cada procedimiento tiene la libertad de llamar a cualquier otro; al tener miles de procedimientos que se pueden llamar entre sí sin restricción, se produce un sistema poco manejable y difícil de entender.
- En términos de ocultamiento de información, en esencia no hay nada: todos los procedimientos son visibles para cualquier otro procedimiento.
- Sin embargo, tienen una cierta estructura: para solicitar los servicios del SO, los parámetros se colocan en una pila y se ejecuta una *trap*.

42

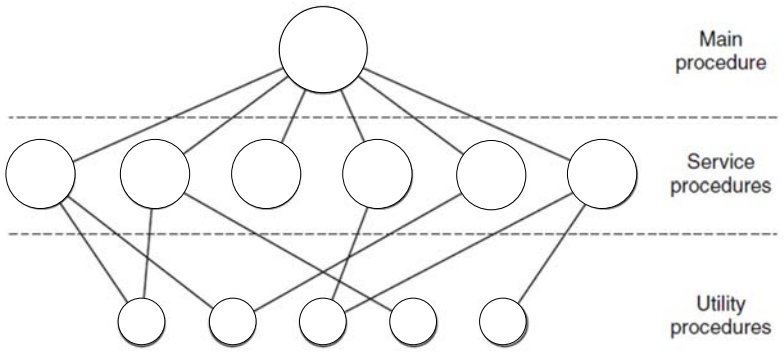
Estructura del Sistema Operativo

- **Estructura Monolítica.** (*cont.*)
- Esta instrucción cambia la máquina a modo kernel y transfiere el control al SO. Luego éste obtiene los parámetros de la pila y determina cuál es la llamada al sistema que debe ejecutar.
- Esta organización sugiere una estructura básica para el SO:
  - Un programa principal que invoca al procedimiento de servicio.
  - Un conjunto de procedimientos de servicios que ejecutan las llamadas al sistema.
  - Un conjunto de procedimientos utilitarios que ayudan a los procedimientos de servicio.
- Además del núcleo del SO que se carga al arrancar la computadora, muchos SO soportan extensiones que se pueden cargar, como los drivers de dispositivos de E/S y sistemas de archivos. Estos componentes se cargan por demanda.

43

Estructura del Sistema Operativo

- **Estructura Monolítica.** (*cont.*)



44

Estructura del Sistema Operativo

- **Estructura Monolítica.** (*cont.*)
- Una desventaja de este tipo de estructura es que un proceso de usuario también puede llamar a cualquier procedimiento del SO, por ej., a rutinas de E/S, lo que lo hace vulnerable a código erróneo o malicioso, y una falla en un procedimiento hace que todo el SO falle y "se caiga".
- Además, añadir una nueva característica implica la modificación de un gran programa compuesto por miles o millones de líneas de código fuente y de una infinidad de funciones.
- Ejemplos de SO con esta estructura:
  - MS-DOS.
  - IBM PC DOS.

45

Estructura del Sistema Operativo

- **Estructura por capas.**
- Con el soporte de hardware apropiado, los SO pueden dividirse en partes más pequeñas de lo que permitía MS-DOS. Esto lleva a mantener un control mucho mayor sobre la computadora y sobre el uso de los recursos por parte de los programas de usuario.
- Una forma de hacer modular un SO es mediante una estructura de capas: la capa inferior (nivel 0) es el hardware, y la capa superior (nivel N), la interfaz de usuario.
- Una capa del SO es una implementación de un objeto abstracto formado por datos y operaciones que manipulan estos datos. Entonces, cada capa consta de estructuras de datos y rutinas que los niveles superiores pueden invocar.
- Esta implementación permite el ocultamiento de información, ya que una capa superior sólo necesita saber qué hacen las rutinas de la capa inferior para solicitar sus servicios.

46

Estructura del Sistema Operativo

- **Estructura por capas.** (*cont.*)
- Las funciones de las capas serían entonces:
- **Capa Inferior:** normalmente es la encargada de crear procesadores virtuales para todos los procesos.
- **Siguiente capa:** encargada de administrar la memoria virtual (memoria principal + disco).
- **Capas sucesivas:** se aplican las abstracciones señaladas hasta llegar a la capa superior donde se ejecutan los procesos de usuario.
- En cada nivel se administra un dado recurso. Por lo tanto, a medida que se asciende en los niveles, se tiene un mayor número de tareas de administradas.
- Así, un proceso en el nivel N puede asumir que todas las tareas de administración en niveles inferiores están realizadas y, por lo tanto, puede utilizar los servicios provistos por ellos.

47

Estructura del Sistema Operativo

- **Estructura por capas.** (*cont.*)
- Ejemplo de SO con estructura por capas: **THE**.

Capa	Función
5	El operador
4	Programas de usuario
3	Administración de la E/S
2	Comunicación operador-proceso
1	Administración de memoria y tambor
0	Asignación del procesador y multiprogramación

- Construido en **T**echnische **H**ogeschool **E**indhoven, Holanda, por E.W. Dijkstra y sus estudiantes, en el año 1968.
- Ejecutaba en la computadora holandesa Electrologica X8, que tenía 32K palabras de 27 bits.

48

Estructura del Sistema Operativo

- **Estructura por capas.** (*cont.*)
- El esquema de capas planteado por THE era sólo una ayuda para el diseño, ya que todas las partes del SO se enlazaban en un solo programa ejecutable.
- La ventaja que presenta es la restricción de acceso de una capa únicamente a su capa inmediata inferior, lo que permite proteger el acceso al hardware por parte de los programas de usuario.
- Una desventaja es la dificultad en definir apropiadamente las funciones de cada capa. Dado que cada nivel sólo puede acceder a la capa inferior es necesario una planificación cuidadosa.
- Otra desventaja es la complejidad de determinar hasta qué capa se ejecutarán las funciones en modo kernel, y a partir de cuál en modo usuario.

49

Estructura del Sistema Operativo

- **Estructura micro-kernel.**
- Con la aproximación de capas, los desarrolladores tenían la libertad de elegir dónde "dibujar" el límite kernel-usuario. Tradicionalmente, todas las capas formaban parte del núcleo del SO, aunque no es necesario.
- De hecho, dejar "lo menos posible" en el núcleo implica reducir la cantidad de líneas de código, y por lo tanto, la cantidad de *bugs* del núcleo.
- La densidad de bugs por líneas de código depende del tamaño del módulo, de su antigüedad, entre otros factores, pero un valor estimado es de entre 2 y 10 bugs cada 1.000 líneas de código. Esto implica que un SO monolítico, con 5 millones de líneas de código, tiene entre 10.000 y 50.000 bugs en el kernel.
- Aunque no todos son "fatales", son suficientes como para poner un botón de *reset* en el frente de la computadora.

50

Estructura del Sistema Operativo

- **Estructura micro-kernel.** (*cont.*)
- La idea básica de diseñar un micro-kernel es lograr alta confiabilidad dividiendo el SO en módulos pequeños y bien definidos, uno de los cuales, el micro-kernel en sí, es el único que ejecuta en modo kernel, y el resto en modo usuario.
- En particular, al ejecutar cada *driver* de dispositivo como un proceso separado, un error en alguno de ellos puede hacer que falle sólo ese componente, sin hacer que falle el resto del sistema.
- Por ejemplo, un *bug* en el *driver* del dispositivo de audio puede provocar que el sonido se distorsione o se detenga, pero no va a hacer que el sistema completo "caiga".
- En contraste, en un SO monolítico, esta falla en el driver puede, por ejemplo, referenciar una dirección de memoria no válida, y como consecuencia, hacer caer el sistema completo.

51

Estructura del Sistema Operativo

- **Estructura micro-kernel.** (*cont.*)
- Por décadas, muchos micro-kernel se implementaron; con la excepción de OS X, que está basado en el micro-kernel de Mach, ningún SO de escritorio los utiliza.
- Sin embargo, son predominantes en el mercado de los SO de tiempo real, en aplicaciones industriales, aviación y militares que son de misión crítica, y tienen altos requerimientos de alta confiabilidad.
- Algunos ejemplos de micro-kernel son:
  - **PikeOS:** automotores, aviación, medicina, ferrocarriles, industria.
  - **QNX:** comprado por BlackBerry en 2010. SO que lo utilizan: BlackBerry 10. Otras aplicaciones: automotores, medicina.
  - **Symbian:** diseñado originalmente para PDA, luego utilizado en smartphones de Nokia, Samsung, Motorola y Sony Ericsson.
  - **MINIX 3:** desarrollado A. Tanenbaum (¡quien escribió el libro de SO!). Aplicaciones: sistemas embebidos (Intel ME) y fines académicos.

52

Estructura del Sistema Operativo

- **Estructura micro-kernel.** (*cont.*)
- **MINIX 3.**
- Tiene alrededor de 12.000 líneas de código hecho en C, y unas 1.400 en leguaje de máquina para funciones de muy bajo nivel, tales como la captura de interrupciones y cambios de procesos.
- Este SO se estructura en cuatro capas, donde la inferior es el núcleo, el cual se encarga de manejar las interrupciones, la planificación de procesos (Mód. III), y la comunicación entre procesos. Fuera del núcleo, este SO tiene tres capas de procesos, las cuales ejecutan en modo usuario.
- La primera de estas capas contiene los *drivers* de dispositivos. Dado que ejecuta en modo usuario, los *drivers* no tienen acceso directo al hardware, sino que solicitan el servicio al núcleo. Esto permite verificar que no se acceda erróneamente a otro hardware que no sea el solicitado.

53

Estructura del Sistema Operativo

- **Estructura micro-kernel.** (*cont.*)
- **MINIX 3.** (*cont.*)
- Por encima de los drivers se encuentra la capa de *servidores*. Esta realiza la mayor parte del trabajo del SO. Uno o más servidores de archivos manejan el o los sistemas de archivos, el servidor de procesos crea, destruye y administra los procesos, etc..
- Los programas de usuario obtienen los servicios del SO mediante envío de mensajes cortos a estos servidores, pidiendo las llamadas al sistema de POSIX.
- Un servidor interesante es el servidor de reencarnación, cuyo trabajo es verificar si los drivers y servidores funcionan correctamente. Si detecta una falla en alguno, lo reemplaza automáticamente sin la intervención del usuario. Esto le brinda alta confiabilidad y alta disponibilidad.

54

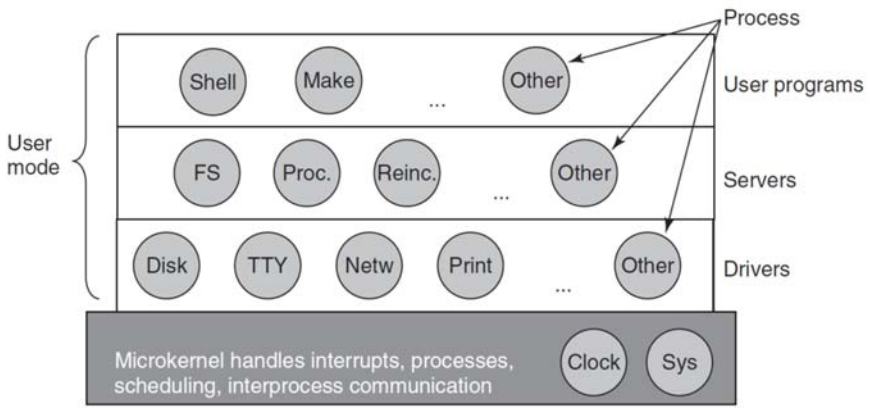
Estructura del Sistema Operativo

- **Estructura micro-kernel.** (*cont.*)
- **MINIX 3.** (*cont.*)
- Una idea que está en parte relacionada con el concepto de micro-kernel es colocar en el núcleo el **mecanismo** para hacer algo, pero no la **directiva**.
- Por ejemplo, un algoritmo simple para la planificación de procesos podría asignar prioridades a cada proceso y hacer que el núcleo ejecute el de mayor prioridad.
- El **mecanismo** para este algoritmo, en el núcleo (modo kernel), es **buscar el proceso de mayor prioridad** y ejecutarlo.
- La **directiva**, **asignar las prioridades a los procesos**, puede realizarse mediante una tarea que ejecute en modo usuario.
- De esta manera, el mecanismo y la directiva pueden desacoplarse, logrando reducir el tamaño del núcleo.

55

Estructura del Sistema Operativo

- **Estructura micro-kernel.** (*cont.*)
- **MINIX 3.** (*cont.*)
- Esquema de capas de MINIX 3:



©Tanenbaum, 2015

56

Estructura del Sistema Operativo

- **Estructura cliente-servidor.**
- Una ligera variación del modelo de micro-kernel es diferenciar dos clases de procesos: los **servidores**, cada uno de los cuales proporciona cierto servicio, y los **clientes**, que utilizan estos servicios.
- Este modelo se conoce como cliente-servidor. Usualmente, la capa inferior es un micro-kernel, pero no siempre es requerido, ya que la esencia es la presencia de procesos cliente y procesos servidor.
- La comunicación entre cliente y servidor se realiza mediante el paso de mensajes. Un cliente construye un mensaje indicando qué necesita y lo envía al servicio apropiado. El servidor captura el mensaje, hace el trabajo requerido y envía una respuesta.
- Si ambos procesos ejecutan en un mismo equipo, se pueden hacer algunas optimizaciones, pero el paso de mensajes es la base.

57

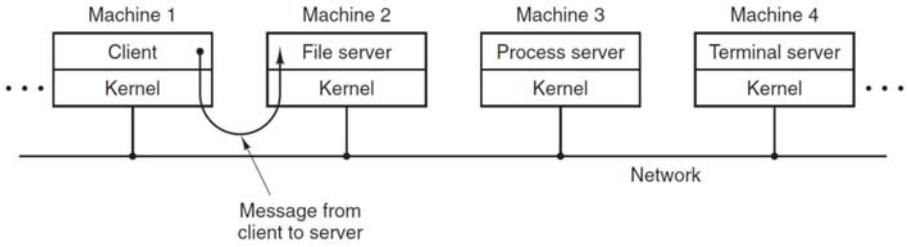
Estructura del Sistema Operativo

- **Estructura cliente-servidor.** (*cont.*)
- Una generalización subyacente, entonces, es hacer que los clientes y servidores ejecuten en computadoras diferentes, conectadas con alguna tecnología de comunicación, como un bus de alta velocidad (MP) una red local (MC) o extendida (SD).
- Como los clientes se comunican con los servidores mediante mensajes, no necesitan saber si estos mensajes se manejan en forma local en sus propios equipos o si se envían a través de una red a servidores en un equipo remoto.
- Lo que “le importa” al cliente es lo mismo en ambos casos: poder enviar sus peticiones y recibir las respuestas. Por lo tanto, el modelo cliente-servidor es una abstracción que se puede utilizar tanto para un solo equipo, como para una red de equipos.
- Las principales aplicaciones de este modelo se verán en detalle en SO II. (*to be continued…*)

58

Estructura del Sistema Operativo

- **Estructura cliente-servidor.** (*cont.*)
- Esquema básico del modelo cliente-servidor en una red de computadoras:



59

Fin del Módulo I

60