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

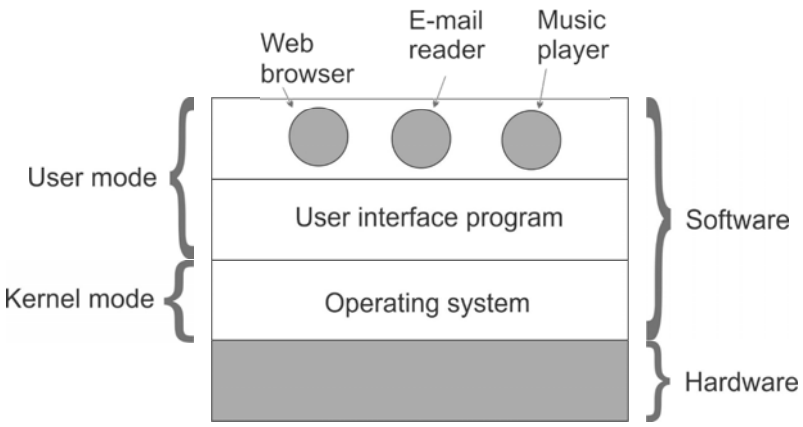
Definición de Sistema Operativo

- **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

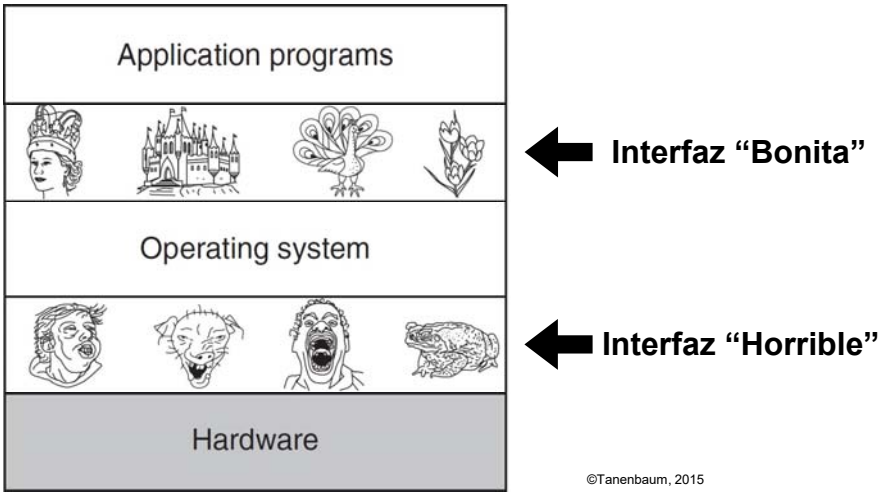
Definición de Sistema Operativo

- **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

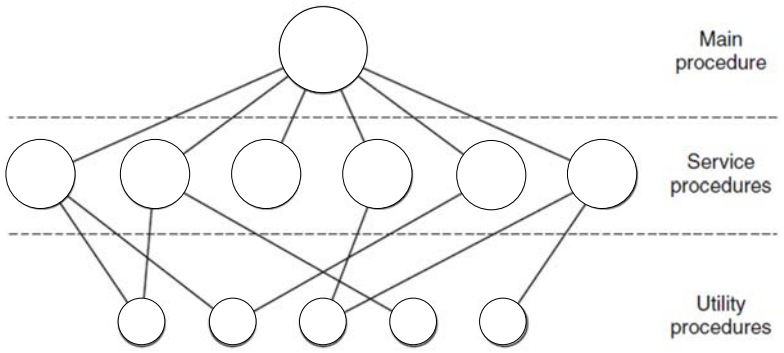
Estructura del Sistema Operativo

- **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

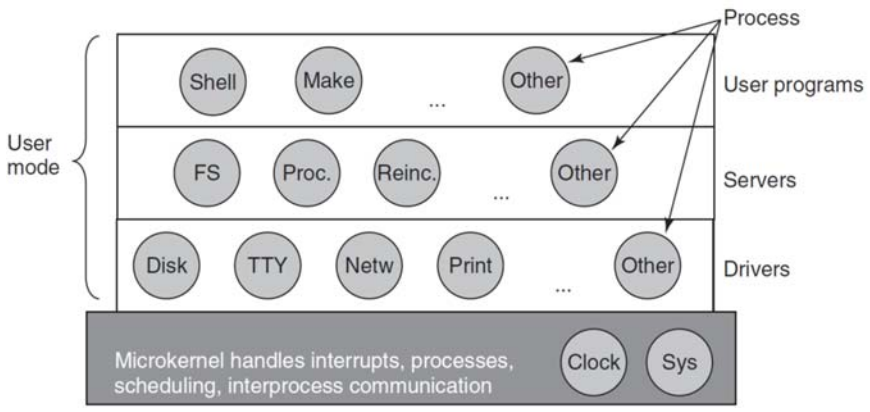
Estructura del Sistema Operativo

- **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

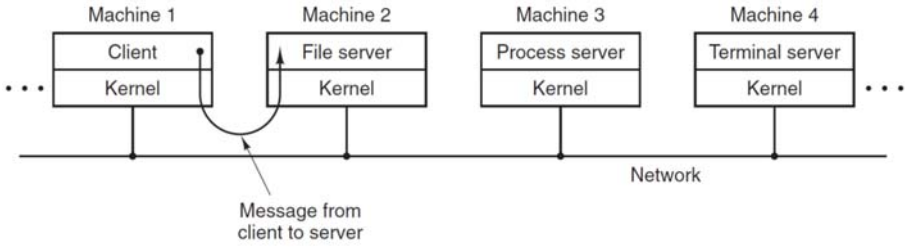
Estructura del Sistema Operativo

- **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

Sistemas Operativos I

Módulo II

PROCESOS E HILOS

1

Temas del Módulo II

- **Procesos.**
  - Concepto de proceso.
  - Creación de un proceso.
  - Terminación de un proceso.
  - Jerarquías de procesos.
  - Estados de un proceso.
  - Implementación de los procesos.
    - Estructuras de control del SO.
    - Componentes de un proceso.
  - Cambio de proceso.
- **Hilos.**
  - Uso de hilos.
  - Modelo clásico de hilos.
  - Hilos en POSIX.
  - Hilos en el espacio de usuario.
  - Hilos en el espacio de núcleo.

2

Procesos

- **Concepto de proceso.**
- Todas las computadoras modernas ofrecen la posibilidad de realizar múltiples tareas “al mismo tiempo”; esta característica está tan naturalizada que no somos completamente conscientes de este hecho.
- Un ejemplo es el caso de una computadora personal, cuando arranca el sistema se inician muchos procesos en forma oculta, que por lo general el usuario desconoce.
- Típicamente, un antivirus que actualiza las definiciones de virus, un cliente de correo electrónico que espera el correo entrante, etc. Todo esto mientras el usuario navega en Internet.
- Toda esta actividad se tiene que administrar, y en estos casos el concepto de proceso es muy útil.

3

Procesos

- **Concepto de proceso.** (*cont.*)
- Los procesos son una de las más antiguas e importantes abstracciones que proporcionan los sistemas operativos: brindan la capacidad de operar concurrentemente, incluso cuando sólo hay una CPU disponible. Esto es, convierten una CPU en varias CPU virtuales.
- En el modelo de procesos, todo el software ejecutable, incluyendo el SO, se organiza en **procesos secuenciales** (procesos, para abreviar). Un proceso no es más que una instancia de un programa en ejecución, incluyendo su contexto de ejecución (contador de programa, registros, variables).
- Conceptualmente, cada proceso tiene su propia CPU virtual; en la realidad, la CPU real conmuta rápidamente de un proceso a otro. Pero es más fácil pensar en un conjunto de procesos ejecutando en paralelo.

4

Procesos

- **Concepto de proceso.** (*cont.*)
- Esta conmutación rápida de un proceso a otro se conoce como multiprogramación. La cantidad de procesos que pueden almacenarse en memoria principal determina el grado de multiprogramación de un sistema.

5

Procesos

- **Concepto de proceso.** (*cont.*)
- La diferencia entre un proceso y un programa es sutil, pero crucial. Una analogía para entenderla, sería la siguiente: una **persona** tiene que cocinar un estofado; para ello cuenta con una **receta**, una cocina y los **ingredientes** (verduras, carne, condimentos).
- La **receta** sería el programa, es decir el algoritmo expresado en una notación adecuada, los pasos a seguir. La **persona** es el procesador (CPU) y los **ingredientes**, los datos de entrada.
- El **proceso**, entonces, es la **actividad** que consiste en que la persona vaya leyendo la receta, obteniendo los ingredientes y cocinando el estofado.

6

Procesos

- **Concepto de proceso.** (*cont.*)
- Imaginemos que el hijo de esta persona entra a la cocina corriendo y gritando que se lastimó la rodilla. Entonces, la persona anota hasta dónde llegó con la receta (**guarda el estado del proceso**), saca un spray antiséptico del botiquín y sigue las instrucciones del envase para aplicárselo al hijo.
- Aquí el procesador conmuta de un proceso (cocinar el estofado) a uno de mayor prioridad (aplicar el antiséptico), cada uno con un programa distinto (la receta y las instrucciones en el envase del antiséptico).
- Cuando se desocupa de la rodilla lastimada, retoma la cocción del estofado en el punto que había anotado.

7

Procesos

- **Concepto de proceso.** (*cont.*)
- La clave es que **un proceso es una actividad**: tiene un **programa**, **datos** de entrada y salida (¡el estofado!), y un **estado**. Varios procesos pueden compartir un procesador mediante un algoritmo de planificación para determinar cuándo detener la ejecución de un proceso para dar servicio a otro.

8

Procesos

- **Creación de un proceso.**
- Los SO necesitan cierta manera de crear procesos. En sistemas pequeños, como los embebidos, se puede saber con exactitud los procesos que se el sistema va a requerir al inicio del mismo; pero en sistemas de propósitos generales, es necesario poder crear procesos dinámicamente, a medida que son requeridos.
- Existen cuatro eventos principales que crean eventos:
  - El arranque del sistema.
  - La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.
  - Una petición de usuario para crear un proceso.
  - El inicio de un trabajo por lotes.

9

Procesos

- **Creación de un proceso.** *(cont.)*
- En el arranque del sistema: Cuando arranca un SO se crean varios procesos. Algunos son procesos en primer plano, es decir, que interactúan con los usuarios y realizan algún trabajo para ellos. Otros son en segundo plano, esto es, no están asociados con usuarios específicos sino con una tarea específica; éstos se conocen como demonios (daemons).
- Desde un proceso: Posterior al arranque se pueden crear otros procesos. A menudo, un proceso en ejecución puede emitir llamadas al sistema para crear procesos que le ayuden en su trabajo, típicamente cuando se puede plantear un esquema de varios procesos relacionados entre sí, pero independientes en otros aspectos.

10

Procesos

- **Creación de un proceso.** *(cont.)*
- Por pedido de un usuario: En sistemas interactivos, un usuario puede iniciar un programa escribiendo un comando o haciendo clic en un ícono. Cualquiera de estas acciones inicia un proceso y ejecuta el programa asociado.
- Un trabajo por lotes: El caso de la creación de un de proceso por inicio de un trabajo por lotes se aplica a sistemas de procesamiento que se encuentran en las mainframes grandes; aquí, los usuarios envían trabajos al sistema, generalmente en forma remota. Cuando el SO tiene los recursos, crea un proceso y ejecuta el siguiente trabajo en la cola de entrada.

11

Procesos

- **Terminación de un proceso.**
- Luego de su creación, un proceso empieza a ejecutar y realiza el trabajo para el que fue destinado. Pero, como en la vida misma, nada es para siempre: tarde o temprano, un proceso terminará su ejecución debido a alguna de las siguientes condiciones:
  - Salida normal (voluntaria).
  - Salida por error (voluntaria).
  - Error fatal (involuntaria).
  - Eliminado por otro proceso (involuntaria).
- Salida normal: La mayoría de los procesos terminan su ejecución porque han concluido su trabajo. Por ejemplo, cuando un compilador ha compilado un programa, ejecuta una llamada al sistema para indicar al SO que ha terminado.

12

Procesos

- **Terminación de un proceso.** *(cont.)*
- Salida por error: La segunda razón de terminación es que el proceso descubre un error en su ejecución. Por ejemplo, si un usuario ejecuta un comando para compilar un archivo fuente y éste último no existe; el compilador, al no encontrar el archivo, simplemente termina.
- Error fatal: Se produce un error fatal, a menudo, debido a un error en el programa, tales como ejecutar una instrucción ilegal, hacer referencia a un lugar de memoria no existente o restringido, o bien una división por cero.
- Eliminado por otro proceso: Un proceso puede invocar una llamada al sistema que indique al SO que elimine otros procesos. Para ello, el proceso eliminador debe contar con la autorización necesaria para realizar la eliminación.

13

Procesos

- **Jerarquías de procesos.**
- En algunos sistemas, cuando un proceso crea otro, padre e hijo continúan asociados de ciertas formas. El proceso hijo puede crear por sí mismo más procesos, formando una jerarquía de procesos.
- En POSIX, un proceso y todos sus hijos, y sus descendientes, forman un grupo de procesos. Cuando un usuario envía una señal del teclado, ésta se envía a todos los miembros del grupo actualmente asociado al teclado. Cada proceso puede atrapar la señal, ignorarla o tomar la acción predeterminada.
- Otro ejemplo del papel de la jerarquía de procesos es la forma en la que POSIX inicializa al encender la computadora: hay un proceso especial, *init*, en la imagen de inicio. Cuando ejecuta, lee un archivo (*/etc/ttytab*) que le indica cuántas terminales hay.

14

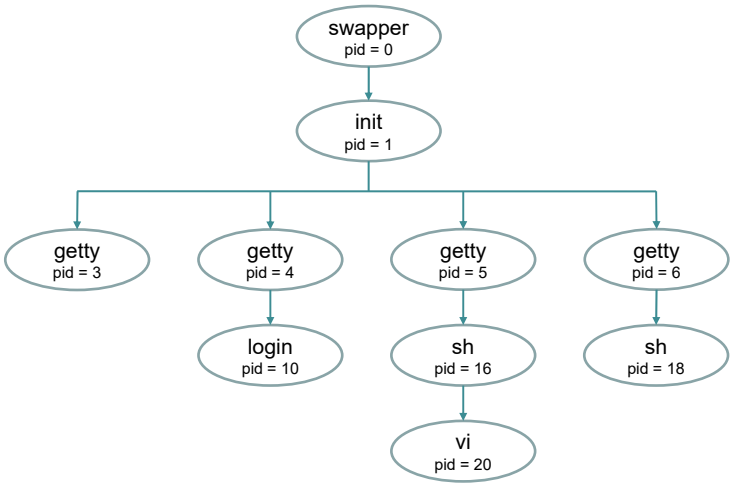
Procesos

- **Jerarquías de procesos.** (*cont.*)
- Después utiliza *fork* para crear un proceso (*getty*) por cada terminal; estos procesos esperan a que alguien inicie sesión. Cuando un usuario entra al sistema se ejecuta *login* con el nombre como argumento, y si el inicio de sesión tiene éxito, se ejecuta un *shell* (*sh*) para aceptar comandos.
- El *shell* del usuario se especifica en el archivo */etc/passwd*; éste contiene información completa acerca del usuario: nombre de la cuenta (*login*), contraseña (encriptada), UID, GID, nombre completo del usuario, directorio de trabajo y el intérprete de comando que utiliza.
- El *shell*, entonces, lanzará un *fork* y un *execve* por cada comando, iniciando así más procesos. Por ende, todos los procesos en el sistema pertenecen a un solo árbol, con *init* en la raíz.

15

Procesos

- **Jerarquías de procesos.** (*cont.*)
- El árbol de procesos en POSIX se arma de la siguiente manera:



16

Procesos

- **Jerarquías de procesos.** (*cont.*)
- Windows, en cambio, no incorpora el concepto de jerarquía de procesos; todos los procesos son iguales.
- La única sugerencia de jerarquía de procesos es que, cuando se crea un proceso, el padre recibe un indicador especial, llamado **manejador** (*handler*), que puede utilizar para controlar al hijo.
- Sin embargo, tiene la libertad de pasar este indicador a otros procesos, con lo cual invalida la jerarquía. Los procesos en POSIX no pueden desheredar a sus hijos.

17

Procesos

- **Estados de un proceso.**
- Aunque cada proceso es una entidad independiente, a menudo necesitan interactuar con otros. Un proceso puede generar una salida que otro utiliza como entrada.
- Dependiendo de la velocidad relativa de los procesos, puede ocurrir que uno esté listo para ejecutar pero que aún no haya una entrada de datos disponible. En este caso, el proceso debe bloquearse hasta que haya una entrada.
- También es posible que un proceso que esté en ejecución se detenga debido a que el SO ha decidido asignar la CPU a otro proceso por un cierto tiempo.
- Estas dos situaciones son completamente distintas. En el primer caso, la suspensión es inherente al problema (no se dispone del recurso para seguir), mientras que, en el segundo, es un tecnicismo (no hay suficientes CPU).

18

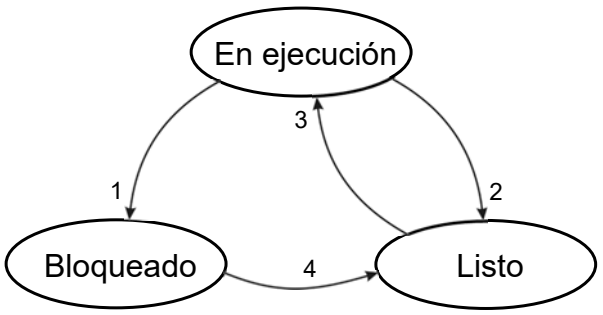
Procesos

- **Estados de un proceso.** (*cont.*)
- Entonces, un proceso puede encontrarse en tres estados posibles:
  - **En ejecución** (está listo y usando la CPU en ese instante).
  - **Listo** (ejecutable; se detuvo temporalmente para dar lugar a otro proceso).
  - **Bloqueado** (no puede continuar hasta que ocurra un evento externo).
- En sentido lógico, los dos primeros son similares: en ambos el proceso está listo para ejecutar, sólo que en el segundo no hay temporalmente una CPU disponible.
- El tercer estado es distinto de los dos primeros en cuanto a que el proceso no puede ejecutar, incluso habiendo una CPU disponible.

19

Procesos

- **Estados de un proceso.** (*cont.*)
- Hay cuatro transiciones posibles entre los tres estados:



1. El proceso se bloquea para recibir entrada.
2. El planificador selecciona otro proceso.
3. El planificador selecciona este proceso.
4. La entrada ya está disponible.

20

Procesos

- **Estados de un proceso.** (*cont.*)
- Si utilizamos el modelo de procesos, es mucho más fácil pensar que está ocurriendo dentro del sistema. Algunos procesos ejecutan programas que llevan a cabo los comandos que escribe un usuario; otros, son parte del SO y se encargan de cumplir con las peticiones de servicios como las *system calls*.
- Por ejemplo, cuando ocurre una IRQ de disco, el SO toma la decisión de dejar de ejecutar el proceso actual y ejecutar el proceso de disco que está bloqueado esperando esa IRQ.
- Entonces, en vez de pensar en IRQ's, podemos pensar en procesos de usuario, procesos de disco, procesos de terminal, etc., que se bloquean cuando están esperando a que algo ocurra.

21

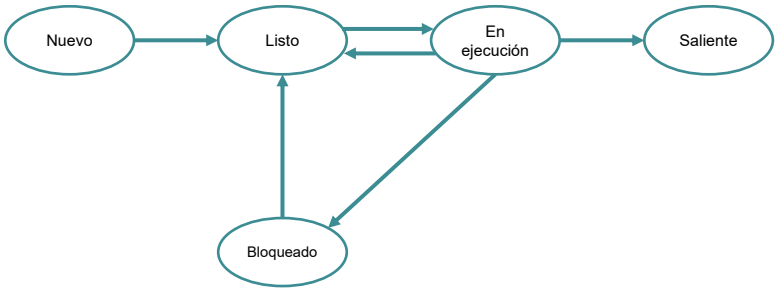
Procesos

- **Estados de un proceso.** (*cont.*)
- Podemos refinar el modelo anterior agregando dos estados muy útiles para la gestión de procesos:
  - **Nuevo:** se corresponde con un proceso que acaba de ser creado. En este estado, el SO crea todas las estructuras de datos necesarias para gestionar al nuevo proceso, pero aún no le ha sido asignado espacio en memoria principal.
  - **Saliente:** al terminar un proceso es movido a éste estado, donde deja de ser elegible para ejecutar. El SO puede mantener las estructuras de datos asociadas para extraer información, por ejemplo, de auditoría, tales como el tiempo de ejecución, recursos utilizados, etc., lo cual es útil para el análisis de rendimiento del sistema.

22

Procesos

- **Estados de un proceso.** (*cont.*)
- El diagrama de estados, incorporando éstos últimos, sería el siguiente:



Stallings, 2005

23

Procesos

- **Estados de un proceso.** (*cont.*)
- Mencionamos que los procesos se almacenan en memoria principal para poder ser ejecutados. Normalmente, los SO's tienen definido un grado de multiprogramación, que lo establece en cierta forma el rendimiento del mismo.
- En algunos casos hay procesos que se bloquean esperando un evento, típicamente de E/S, el cual puede tomar más tiempo de lo habitual, debido por ejemplo, a un gran volumen de datos a transferir.
- Esto nos lleva a tener procesos en memoria principal que no están en ejecución, ocupando espacio que podría se aprovechado por otro proceso que está listo y que por falta de espacio en memoria principal, continúa, por ejemplo, en el estado Nuevo.

24

Procesos

- **Estados de un proceso.** (*cont.*)
- Esta situación plantea la necesidad de quitar esos procesos bloqueados de memoria y llevarlos a un almacenamiento secundario (disco).
- La operación que lleva al proceso de memoria a disco se denomina *swapping* o *intercambio*.
- Tendremos, entonces, un conjunto de procesos bloqueados esperando un evento, almacenados en disco. Una vez que dicho evento se produce y/o hay disponibilidad de memoria, el proceso puede volver a ésta.
- También es posible que un proceso esté en el estado Listo y el SO decida otorgar el uso de CPU a un proceso de mayor prioridad, pero que no puede ser ejecutado por falta de espacio en memoria.

25

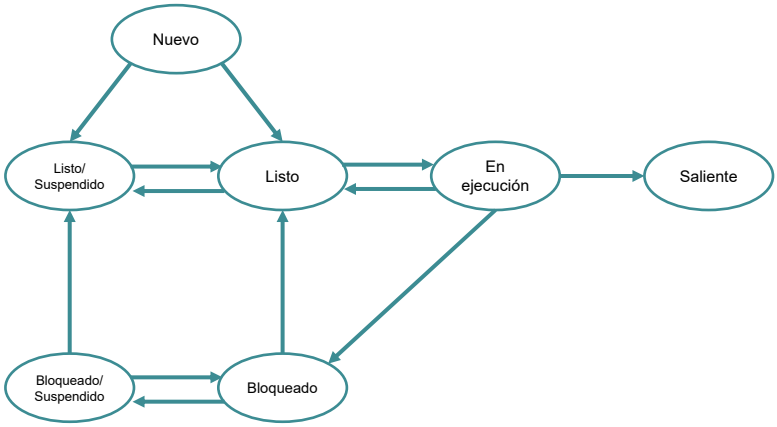
Procesos

- **Estados de un proceso.** (*cont.*)
- Entonces, se mueve al proceso que está listo a disco, y se otorga el espacio al proceso prioritario. Esto también implica una operación de *swapping*, pero desde el estado Listo.
- Por lo tanto, tendremos dos nuevos estados en el diagrama:
  - **Bloqueado/Suspendido:** el proceso está en almacenamiento secundario y esperando un evento.
  - **Listo/Suspendido:** el proceso está en almacenamiento secundario pero disponible para su ejecución en cuanto sea cargado en memoria principal.

26

Procesos

- **Estados de un proceso.** (*cont.*)
- El diagrama, incorporando los estados suspendidos, sería el siguiente:



Stallings, 2005

27

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.**
- Si el SO se encarga de la gestión de procesos y recursos, debe disponer de información sobre el estado actual de cada proceso y cada recurso. El mecanismo universal para esto es construir y mantener tablas de información sobre cada entidad que gestiona.
- Básicamente, debe administrar cuatro entidades primordiales:
  - Memoria.
  - Entrada/Salida.
  - Archivos.
  - Procesos.
- Entonces, el SO construirá tablas para administrar cuatro entidades.

28

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.** (*cont.*)
- Las tablas de memoria se usan para mantener un registro tanto de la memoria principal como de la secundaria. Parte de la memoria principal está reservada para el uso del SO; el resto está disponible para los procesos.
- Las tablas de memoria deben incluir la siguiente información:
  - Las reservas de memoria principal por parte de los procesos.
  - Las reservas de memoria secundaria por parte de los procesos.
  - Los atributos de protección que restringen el uso de memoria principal y secundaria.
  - La información necesaria para manejar la memoria virtual.

29

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.** (*cont.*)
- Las tablas de E/S se utilizan para gestionar los dispositivos de E/S (discos, placas de red, etc.). Contiene información sobre el estado de uso de los dispositivos como también de las colas de procesos que esperan para su uso.
- Un dispositivo puede estar disponible o asignado a un proceso en particular.
- Si una operación de E/S se está realizando, el SO necesita conocer el estado de la operación y la dirección de memoria principal del área usada como fuente o destino de la transferencia de E/S.

30

Procesos

- **Implementación de los procesos.**
- **Estructuras de control del SO.** (*cont.*)
- Las tablas de archivos proveen información sobre la existencia de archivos, su ubicación en almacenamiento secundario, su estado actual (abiertos, bloqueados, compartidos), y otros atributos (permisos, fechas y horas de creación).
- Esta información se puede gestionar a través del sistema de archivos, una funcionalidad especial del SO.
- Por último, las tablas de procesos son utilizadas para la administración de los procesos.
- Existe una tabla primaria de procesos, a la cual llamaremos simplemente **tabla de procesos**. Cada entrada de ésta tabla contiene una referencia a cada proceso en el sistema.

31

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.**
- Ahora bien, ¿cómo es la representación física de un proceso? Como mínimo, un proceso debe contar con los siguientes componentes:
  - Un programa o un conjunto de programas a ejecutar, esto es, código ejecutable.
  - Asociados con los programas, posiciones de memoria para los datos de variables locales y globales y de cualquier constante definida.
  - Una pila, esto es, un espacio adicional de memoria incluida en la ejecución del programa, utilizado para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas al sistema.
  - Atributos utilizados por el SO para controlar el proceso. El conjunto de éstos se denomina **bloque de control del proceso** (**PCB**, su sigla en inglés).
- El conjunto formado por el programa, datos, pila y atributos se denomina **imagen del proceso**.

32

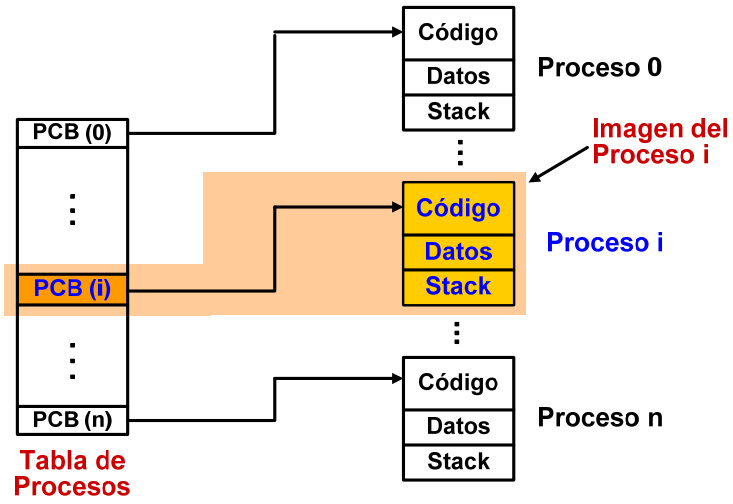
Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- La posición de la imagen del proceso dependerá de cómo el SO gestione la memoria. Usualmente, se mantiene en memoria secundaria (disco).
- Para que el SO pueda gestionar el proceso, al menos una pequeña parte de su imagen se debe mantener en memoria principal; para su ejecución, una parte mayor o completo.
- La tabla de procesos, en cambio, debe residir en memoria principal.
- Cada entrada en la tabla de procesos, contiene entonces, al menos un puntero a la imagen del proceso. Si ésta contiene múltiples bloques, puede tener referencias cruzadas entre las tablas de memoria.

33

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)



34

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Se mencionó que los **atributos del proceso** están contenidos en el bloque de control de proceso (**PCB**). Podemos agrupar la información de éste último en tres categorías generales:
  - Identificación del proceso.
  - Información de estado del procesador.
  - Información de control del proceso.

35

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Identificación del proceso: son números generados por el SO que contienen la identificación única del proceso (PID), la identificación del proceso que lo creó (proceso padre, PPID), y la identificación del usuario del proceso (UID).
- Información de estado del procesador: es el contenido de los registros del procesador:
  - Registros visibles por el usuario.
  - Registros de estado y control.
  - Puntero de pila.

36

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Información de control de proceso: necesaria para poder controlar y coordinar las actividades de los diversos procesos del sistema:
  - Información de estado y de planificación: necesaria para que el SO pueda analizar las funciones de planificación: estado del proceso, prioridad, parámetros de planificación, eventos.
  - Estructuras de datos: un proceso puede estar enlazado con otros en una cola o cualquier otra estructura, por ejemplo, en los estados de espera, o para indicar una relación padre-hijo.

37

Procesos

- **Implementación de los procesos.**
- **Componentes de un proceso.** (*cont.*)
- Información de control de proceso: (*cont.*)
  - Comunicación entre procesos: pueden asociarse banderas, señales y mensajes relativos a la comunicación entre procesos independientes.
  - Privilegios de proceso: de acuerdo a la memoria que van a acceder y los tipos de instrucciones que pueden ejecutar, como también el acceso a funcionalidades de sistema.
  - Gestión de memoria: conjuntos de punteros a tablas de segmentos del proceso que describen la asignación de memoria (mód. IV).
  - Apropiación de recursos y utilización: indica los recursos asignados y un histórico de utilización de los mismos.

38

Procesos

- **Cambio de proceso.**
- En el Módulo Introductorio vimos las técnicas de comunicación entre los componentes de una computadora, en particular las IRQ. La rutina IH se encargaba de verificar si se atendía o no la IRQ. ¿Cómo impacta esto a la hora de manejar varios procesos?
- Un cambio de proceso se produce cuando, por alguna causa, el SO retira de ejecución el proceso que está haciendo uso de la CPU, e instala otro proceso diferente.
- Las causas que pueden llevar a un cambio de proceso son las siguientes:
  - Interrupción: es producida por un evento externo al proceso. Las interrupciones pueden ser:
    - De reloj: cuando termina el tiempo de uso ininterrumpido de CPU; el proceso cambia al estado Listo y otro proceso pasará al estado En ejecución.

39

Procesos

- **Cambio de proceso.** (*cont.*)
  - Interrupción: (*cont.*)
    - De E/S: cuando un dispositivo indica al SO que ha completado una petición, por ejemplo, bloques de datos de un disco. Se mueve todos los procesos que esperaban este evento al estado Listo, y el SO decide si reanuda la ejecución del proceso interrumpido o si otorga el uso de CPU a otro proceso con mayor prioridad:
    - Por fallo de memoria: se produce cuando la CPU se encuentra con una referencia a una dirección de memoria virtual que no se encuentra en memoria principal. El proceso se bloquea hasta que el SO trae de disco el bloque que contiene la referencia y se pone en ejecución otro proceso.

40

Procesos

- **Cambio de proceso.** (*cont.*)
  - Trap: la produce un evento asociado a la ejecución de una instrucción del proceso que lleva a una condición de error. El SO detecta si esta condición es irreversible, en cuyo caso fuerza la terminación del proceso y elige otro para ejecutar.
  - Llamada al sistema: es una solicitud explícita de cambio de proceso. En este caso, un proceso del SO es quien pasa a hacer uso de la CPU.

41

Procesos

- **Cambio de proceso.** (*cont.*)
- Cuando se interrumpe la ejecución de un proceso, si el SO determina que el proceso interrumpido dejará de utilizar la CPU, se debe guardar el estado de ejecución del mismo para poder retomarlo en otro momento.
- Entonces, los pasos para un cambio de proceso son:
  - Salvar el estado del procesador.
  - Actualizar el bloque de control del proceso actual (estado).
  - Mover el proceso a la cola apropiada (Listo, Bloqueado).
  - Seleccionar un nuevo proceso a ejecutar.
  - Actualizar el bloque de control del proceso elegido (estado Ejecutando).
  - Actualizar estructuras de datos de gestión de memoria.
  - Restaurar el estado del procesador que tenía el proceso seleccionado.

42

Procesos

- **Cambio de proceso.** (*cont.*)
- Si se produce una interrupción, el SO debe determinar la causa de la misma y ejecutar la rutina de servicio de interrupción correspondiente. Para ello coloca en el contador de programa la dirección de comienzo de la rutina de manejo de interrupciones, y cambia de modo usuario a modo kernel, de manera que el código de tratamiento de la interrupción pueda incluir instrucciones privilegiadas.
- Dependiendo de la naturaleza de la interrupción, se puede producir o no un cambio de proceso. Al cambiar de modo de ejecución, el proceso que fue interrumpido no cambió aún de estado Ejecutando, por lo que, si la interrupción no produce un cambio de proceso, se retoma su ejecución.

43

Procesos

- **Cambio de proceso.** (*cont.*)
- Esta situación se conoce como cambio de modo, en donde la salvaguarda del estado y su posterior restauración representan sólo una ligera sobrecarga. No así un cambio de proceso completo, ya que el SO deberá realizar las tareas descriptas anteriormente, las cuales insumen una cantidad de ciclos de CPU varios órdenes de magnitud mayor.

44

Hilos

- **Uso de hilos.**
- En los SO tradicionales, cada proceso tiene un espacio de direcciones y un sólo hilo de ejecución. Sin embargo, hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones.
- La principal razón de tener hilos es que en muchas aplicaciones se desarrollan varias actividades a la vez: un modelo de programación con ejecución cuasi-paralelo.
- Un segundo argumento es que son más ligeros que los procesos, son más rápidos para crear y destruir, típicamente de 10 a 100 veces más.
- Una tercera razón está relacionada con el rendimiento: el uso de hilos permite solapar tareas que usen sólo CPU con operaciones de E/S.
- Por último, en sistemas con múltiples procesadores es posible implementar el verdadero paralelismo.

45

Hilos

- **Uso de hilos.** (*cont.*)
- La diferencia de tener varios procesos o varios hilos es que en el caso de varios procesos tenemos varios espacios de direcciones separados, que para poder compartirlos es necesaria la intervención del SO.
- En cambio, los hilos comparten un mismo espacio de direcciones, por lo que para realizar la comunicación entre hilos no es necesario invocar llamadas al sistema, ahorrando así sobrecarga en la ejecución de la aplicación.
- Consideremos el caso de un programa **procesador de textos**. Éstos realizan varias tareas a la vez: muestran en pantalla el texto que se escribe, capturan los caracteres y los clics de mouse para hacer alguna acción en particular, realizan copias de respaldo del archivo que se está editando, etc.

46

Hilos

- **Uso de hilos.** (*cont.*)
- Si estas tareas se realizaran en un único proceso, cada vez que se inicie un respaldo en disco del texto se ignorarían los comandos de teclado y mouse hasta que termine el respaldo; o bien el respaldo ser interrumpiría al detectar entradas de teclado. El usuario considerará esto como un rendimiento pobre de la aplicación.
- Tampoco funcionaría utilizar varios procesos, ya que cada uno necesitará acceder al archivo del texto, lo que implica cambiar la asignación del recurso entre los procesos, lo cual implica una sobrecarga importante.
- Si en cambio, se utiliza un proceso con varios hilos, cada uno de ellos ejecutando una tarea de las descripta, estas pueden incluso ejecutar en un cuasi-paralelo, además de la ventaja de compartir los recursos asignados al proceso.

47

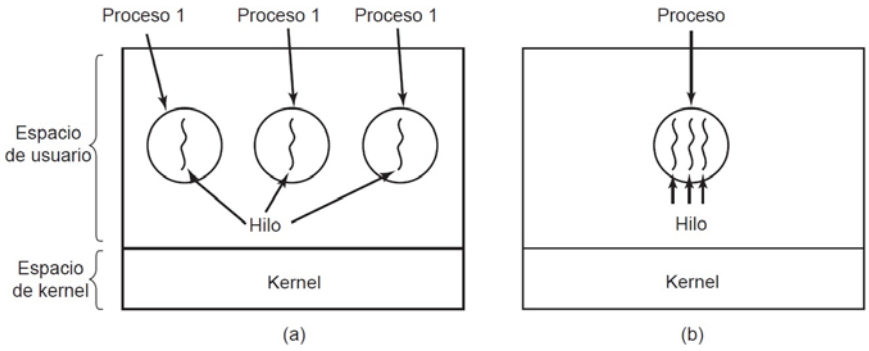
Hilos

- **Modelo clásico de hilos.**
- El modelo de proceso se basa en 2 conceptos independientes: agrupamiento de recursos y ejecución.
- Una manera de ver un proceso es como una forma de agrupar recursos relacionados. Tiene un espacio de direcciones que contiene código y datos del programa, archivos abiertos, procesos hijos, alarmas, señales, etc. El proceso es, por tanto, una unidad de posesión de recursos.
- Otro concepto que tiene el proceso es el *hilo de ejecución*, llamado simplemente **hilo**. Éste tiene un contador de programa, registros para almacenar variables, una pila, etc. El hilo es, entonces, una unidad de ejecución.
- Lo que agregan los hilos al modelo de procesos es permitir que se lleven a cabo varias ejecuciones independientes en el mimo entorno del proceso.

48

Hilos

- **Modelo clásico de hilos.** (*cont.*)
- Comparación entre múltiples procesos independientes y múltiples hilos independientes:



(a) Tres procesos, cada uno con un hilo. (b) Un proceso con tres hilos.

49

Hilos

- **Hilos en POSIX.**
- El IEEE ha definido un estándar para los hilos: 1003.1c. El paquete de hilos se conoce como **Pthreads**. La mayoría de los sistemas compatibles con POSIX aceptan éste paquete.

Llamada de hilo	Descripción
Pthread_create	Crea un nuevo hilo
Pthread_exit	Termina el hilo llamador
Pthread_join	Espera a que un hilo específico termine
Pthread_yield	Libera la CPU para dejar que otro hilo se ejecute
Pthread_attr_init	Crea e inicializa la estructura de atributos de un hilo
Pthread_attr_destroy	Elimina la estructura de atributos de un hilo

Algunas de las llamadas a funciones de Pthreads.

50

Hilos

- **Hilos en el espacio de usuario.**
- Existen dos categorías de implementación de hilos en SO: hilos en el espacio de usuario (user-level threads, ULT) e hilos en el espacio de núcleo (kernel-level threads, KLT).
- En el enfoque de hilos en espacio de usuario, la aplicación gestiona todo el trabajo de los hilos y el núcleo del SO no es consciente de la existencia de los mismos. Cualquier aplicación puede programarse para ser multihilo a través del uso de una biblioteca de hilos, que es un paquete de rutinas para la gestión de ULT.
- Por defecto, una aplicación comienza con un hilo, los que se localizan en un solo proceso gestionado por el núcleo. La creación, destrucción y planificación de los hilos se realiza llamando a funciones de la biblioteca de hilos.

51

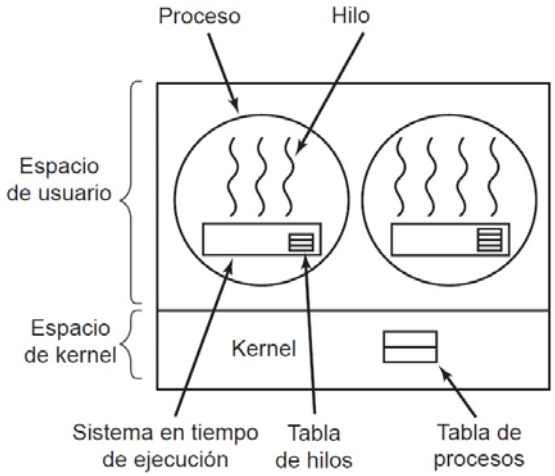
Hilos

- **Hilos en el espacio de usuario.** (*cont.*)
- Las ventajas de utilizar ULT son:
  - El cambio de hilo no requiere privilegios de modo núcleo, ya que todas las estructuras de datos se encuentran en el espacio de usuario, lo que ahorra la sobrecarga de los cambios de modo.
  - La planificación la puede realizar la aplicación, eligiendo el algoritmo más conveniente para cada aplicación.
  - Los ULT pueden ejecutar en cualquier SO; no se necesitan cambios en el núcleo para dar soporte a los ULT.
- Desventajas de utilizar ULT:
  - En un SO típico, muchas llamadas al sistema son bloqueantes, por lo que, si un hilo realiza una llamada al sistema, no sólo se bloquea el hilo sino todos los hilos del proceso.
  - No se puede sacar provecho de sistemas multiprocesadores, ya que el núcleo asigna el proceso a un solo procesador al mismo tiempo. Esto implica que, en determinado momento, sólo puede ejecutar un hilo del proceso, aunque sí se puede aplicar multiprogramación.

52

Hilos

- **Hilos en el espacio de usuario.** (*cont.*)
- Modelo de hilos en espacio de usuario:



53

Hilos

- **Hilos en el espacio de núcleo.**
- En este enfoque, la gestión de los hilos la realiza el núcleo del SO. No hay código de gestión en la aplicación, sólo una interfaz (API) para acceder a las utilidades de hilos del núcleo. Además, el núcleo mantiene información del contexto del proceso como una entidad y de los hilos individuales del proceso.
- Este enfoque resuelve los dos principales problemas del ULT:
  - Por un lado, el núcleo puede planificar simultáneamente múltiples hilos de un solo proceso en múltiples procesadores.
  - Por otro lado, si se bloquea un hilo, el núcleo puede planificar otro hilo del mismo proceso. Además, las rutinas del núcleo pueden ser en sí mismas multihilo.

54

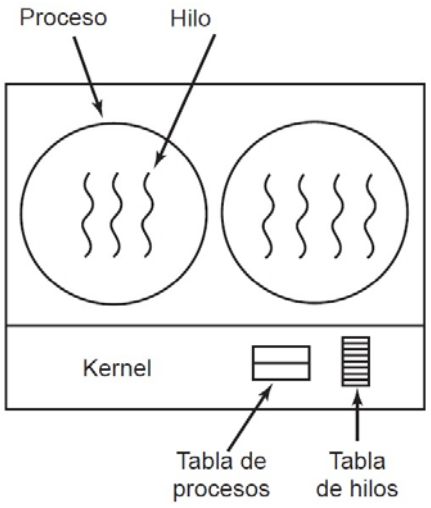
Hilos

- **Hilos en el espacio de núcleo.** (*cont.*)
- La principal desventaja del enfoque KLT es que la transferencia de control de un hilo a otro requiere de un cambio de modo al núcleo, ya que se crean y gestionan en su espacio de direcciones.
- Los SO multihilos presentan mayor performance global; sin embargo, el máximo beneficio se obtiene cuando las aplicaciones de usuario también son programadas multihilos.

55

Hilos

- **Hilos en el espacio de núcleo.** (*cont.*)
- Modelo de hilos en espacio de núcleo:



56

Fin del Módulo II

57

Sistemas Operativos I

Módulo III

PLANIFICACIÓN DE LA CPU

1

Introducción

Sistemas multitarea

Característica principal: múltiples procesos de usuarios y del sistema operativo comparten dinámicamente los recursos físicos y lógicos del sistema.

Recursos del sistema: procesador, memoria, archivos, dispositivos de entrada-salida, etc.

Módulos siguientes de Sistemas Operativos I

Se estudiarán temas relacionados con la administración de los recursos más importantes del sistema:

Módulo III: Planificación del procesador.

Módulo IV: Administración de la memoria.

Módulo V: Administración de archivos y dispositivos de E/S.

2

Introducción

Objetivo de los SO multitarea: maximizar la utilización de los recursos del sistema por parte de los procesos.

Utilización del Procesador: se deben planificar los procesos para optimizar su uso.

¿Cuál es la importancia de planificar el uso del procesador?

Planificación del procesador:

- Maximizar su aprovechamiento tiene gran incidencia en el rendimiento global del sistema: p. ej., se debe alcanzar un alto throughput de procesos.
- Throughput: número de procesos ejecutados totalmente por unidad de tiempo. Este es uno de los parámetros indicadores de la eficiencia del sistema en la computación de datos.

3

Introducción

Planificación del procesador *(cont.)*

Planificando eficientemente los procesos que están en la cola Ready ¿es suficiente para obtener un alto throughput?

Es **sólo una** de las tareas para alcanzar el objetivo anterior. Hay otras que también son importantes.

4

Introducción

Consideraciones generales de la planificación de procesos

- Durante su vida en el sistema, un proceso transiciona por distintos estados, existiendo colas asociadas a cada estado.
- Una de las tareas del núcleo es la de seleccionar y mover los procesos entre colas, para lo cual utiliza diferentes algoritmos que inciden en el rendimiento del sistema.
- Punto de vista general: seleccionar y mover procesos entre colas o ponerlos a ejecutar se denomina **planificación o "scheduling"**.
- Dentro del SO existen varios programas planificadores o schedulers. Uno de ellos (el de mayor importancia) es el **planificador de la CPU** o **"dispatcher"**.
- Otro planificador importante puede ser el planificador de disco.

5

Introducción

Planificación de procesos para uso de la CPU

- Los algoritmos usados para planificar la CPU son diferentes y de mayor complejidad que los algoritmos que planifican la utilización de otros dispositivos de E/S.
- Esta diferencia es por la incidencia que tienen en la performance global del sistema y por ser la CPU el recurso más escaso y más rápido del mismo.

6

Introducción

Parámetros indicadores de la eficiencia del sistema

- **Tiempo de respuesta:** Tiempo de primera salida de datos desde que comenzó a ejecutar un programa.
- **Throughput de procesos:**
  (número de procesos terminados) / (unidad de tiempo)
- **Eficiencia del procesador:**
  (tiempo útil) / (tiempo total de uso)
  - Tiempo útil: tiempo neto de cómputo (ejecución de programas de usuario).
  - Tiempo total de uso: tiempo útil + tiempo de no cómputo.
  - Tiempo de no cómputo: tiempo usado en cambios de procesos, procesamiento de IRQ, compactación de memoria, de disco, etc.

7

Tipos de Planificadores

Planificadores existentes en el sistema

- El objetivo de los planificadores de CPU es asignar procesos al procesador para optimizar el tiempo de respuesta, el throughput y el uso eficiente del procesador.
- Para ello, se requiere de la tarea conjunta de **3 planificadores diferentes:**
- Planificador de largo plazo o long-term scheduler.
- Planificador de mediano plazo o medium-term scheduler.
- Planificador de corto plazo o short-term scheduler.

8

Actuación de los Planificadores

Los distintos planificadores actúan sobre colas diferentes.

Long Term:
Actúa cuando se crea un proceso nuevo. (No todos los SO lo tienen). Algunos SO no limitan a priori el ingreso de procesos al sistema, sino que la limitación se da cuando rebalsan las tablas internas (de procesos, de memoria, etc.) o por razones de performance.

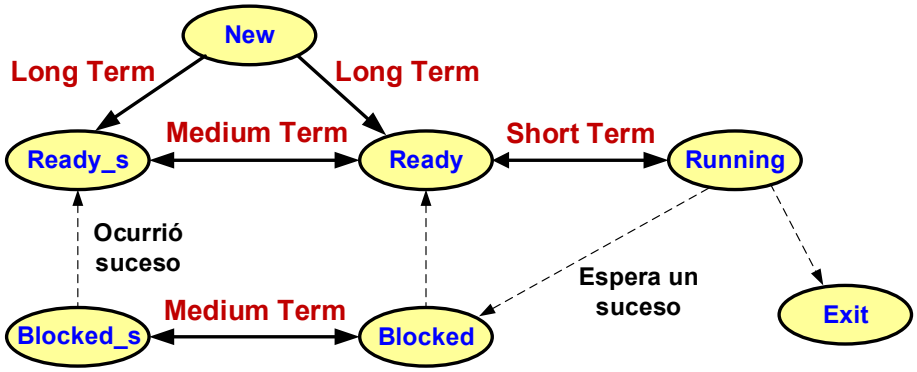
Medium Term:
Participa en la operación de swapping (en la cola Ready o en Blocked). Analiza la disponibilidad de memoria principal en relación a las prioridades de los procesos.

Short Term:
Realiza la selección de un nuevo proceso para usar la CPU (actúa sobre procesos en la cola de Ready).

9

Tipos de Planificadores

Planificadores en un diagrama de estados de procesos



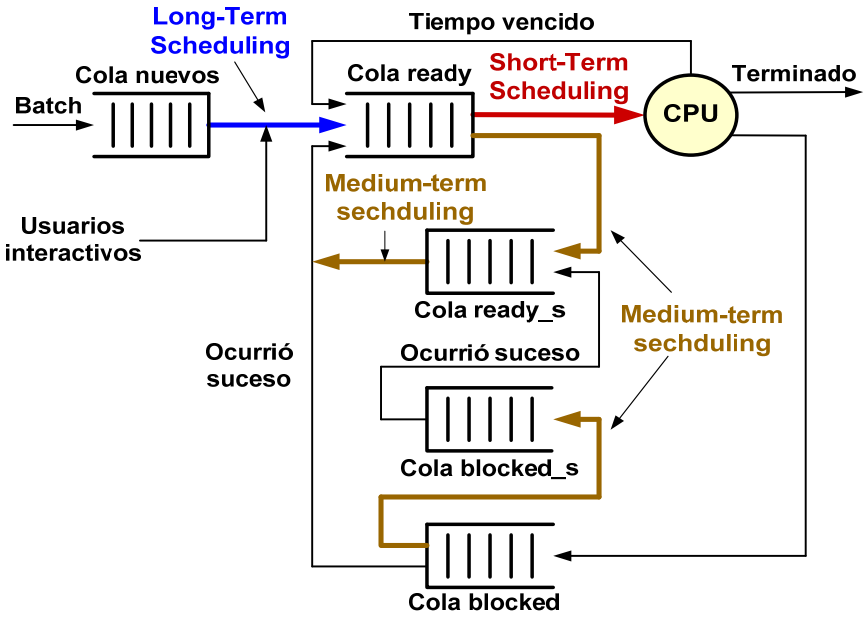
10

Actuación de los Planificadores

- Los planificadores actúan sobre colas diferentes asociadas a los estados de procesos, con particulares restricciones de tiempo por lo que pueden usar distintas directivas (algoritmos) para manipular los procesos.
- En el siguiente diagrama se representan estas colas y lo eventos que disparan las transiciones.

11

Diagrama de Colas y Planificadores



12

Long-Term Scheduler

- Es el encargado de controlar el grado de multiprogramación que admite el SO a fin de evitar congestión y bajo rendimiento.
- Actúa en la admisión de los procesos nuevos a la cola Ready.
- **Sistemas que admiten trabajos por lotes (batch):** existe un planificador de largo plazo. Las tareas nuevas que ingresan al sistema son almacenadas en disco y mantenidos en cola de procesos batch. El planificador llevará procesos de esta cola a lo cola Ready, si el SO los admite y de acuerdo a prioridades.
- **Sistemas de tiempo compartido:** cuando se crea un nuevo proceso interactivo, el SO **no lo pone en una cola**, sino que directamente son admitidos en el sistema hasta que éste llega a un estado de saturación. En cuyo caso se rechaza el proceso y se notifica al usuario de tal situación, invitándolo a intentar en otra ocasión.
- Normalmente, Long-Term Scheduler utiliza el algoritmo FIFO.

13

Medium-Term Scheduler

Planificación de mediano plazo: forma parte de la operación de swapping en los computadores.

- Decisión de desalojar un proceso de memoria principal. Depende de:
  1. Por un lado, tratar de mantener un **grado máximo de multiprogramación**.
  2. Por otro lado, el grado de multiprogramación está limitado por el espacio disponible en memoria.
- Tarea del Medium-Term Scheduler: debe conciliar lo mejor posible entre 1 y 2 (Memoria virtual).

14

Short-Term Scheduler

También es llamado **dispatcher**, o directamente **scheduler**.

Es el encargado de sacar los procesos en estado de Ready y ponerlos en estado de Running.

- **El dispatcher planifica (elige un proceso) cuando:**

1. Un proceso abandona la CPU porque:
   - Ha terminado su ejecución.
   - Pasa de Running a Blocked.
   - Pasa de Running a Ready.
2. Un proceso pasa de Blocked a Ready.

15

Short-Term Scheduler

¿Cuándo entra en acción el dispatcher?

- En general, entra en acción cada vez que ocurre una interrupción del proceso que está en uso de la CPU. (IRQ del timer, system call, trap, etc.).
- El hecho de que el scheduler entre en acción no quiere decir que vaya a planificar (elegir un proceso):

  Primero analiza si corresponde planificar. Si corresponde, planifica; si no, devuelve el procesador al proceso anterior.

  Esta es otra forma de diferenciar entre cambio de proceso y cambio de modo.

  ¿Cuándo decide planificar? Cuando el proceso sí o sí será desalojado de la CPU (cambio de proceso).

16

Tipos de Algoritmos de Planificación de la CPU

Los algoritmos pueden ser:

No apropiativos o no expulsivos (non-preemptives).

Apropiativos o expulsivos (preemptives).

- Algoritmos **no apropiativos:** cuando el proceso abandona la CPU sólo en forma voluntaria (cuando finaliza o se bloquea).

  Ejemplo: MS Windows v3.x.

- Algoritmos **apropiativos:** cuando el uso de la CPU (por parte de un proceso) está siendo controlado por el núcleo.
- Motivos de retiro de un proceso del procesador:
  - Expiración de la cuota (quantum) de tiempo de uso del CPU.
  - Ingreso a la cola Ready de un proceso de mayor prioridad.

  Ejemplo de SO: UNIX, MS Windows 20xx/Win10, IBM OS/2.

17

Tipos de Algoritmos de Planificación de la CPU

- Algoritmos **apropiativos** (*cont.*)**:** significan una mayor sobrecarga al sistema debido al gran número de cambios de procesos que generan respecto de los no apropiativos.

  Sin embargo, los apropiativos proveen un mejor servicio global.

  SO que usan estos algoritmos: UNIX, MS Windows 20xx/Win10, IBM OS/2.

18

Algoritmos de Planificación de la CPU

Algoritmos de planificación

- **First-Come, First-Served (FCFS):** es el más simple. Es no apropiativo por definición. No se utiliza en sistemas de tiempo compartido. El proceso elegido ejecuta hasta que finaliza.
- **Shortest-Job-First (SJF):** otorga la CPU al proceso que la utilizará por menor tiempo en el futuro. En caso de haber más de un proceso en estas condiciones, se usa FCFS. Puede implementarse en forma apropiativa, con una versión no apropiativa, llamada **Shortest Remaining Time (SRT)**.

  Desventaja: es complicado determinar a futuro la duración de una tarea.

19

Algoritmos de Planificación de la CPU

Algoritmos de planificación *(cont.)*

- **Prioritario:** cada proceso tiene asociada una prioridad. La CPU se otorga al proceso que en la cola tenga la mayor prioridad. ¿Varios procesos de igual prioridad? Se usa FCFS.

  Para poder usar este algoritmo se debe asignar una prioridad a los procesos.
- Prioridad: es un número que debe calcularse y, normalmente, es función de diversos enfoques que son representados por parámetros. Por ejemplo:
  a) Desde el punto de vista interno o externo al sistema; o una combinación de ambos.
  b) La prioridad puede ser estática o dinámica en el tiempo.

20

Algoritmos de Planificación de la CPU

Algoritmos de planificación *(cont.)*

- **Prioritario** *(cont.)*

  Definición interna de la prioridad: depende de los requerimientos de computación del proceso en sí: tiene en cuenta los recursos de hardware y software que utiliza el proceso en cuestión (tiempo que utilizó la CPU, tiempo total que lleva en el sistema, memoria utilizada, número de archivos abiertos, etc.).

  Definición externa de la prioridad: requerimientos independientes del SO: prioridades de determinados usuarios; procesos del sistema vs. procesos de usuarios.

  Las prioridades interna y externa se combinan para calcular una única prioridad, la que será usada a los fines de la planificación.

21

Algoritmos de Planificación de la CPU

Algoritmos de planificación *(cont.)*

- **Prioritario** *(cont.)*
  - Además, puede ser apropiativo o no apropiativo.
  - Puede presentar un grave problema: inanición. Una solución posible es aumentar la prioridad cada determinado tiempo para que pueda ejecutar; así, hasta finalizar.
- **Round Robin (RR):** fue especialmente diseñado para sistemas de tiempo compartido. Se define una pequeña unidad de tiempo denominada quantum o time slice (entre 10ms y 1s). La cola Ready se implementa como una "cola circular" donde el scheduler otorga un quantum a cada proceso en dicha cola. De esta forma todos los procesos tienen garantizado el uso de la CPU. Este algoritmo es apropiativo por naturaleza.

22

Algoritmos de Planificación de la CPU

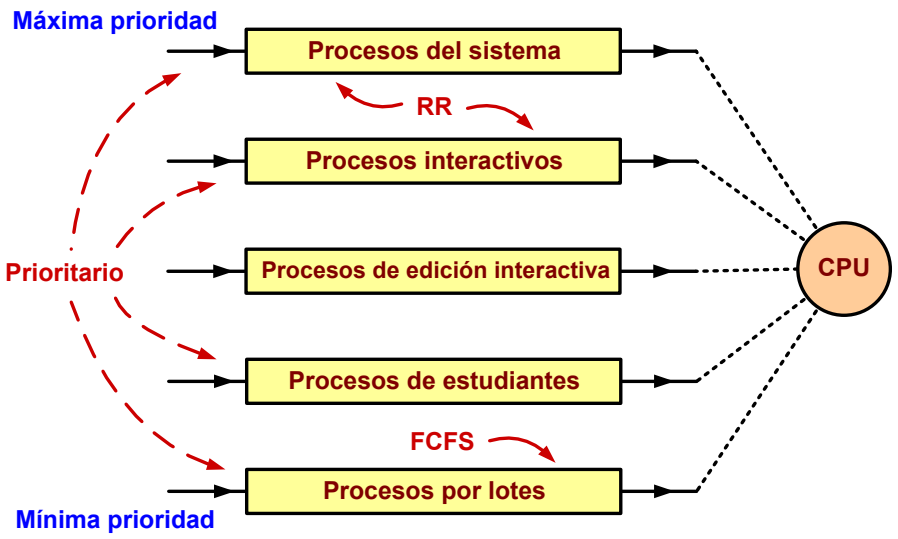
Algoritmos de planificación *(cont.)*

- **Colas Multinivel:**
- Hasta ahora tratamos sistemas con una única cola Ready.
  - Sin embargo, esta no es la solución más apropiada puesto que en los sistemas existen diferentes tipos de procesos (batch, interactivos, de usuario, del sistema) y cada tipo de proceso requiere un tratamiento particular; por lo tanto, debe ser planificado de forma diferente.
  - Una solución adecuada a este requerimiento es contar con varias colas Ready.

Ejemplo: En la figura siguiente se muestra un esquema de planificación con varias colas Ready de acuerdo al tipo de proceso que ingresa al sistema.

23

Diagrama de Colas Ready Multinivel



24

Colas Ready Multinivel

Planificación basada en colas multinivel

Consiste de varias colas sobre las que operan, de manera organizada, un conjunto de algoritmos.

Ejemplo1: un modo de funcionamiento de la planificación en el esquema de la figura anterior podría ser:

1. Un algoritmo ubica los procesos en la cola correspondiente. (p. ej. : los ubica de acuerdo a la prioridad asignada a cada proceso).
2. A su vez, cada cola posee su propio algoritmo de planificación. (p. ej.: la cola con procesos interactivos utiliza RR, mientras que la de procesos por lotes usa FCFS).
3. Los procesos de la cola con una dada prioridad podrán hacer uso de la CPU **siempre que no haya un proceso en una cola de mayor prioridad.**

25

Colas Ready Multinivel

Planificación basada en colas multinivel *(cont.)*

Ejemplo1: *(cont.)*

4. El algoritmo que planifica el proceso que va a hacer uso de la CPU puede ser preemptive:

   Si está haciendo uso del procesador un proceso de una cola con una dada prioridad y llega otro proceso a una cola de mayor prioridad, el primer proceso es retirado del procesador y puesto el proceso recién llegado.

Ventaja del esquema basado en colas multinivel sobre SJF:

Es muy difícil implementar el SJF puro debido a que es imposible, en muchos casos, conocer a priori el tiempo que llevará la ejecución de un proceso.

26

Colas Ready Multinivel con Realimentación

Un esquema basado en colas de Ready multinivel que mejora aún más la performance, es el denominado Colas Multinivel con Realimentación:

Colas Multinivel con Realimentación:

1. Está basado en el esquema de colas multinivel (varias colas Ready sobre las que actúan organizadamente un conjunto de algoritmos).
2. Además, utiliza un **mecanismo dinámico de prioridades** que se provee mediante un mecanismo de realimentación. (Se describe a continuación).

27

Colas Ready Multinivel con Realimentación

Ejemplo2: uso de mecanismo dinámico de prioridades.

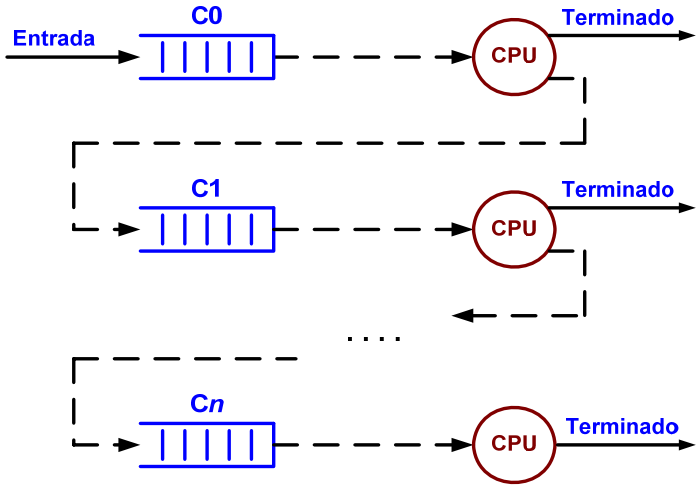
- Cuando un proceso ingresa por primera vez al sistema es puesto en la cola de prioridad C0 (máxima).
- Después de su primera ejecución, si no ha finalizado, el proceso vuelve al estado Ready y es puesto en la cola de prioridad C1 (inferior a C0).
- En resumen: después de cada ejecución, si no ha finalizado, el proceso es puesto en una cola de prioridad inferior a la que estuvo anteriormente.

Por ejemplo, en cada cola puede usarse el algoritmo RR que otorga un quantum mayor a medida que la cola tiene menor prioridad, hasta llegar a la última cola en la que se usa FCFS.

28

Ejemplo de Planificación con Realimentación

Diagrama dinámico en un sistema monoprocesador



El esquema muestra las distintas colas por las que puede pasar un proceso que se ingresa inicialmente en **C0** y necesita, a lo largo de su ejecución, usar la CPU varias veces antes de finalizar.

29

Colas Ready Multinivel con Realimentación

- Este esquema de planificación así concebido, favorece los procesos **nuevos** y **cortos** porque finalizarán rápidamente antes que los viejos y largos. (es una implementación similar a SJF).

  La mayoría de los SO actuales utiliza un esquema similar al presentado, pero cada uno con su propia variante.

  El ejemplo presentado tiene un problema: los procesos largos pueden llegar a padecer inanición.

  Los SO lo salvan implementando un mecanismo que, por ejemplo, cada tanto se indaga en la última cola si hay procesos que no han hecho uso del procesador por un tiempo dado. Si hay, los promocionan a colas de mayor prioridad, aumentándoles, incluso, el quantum.

30

Comparación de Métodos de Planificación

- **Criterios orientados al usuario:**
  Percepción del usuario respecto del comportamiento del sistema.
  Ejemplo: el objetivo del planificador de CPU en un sistema interactivo puede ser **maximizar** el número de procesos cuyo tiempo de respuesta no sea mayor que 2s.
- **Criterios orientados al sistema:**
  El más importante: uso eficiente del procesador.
  Ejemplo: maximizar el throughput de procesos terminados.
  (número de procesos terminados/unidad de tiempo).

31

Parámetros de Evaluación de Algoritmos

- **Utilización del procesador.** (orientado al sistema)
  Lo ideal es mantener la CPU ocupada la mayor cant. de tiempo
- **"Throughput".** (orientado al sistema)
  Mayor cantidad de procesos finalizados por unidad de tiempo.
- **Tiempo de "Turnaround" (retorno).** (orientado al usuario)
  Intervalo de tiempo desde que un proceso ingresa al sistema hasta que lo abandona. Se aplica especialmente a los procesos batch.
  (Suma de T de uso de: CPU + dispos. E/S + espera en colas).
- **Tiempo de Espera.** (orientado al usuario)
  Tiempo de espera del proceso en la cola Ready.
- **Tiempo de Respuesta.** (orientado al usuario)
  Es similar al Tiempo de "Turnaround" pero referido a procesos interactivos: intervalo de tiempo desde que el proceso ingresa al sistema hasta que produce la primera salida de datos.

32

Comportamiento de Algoritmos

Evaluación de algoritmos planificadores según parámetros

- **FCFS:**
  Tiempo de "Turnaround" y Espera. Tiempo de Espera: puede ser bastante alta, puesto que el proceso deberá permanecer en la cola Ready todo el tiempo que demande la ejecución de los procesos que están antes que él.
- **SJF:**
  Tiempo de espera promedio: SJF tiene un comportamiento óptimo porque privilegia los procesos cortos.
- **Prioritario:**
  No se puede decir nada a priori, pues depende de la función utilizada en calcular la prioridad de los procesos.

33

Comportamiento de Algoritmos

Evaluación de algoritmos planificadores según parámetros (*cont.*)

- **Round Robin:**
  Performance íntimamente vinculada con el quantum de tiempo.
  - Si el quantum es muy grande: RR→FCFS.
  - Si es muy pequeño: baja la performance global del sistema por la gran cantidad de cambios de procesos.
- **Colas Multinivel:**
  La evaluación de performance del conjunto de algoritmos usado es bastante compleja de hacer pero, en general, presenta un comportamiento mucho mejor que los algoritmos anteriores.

34

Comportamiento de Algoritmos

	FCFS	SJF	Prioridad	Round Robin	Cola Multinivel
Modo de seleccionar el proceso	Por orden de llegada a la cola de Ready.	Mínimo tiempo para procesar.	Máxima prioridad.	Constante.	Depende de los algoritmos usados.
Modo de decidir el uso de la CPU	Nonpreemptive	Nonpreemptive y preemptive.	Nonpreemptive y preemptive.	Preemptive (cumplido el quantum).	Preemptive.
Throughput (productividad)	No relevante.	Alto.	No relevante. Depende de cómo se asigne	Puede ser baja si el quantum es pequeño.	No relevante.
Tiempo de respuesta (tiempo de 1ª salida de datos)	Puede ser alto. Especialmente si hay procesos CPU bound.	Bueno para procesos cortos.	Bueno para procesos de alta prioridad.	Bueno para procesos cortos.	No relevante.
Overhead (sobrecarga)	Mínimo.	Puede ser alto, sobre todo si es preemptive.	Puede ser alto, sobre todo si es preemptive.	Puede ser alto si el quantum es pequeño.	Puede ser alto.
Efecto en los procesos	Penaliza los procesos cortos y los I/O bound.	Penaliza los procesos largos.	Penaliza los procesos de baja prioridad.	Tratamiento igualitario.	Puede favorecer algunos procesos (según tipo de usuario).

35

Fin del Módulo III

36