Sistemas Operativos I

Módulo Introductorio

CONCEPTOS BÁSICOS NECESARIOS

1

Temas del Módulo Introductorio

- **Definiciones y convenciones de notación.**
  - Unidades de cantidad de información.
  - Tiempo y Velocidad de transmisión de datos.
  - Frecuencia y Período.
- **Fundamentos de hardware de una computadora.**
  - CPU.
  - Memoria principal.
  - Subsistema de Entrada/Salida.
  - Buses.
- **Jerarquía de memoria.**
  - Diseño y componentes.
  - Características: tiempo de acceso y capacidad de almacenamiento.
  - Principio de cercanía de referencias.
- **Comunicación entre componentes de una computadora.**
  - Entrada/Salida programada. Polling.
  - Interrupciones.
  - DMA (Direct Memory Access).

2

Definiciones y convenciones de notación.

Unidades de cantidad de información:

Bit (b): Binary Digit. Es la unidad elemental de información.

Byte (B): conjunto de bits, dependiendo de la arquitectura, pueden ser de 4 a 12 bits.
Actualmente se lo considera de 8 bits. También llamado *octeto* para evitar ambigüedades.

https://en.wikipedia.org/wiki/Byte
https://es.wikipedia.org/wiki/Byte

Múltiplos de las unidades de información: (IEC 80000-13)

Kilobit (Kb):	**1.000 b - 10^3 b**	**Kilobyte (KB):**	**1000 B - 10^3 B**
Megabit (Mb):	**1.000.000 b - 10^6 b**	**Megabyte (MB):**	**10^6 B**
Gigabit (Gb):	**10^9 b**	**Gigabyte (GB):**	**10^9 B**
Terabit (Tb):	**10^{12} b**	**Terabyte (TB):**	**10^{12} B**

3

Definiciones y convenciones de notación.

Unidades de cantidad de información: (*cont.*)

Múltiplos de las unidades de información: (IEC 80000-13)

Kibibit (Kib):	**1024 b - 2^{10} b**	**Kibibyte (KiB):**	**1024 B - 2^{10} B**
Mebibit (Mib):	**1024^2 b - 2^{20} b**	**Mebibyte (MiB):**	**1024 KiB - 2^{20} B**
Gibibit (Gib):	**1024^3 b - 2^{30} b**	**Gibibyte (GiB):**	**1024 MiB - 2^{30} B**
Tebibit (Tib):	**1024^4 b - 2^{40} b**	**Tebibyte (TiB):**	**1024 GiB - 2^{40} B**

4

Definiciones y convenciones de notación.

Unidades de cantidad de información: (*cont.*)

Palabra (wd) (*word*, en inglés): cadena de bits tratados en conjunto.

Normalmente, los conjuntos son grupos de Bytes, y según la arquitectura de hardware, en números pares y potencia de 2.

En las arquitecturas modernas, la palabra tiene 64 bits.

https://en.wikipedia.org/wiki/Word_(computer_architecture)
https://es.wikipedia.org/wiki/Palabra_(informática)

5

Definiciones y convenciones de notación.

Unidad de tiempo (t):

Segundo (s): duración de 9.192.631.770 oscilaciones de la radiación emitida en la transición entre los dos niveles hiperfinos del estado fundamental del isótopo 133 del átomo de cesio (133Cs), a una temperatura de 0°K.

Submúltiplos de la unidad de tiempo:

Milisegundo (ms): 10^{-3} s.

Microsegundo (μs): 10^{-6} s.

Nanosegundo (ns): 10^{-9} s.

Picosegundo (ps): 10^{-12} s.

6

Definiciones y convenciones de notación.

Velocidad de transmisión de datos:

Cantidad de unidades de información por unidad de tiempo.

Unidad de velocidad de transmisión de datos:

Bit por segundo (bps – b/s) ¿Y el B/s?

Múltiplos de la unidad de velocidad de transmisión:

Kilobit por segundo (Kbps): 10^3 bps - 1.000 bps

Megabit por segundo (Mbps): 10^6 bps - 1.000.000 bps

Gigabit por segundo (Gbps): 10^9 bps - 1.000.000.000 bps

Terabit por segundo (Tbps): 10^{-12} bps

7

Definiciones y convenciones de notación.

Frecuencia (f):

Magnitud que mide el número de repeticiones por unidad de tiempo de cualquier fenómeno o suceso periódico.

Unidad de frecuencia:

Hertz (Hz): ciclos por segundo.

Múltiplos de la unidad de frecuencia:

Kilohertz (KHz): 10^3 Hz.

Megahertz (MHz): 10^6 Hz.

Gigahertz (GHz): 10^9 Hz.

Terahertz (THz): 10^{12} Hz.

		Longitud de onda	Frecuencia	Energía
Radio	Muy Baja Frecuencia	> 10 km	< 30 Khz	< 1.99 e-29 J
	Onda Larga	< 10 km	> 30 Khz	> 1.99 e -29 J
	Onda media	< 650 m	> 650 Khz	> 4.31 e-28 J
	Onda corta	< 180 m	> 1.7 Mhz	> 1.13 e-27 J
	Muy alta frecuencia	< 10 m	> 30 Mhz	> 2.05 e-26 J
	Ultra alta frecuencia	< 1 m	> 300 Mhz	> 1.99 e-25 J
Microondas		< 30 cm	> 1.0 Ghz	> 1.99 e-24 J
Infrarrojo	Lejano / submilimétrico	< 1 mm	> 300 Ghz	> 199 e-24 J
	Medio	< 50 um	> 6.0 Thz	> 3.98 e-21 J
	Cercano	< 2.5 um	> 120 Thz	> 79.5 e-21 J
Luz Visible		< 780 nm	> 384 Thz	> 255 e-21 J
Ultravioleta	Cercano	< 380 nm	> 789 Thz	> 523 e-21 J
	Extremo	< 200 nm	> 1.5 Phz	> 993 e-21 J
Rayo X		< 10 nm	> 30.0 Phz	> 19.9 e-18 J
Rayos Gamma		< 10 pm	> 30.0 Ehz	> 19.9 e-15 J

8

Definiciones y convenciones de notación.

Período (T): duración de un ciclo de reloj.

$$T = \frac{1}{f}$$

Equivalencia entre frecuencia y velocidad:

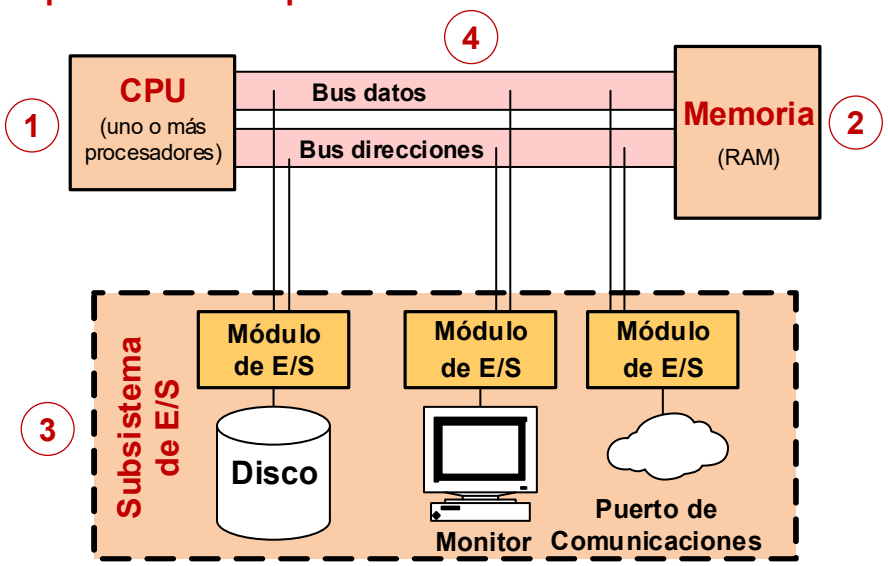
$$1Hz \equiv 1bps$$

$$1KHz \equiv 1Kbps$$

9

Fundamentos de HW de una computadora.

Componentes Principales:



10

Fundamentos de HW de una computadora.

1. CPU (Unidad Central de Procesamiento).

Puede estar conformado por uno o más procesadores. Es el dispositivo que controla la operación del computador y procesa los datos.

Contiene registros internos que se utilizan para almacenar temporalmente instrucciones u operandos.

Estos registros se pueden clasificar en:

- **Registros Visibles al Usuario:** están disponibles para ser usados por los programadores para diversas tareas. Ej.: cargar datos o instrucciones para minimizar las referencias a memoria principal y, así, reducir tiempos de procesamiento.
- **Registros de Control y Registros de Estado:** No pueden ser accedidos por lo usuarios. Sólo pueden ser utilizados por la lógica interna del procesador (para control de las operaciones que realiza), o por subrutinas privilegiadas del SO para controlar la ejecución de los programas.

11

Fundamentos de HW de una computadora.

1. CPU (Unidad Central de Procesamiento). (*cont.*)

Registros Visibles al Usuario.

Se pueden acceder con instrucciones de alto y bajo nivel. Se dividen en:

- Registros de Datos: se usan para almacenar datos en general. Algunos registros pueden sólo usarse para datos especiales como, por ejemplo, para operaciones con coma flotante.
- Registros de Dirección: pueden ser de uso general o para un uso específico de direccionamiento. Para uso específico los más comunes son:
  - **Registro Índice (IR):** para direccionamiento indexado.
  - **Puntero de Segmento:** se usa cuando la memoria se divide en segmentos. Indica un desplazamiento de un segmento de memoria.
  - **Puntero de Pila:** apunta al comienzo de una pila.
- Registros de Código de Condición: El código de condición es un conjunto de bits indicadores, banderas, que indican alguna característica del resultado de una operación. Ej.: resultado positivo, negativo, cero, desbordamiento, etc. Normalmente estos registros de código de condición sí pueden ser leídos por el programador pero no pueden ser modificados.

12

Fundamentos de HW de una computadora.

1. CPU (Unidad Central de Procesamiento). (*cont.*)

Registros de Control y Registros de Estado.

Se emplean para controlar operaciones del procesador. No pueden ser accedidos por los usuarios y, en la mayoría de los casos, no son visibles. Sólo pueden ser accedidos por las subrutinas del SO. Ej.: en cambios de procesos. Los registros más típicos son:

- Program Counter (PC): Contiene la dirección de la próxima instrucción a ser referenciada y leída por el procesador.
- Instruction Register (IR): Contiene el código de la última instrucción leída.
- Program Status Word (PSW): Contiene información del estado actual del procesamiento. Los campos o bits más típicos son los indicadores de:
  - Signo.
  - Cero.
  - Acarreo.
  - Igualdad.
  - Overflow.
  - Habilitar/inhabilitar interrupciones, etc.

13

Fundamentos de HW de una computadora.

2. Memoria Principal.

Almacena datos y programas. Es normalmente volátil. También se la denomina memoria principal, memoria física, memoria real, RAM (memoria de acceso aleatorio).

La memoria es el dispositivo donde se carga parte o todo de:

- Sistema Operativo (en adelante, SO).
- Programas de usuarios.
- Datos que va a utilizar el procesador en la ejecución de un programa.

14

Fundamentos de HW de una computadora.

3. Subsistema de Entrada/Salida[1].

Se divide en dos subsistemas:

- Subsistema de almacenamiento[2].
- Subsistema de comunicaciones.

Ambos subsistemas están compuestos por un conjunto de:

- Módulos de E/S o Controladores.
- Dispositivos periféricos.

Módulo de E/S:

- Transporta datos entre la memoria del computador y los dispositivos periféricos.

Dispositivos periféricos:

- Discos.
- Equipos de comunicación.
- Monitor, teclado, etc.

[1] En adelante, E/S.
[2] También llamado "de almacenamiento secundario".

15

Fundamentos de HW de una computadora.

3. Subsistema de E/S. (*cont.*)

Cada módulo de E/S realiza las siguientes tareas:

- "Dialoga" e intercambia datos con el procesador y con la memoria a través del bus del sistema.
- Maneja y controla el dispositivo en el cual lee o escribe datos.

Cada uno de ellos contiene:

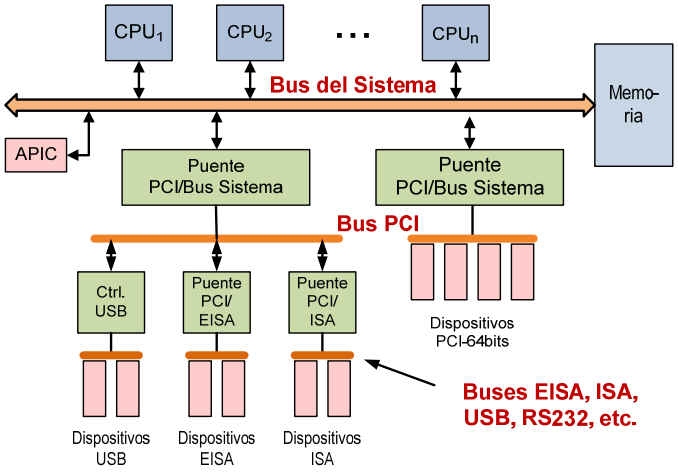
- Un procesador.
- Memoria ROM con código almacenado.
- Registro de estado.
- Buffer.
- Electrónica para el manejo del dispositivo que controla.

16

Fundamentos de HW de una computadora.

4. Buses.

Son la vía de conexión entre procesadores, memoria y módulos de E/S. También entre los módulos de E/S y los periféricos.



17

Jerarquía de memoria.

Diseño de la jerarquía de memoria.

- Al ser la memoria el dispositivo donde se almacena el SO, los programas y sus datos, lo ideal es contar con una gran cantidad, con alta velocidad de acceso y de bajo costo.
- La cantidad de memoria está relacionada con el espacio que ocupará el SO, las aplicaciones que van a ejecutar y sus datos.
- La velocidad de acceso a la memoria debe seguir el ritmo del procesador, para que éste no tenga que esperar la llegada de los datos almacenados en ella.
- El costo de la memoria debe ser razonable, en relación a otros componentes del computador.
- Sin embargo, la realidad es que la cantidad, la velocidad y costo **compiten entre sí**.

18

Jerarquía de memoria.

Diseño de la jerarquía de memoria. (*cont.*)

- Las tecnologías de fabricación, además, imponen ciertas limitaciones:
- A menor tiempo de acceso, mayor costo por bit.
- A mayor cantidad, menor costo por bit.
- A mayor cantidad, mayor tiempo de acceso.
- La salida del dilema anterior consiste en no depender de un único componente o de una única tecnología de memoria, sino emplear una **jerarquía de la memoria**.
- Esto significa, combinar distintos dispositivos de memoria, cada uno con características distintas de cantidad, velocidad y costo.

19

Jerarquía de memoria.

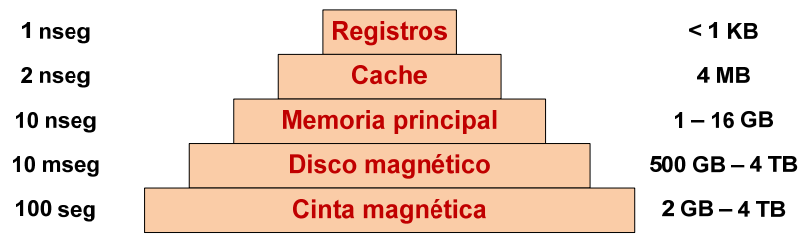
Componentes de la jerarquía de memoria.

- La cantidad total de memoria de una computadora es la suma de los tamaños de:
  - Registros el procesador.
  - Cache.
  - Memoria principal.
  - Almacenamiento secundario.
- La **jerarquía de memoria es la manera en que se organizan estos elementos**, según sus capacidades de almacenamiento, su velocidad de acceso y su costo relativo.
- El objetivo de esta organización es optimizar los accesos a memoria por parte del procesador, en cuanto minimizar los tiempos de acceso totales, lograr la máxima cantidad posible y al costo más razonable.

20

Jerarquía de memoria.

Características de los componentes.



- En esta figura, a medida que descendemos por la pirámide:
  - Disminuye el costo por bit.
  - Aumenta la capacidad.
  - Aumenta el tiempo de acceso.
  - Disminuye la frecuencia de acceso a la memoria.

21

Jerarquía de memoria.

Principio de cercanía de referencias.

La clave está en utilizar adecuadamente la disminución de la frecuencia de acceso a memoria.

Con una frecuencia de acceso baja, se puede obtener un buen rendimiento utilizando 2 o más niveles de memoria:

- Nivel 1: memoria de baja capacidad, alto costo por bit y rápida.
- Nivel 2: memoria de gran capacidad, bajo costo por bit y lenta.

Si la palabra a acceder está en el nivel 1, entonces es leída directamente por el procesador. A esto le llamaremos acierto.

Si la palabra no está en el nivel 1 y está en el nivel 2, entonces el procesador debe primero traer la palabra al nivel 1 y recién después puede accederla. A esto le llamaremos fallo. En este caso, el tiempo de acceso será la suma de ambos niveles.

22

Jerarquía de memoria.

Principio de cercanía de referencias. (*cont.*)

Para tratar de eliminar o amortiguar el efecto negativo de la lentitud es necesario disminuir la frecuencia de acceso a las memorias lentas. Para eso, es necesario que en cada acceso a una memoria lenta se traigan muchos datos a memorias más rápidas.

Entonces, surge una pregunta: ¿Qué datos deben traerse?

> **Principio de cercanía de referencias.**
> Durante la ejecución de un programa, las referencias a memoria (tanto para instrucciones como para datos) están generalmente cercanas. Esto sucede porque los programas contienen lazos y subrutinas iterativas que hacen que se produzcan referencias repetidas a un pequeño conjunto de instrucciones y datos que está cercanos entre sí.

23

Jerarquía de memoria.

Principio de cercanía de referencias. (*cont.*)

Para mejorar el rendimiento del procesador, es decir, disminuir el tiempo de ejecución conviene, por ejemplo, traer varias instrucciones y/o datos cada vez que se accede a memoria y almacenarlos en una memoria cache que es más rápida.

El principio de cercanía de referencias puede aplicarse en cualquier nivel de la jerarquía:

- De memoria principal a cache.
- De cache a registros del procesador.
- De disco a memoria principal (se usa parte de la memoria como cache de disco).

24

Comunicación entre componentes de una PC.

Hemos visto que el CPU es el dispositivo que controla las operaciones que se realizan en la computadora y procesa los datos almacenados.

Para ello, el CPU establece comunicaciones con la memoria y con los dispositivos de E/S a través de los Buses.

Debido a que la memoria principal es básicamente una sola, la velocidad de comunicación entre CPU y memoria es constante; no así con los dispositivos de E/S, que tienen un amplio rango de velocidades.

Como éstos últimos tienen tiempos de acceso varios órdenes de magnitud más grandes que los de memoria, se utilizan técnicas diferentes para la comunicación con el CPU, a fin de optimizar la ejecución de los programas.

Estas técnicas se diferencian según el grado de intervención del procesador en la operación de E/S.

25

Comunicación entre componentes de una PC.

E/S Programada.

La técnica se denomina **programada** porque las tareas son realizadas, en su mayoría, por rutinas de software ejecutadas por el CPU. Éstas rutinas son:

- **Rutina de E/S (*driver*):** programa del SO cuya función es llevar a cabo la operación de E/S. Hay una rutina de E/S específica para cada dispositivo de E/S.
- **Código del módulo de E/S:** está almacenado en la ROM del módulo y realiza el transporte de datos entre buffer y el dispositivo, controlando los circuitos electrónicos y elementos mecánicos (si hubiere).

El procesador interviene en todas las tareas de la operación de E/S de los datos entre memoria y dispositivo.

26

Comunicación entre componentes de una PC.

E/S Programada. (*cont.*)

Los tipos de operaciones de E/S en esta técnica son:

1. Transferencia de datos de memoria a dispositivo de E/S.
2. Transferencia de datos de dispositivo de E/S a memoria.

Por ejemplo, en **1**, éstas son las tareas que se realizan:

- Un programa que ejecuta en el procesador genera datos y llena con ellos un bloque de memoria.
- Se transfieren esos datos al buffer de un módulo de E/S, mediante un conjunto específico de instrucciones del procesador.
- El procesador envía los comandos específicos al módulo para realizar alguna tarea de procesamiento de los datos enviados.
- Mientras los datos son procesados por el módulo de E/S, el procesador se encuentra desocupado, esperando que finalice el procesamiento.
- Al finalizar el procesamiento, el módulo de E/S notifica al procesador.
- El procesador retoma la ejecución del programa para generar nuevos datos.

27

Comunicación entre componentes de una PC.

E/S Programada. (*cont.*)

En las tareas descriptas anteriormente, el procesador se mantiene desocupado mientras de el dispositivo de E/S realiza el procesamiento de los datos enviados desde memoria.

Esto implica que se detiene la ejecución del programa, por lo que se estaría perdiendo la oportunidad de, o bien ejecutar otro programa mientras el primero espera al dispositivo de E/S, o seguir con la ejecución del mismo programa realizando operaciones que no dependan de la E/S.

Para aprovechar el tiempo de procesador desocupado se plantea la técnica de **polling**.

28

Comunicación entre componentes de una PC.

Polling.

Consiste volver a utilizar el procesador para ejecutar un programa, ya sea el que estaba en ejecución u otro distinto, una vez que éste ha enviado la orden al dispositivo de E/S para que procese los datos enviados.

Luego, periódicamente, el procesador chequea el estado del dispositivo de E/S, a fin de verificar si han concluido las tareas de procesamiento que éste último debe realizar:

- Si aún no terminaron, retoma la ejecución del programa en curso.
- Si terminaron, invoca las siguientes rutinas de SO para concluir con la operación de E/S, y luego retoma la ejecución del programa en curso, o bien vuelve a ejecutar el programa que esperaba el resultado de la operación de E/S.

29

Comunicación entre componentes de una PC.

Polling. (*cont.*)

De esta manera, se aprovecha mejor el tiempo de uso de procesador, a pesar de que la rutina de polling para verificar el estado del dispositivo de E/S consuma parte de ese tiempo.

Observaciones:

Cuando se usa **polling** es muy importante determinar con qué frecuencia se debe realizar la consulta:

- Si la frecuencia es alta, hay ineficiencia del procesador.
- Si la frecuencia es muy baja, el dispositivo puede estar mucho tiempo desocupado hasta que el procesador detecte su estado.

Conclusión.

Este método es eficiente en el caso de dispositivos **lentos** (teclado), pero ineficiente en los dispositivos rápidos (discos).

30

Comunicación entre componentes de una PC.

Interrupciones.

El objetivo de esta técnica es permitir al procesador ejecutar un programa mientras se lleva a la cabo la operación de E/S.

Delega al controlador del dispositivo la responsabilidad de notificar la finalización de la operación de E/S. De esta manera, el procesador no realiza el polling y por lo tanto su participación en la operación es mucho menor.

Por esta razón, todas las plataformas de hardware de las computadoras actuales utilizan esta técnica.

Existen distintos **tipos de interrupciones**:

- De programa: se genera por un resultado específico obtenido luego de la ejecución de una instrucción.
- De reloj: un temporizador interno del CPU interrumpe la ejecución en intervalos regulares de tiempo.
- De operación de E/S: el módulo envía una interrupción (en adelante IRQ) al procesador para indicar algún evento específico.

31

Comunicación entre componentes de una PC.

Interrupciones. (*cont.*)

El funcionamiento de esta técnica es el siguiente:

- Cuando el programa en ejecución requiere de una operación de E/S, el procesador envía los comandos específicos al módulo controlador.
- Una vez enviados estos comandos, el procesador retoma la ejecución del programa o bien ejecuta otro, si es que la ejecución del primero depende de la operación de E/S.
- El módulo controlador del dispositivo realiza la operación que solicita el programa.
- Al finalizar la operación, envía al procesador una señal de interrupción (IRQ), para notificar esta situación.
- El procesador interrumpe la ejecución del programa en curso y ejecuta una rutina especial, llamada "**Interrupt Handler**" (IH), que se encarga de gestionar el pedido del dispositivo.
- Una vez finalizada la rutina de interrupción, se continúa con la ejecución del programa que corresponda.

32

Comunicación entre componentes de una PC.

Interrupciones. (*cont.*)

Las tareas que realiza el **Interrupt Handler** son las siguientes:

1. Guarda el estado de ejecución del programa que estaba en curso. Esto consiste en guardar los valores de los registros de control y de estado del procesador.
2. Chequea la prioridad, el origen y tipo de tarea requerida por la interrupción. En base a esto, evalúa si corresponde atender o no la interrupción:
   - Si IH decide **no atender** la interrupción, entonces IH restituye el uso del procesador al programa que venía ejecutando antes de producirse la interrupción (restituye en los registros del procesador la información que existía en el momento en que dejó de ejecutar el programa anterior para que pueda continuar su ejecución).
   - Si IH decide **atender** la interrupción, y la misma está relacionada con una operación de E/S, el IH llama a un módulo específico - Rutina de E/S - que realizará la tarea correspondiente.

33

Comunicación entre componentes de una PC.

Interrupciones. (*cont.*)

Observaciones:

- La ejecución del Interrupt Handler produce una sobrecarga a la tarea de procesamiento.
- No obstante, esta sobrecarga, en tiempo, es mucho menor que la operación de lectura/escritura de un dispositivo.
- De esta forma, si bien hay una sobrecarga en la tarea del procesador (overhead), **el uso de las interrupciones mejora la eficiencia global del procesamiento** y, por este motivo, las IRQ son usadas por todos los procesadores actuales.

34

Comunicación entre componentes de una PC.

Interrupciones. (*cont.*)

Interrupciones múltiples.

Cabe la posibilidad de que cuando un programa en ejecución se interrumpe, mientras se atiende la interrupción, llegue otra solicitud de interrupción. Y mientras ésta última se atiende, otra más puede llegar.

En ese caso, se deberá guardar el estado de ejecución de todos los programas que fueron interrumpidos. Esta situación se denomina **interrupciones múltiples**.

Se hace necesario, entonces, algún mecanismo que permita gestionar eficientemente estas sucesivas interrupciones, a fin de garantizar la correcta continuidad en la ejecución de los programas interrumpidos.

35

Comunicación entre componentes de una PC.

Interrupciones. (*cont.*)

Interrupciones múltiples. (*cont.*)

- Existen dos maneras básicas de gestionar las interrupciones múltiples:
  - Inhabilitando las interrupciones.
  - Asignando prioridades.
- Inhabilitación de interrupciones: cuando llega una interrupción, y el procesador la atiende, inmediatamente se inhabilitan las interrupciones posteriores. Esto implica que las que lleguen mientras se atiende una, quedan pendientes de atención, por lo que la atención de las mismas se realiza en forma secuencial, en el orden de llegada.
  - Ventaja: es una manera ordenada y simple de manejar las IRQ.
  - Desventaja: no tiene en cuenta las prioridades relativas de los programas, lo que puede llevar a situaciones críticas.

36

Comunicación entre componentes de una PC.

Interrupciones. (*cont.*)

Interrupciones múltiples. (*cont.*)

- Asignación de prioridades: de esta manera, cuando llega una interrupción de prioridad más alta que la que se está atendiendo, ésta interrumpe la atención de la de menos prioridad y pasa a ser atendida.
  - Ventaja: al tener en cuenta las prioridades relativas de los programas, es posible evitar situaciones críticas en la ejecución de programas de mayor prioridad.
  - Desventaja: la gestión de prioridades es más compleja que la anterior; sin embargo es la más eficiente.

37

Comunicación entre componentes de una PC.

DMA (Direct Memory Access).

Si bien en la técnica de interrupciones la participación del CPU en las operaciones de E/S es baja, sigue encargándose de la transferencia de datos.
Esto implica que la velocidad de transferencia de E/S está limitada por la velocidad con la que el procesador puede dar servicio a un dispositivo, lo cual representa una desventaja.
Por otro lado, resulta ineficiente:

- **En dispositivos rápidos.** Es el caso de una línea de comunicación de alta velocidad (Ej: LAN), la sobrecarga del procesador en el traspaso de estos datos puede significar una gran cantidad de ciclos del procesador, puesto que los servicios de éste son requeridos con alta frecuencia.
- **En alto volumen de datos a transferir.** En dispositivos medianamente rápidos y orientados a bloques de datos, como es el caso de un disco, los servicios del procesador serán requeridos varias o muchas veces en cada llenado del buffer del módulo de E/S, lo que también consume muchos ciclos del procesador.

38

Comunicación entre componentes de una PC.

DMA (Direct Memory Access). (*cont.*)

La alternativa para estos casos, entonces, es contar con un módulo de E/S con hardware específico para la transferencia de datos: el **módulo DMA**.

Con éste módulo, cuando se necesita transferir datos, el procesador emite una orden al módulo DMA enviándole información sobre:

- Tipo de operación a realizar.
- Dirección del dispositivo de E/S.
- Dirección inicial en memoria del bloque de datos a transferir.
- Cantidad de palabras a transferir.

Luego, el procesador se dedica a hacer otra tarea, y el módulo de DMA se encarga de toda la operación de E/S.

Cuando la transferencia de datos se completa, envía una interrupción al procesador.

39

Comunicación entre componentes de una PC.

DMA (Direct Memory Access). (*cont.*)

El módulo de DMA debe tomar el control del bus para transferir los datos a la memoria, por lo que puede existir competencia con el procesador por el uso del bus. El procesador debe esperar a que el bus se desocupe.

El efecto global que produce una transferencia de DMA es una ejecución del procesador más lenta que la normal durante la misma; puesto que, si bien, el procesador está más libre para procesar otros programas, debe disputar el bus con el módulo DMA, por ejemplo, cuando necesita traer un dato de memoria.

No obstante, en una transferencia de E/S con alto volumen de datos, **DMA es bastante más eficiente** que la E/S con interrupciones.

40

Fin del Módulo Introductorio

41

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

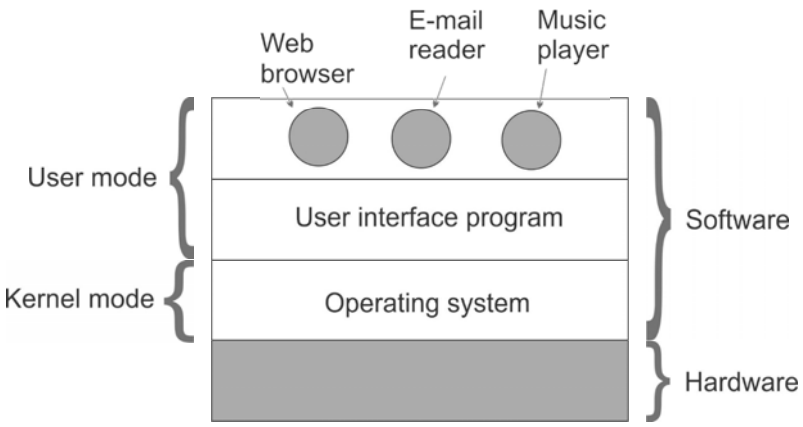
Definición de Sistema Operativo

- **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

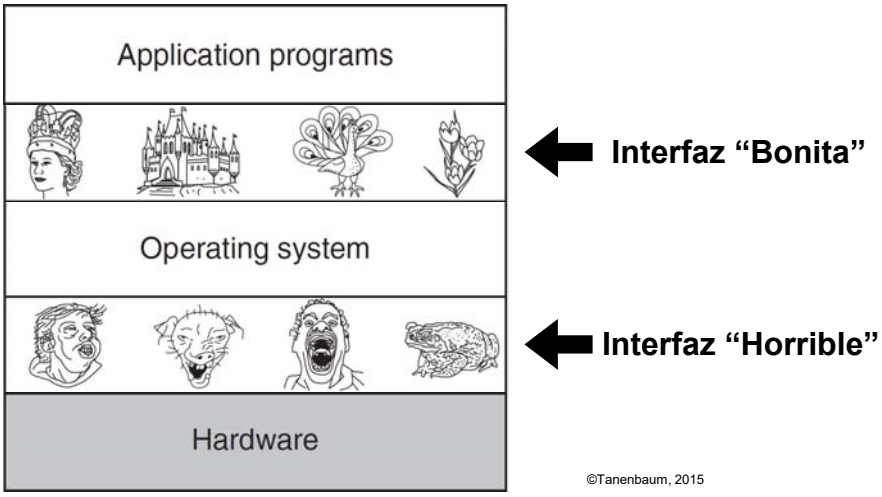
Definición de Sistema Operativo

- **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

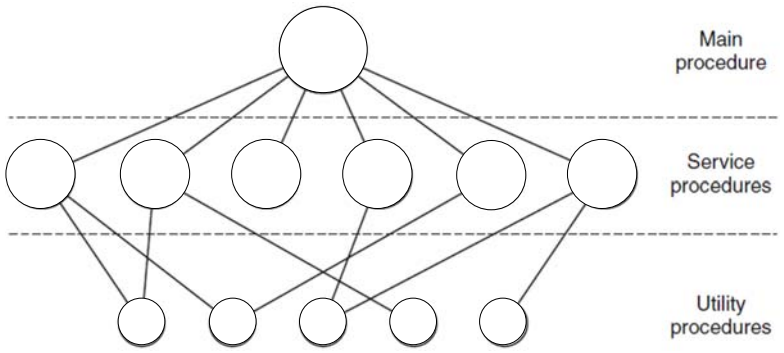
Estructura del Sistema Operativo

- **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

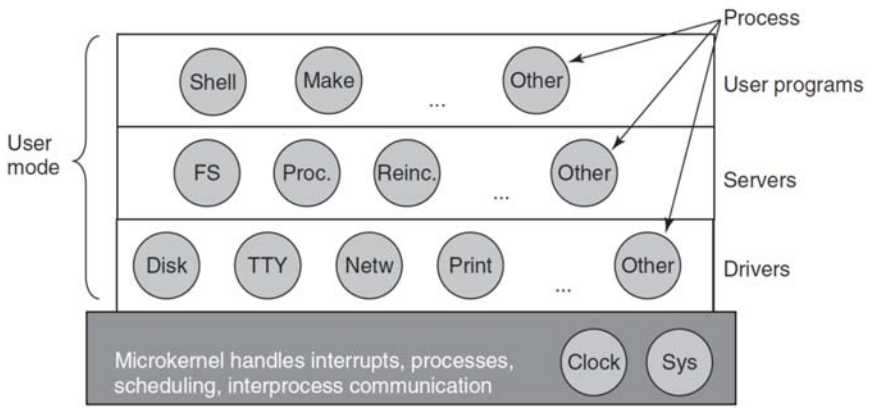
Estructura del Sistema Operativo

- **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

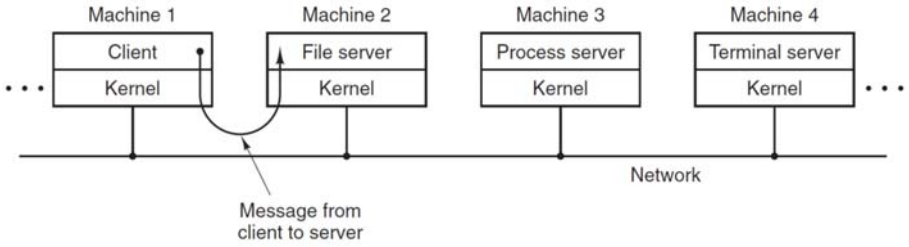
Estructura del Sistema Operativo

- **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

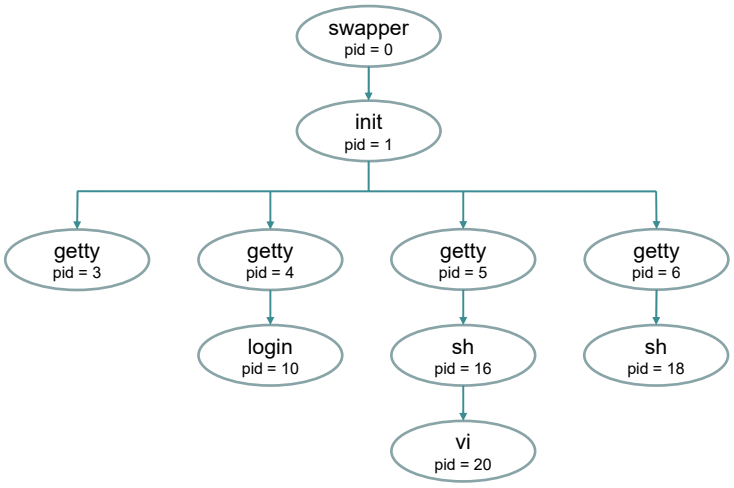
Procesos

- **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

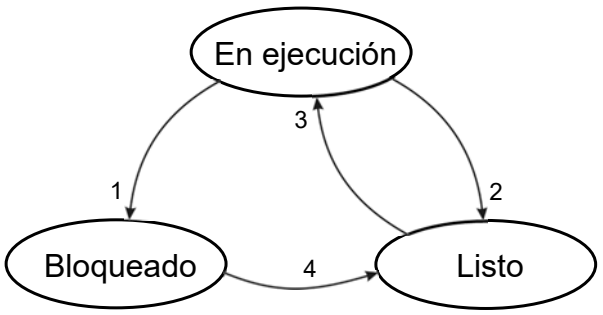
Procesos

- **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

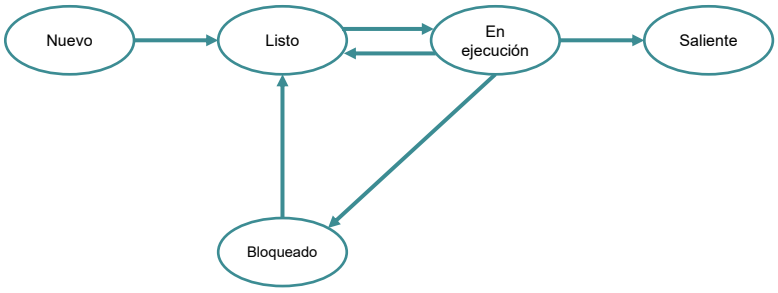
Procesos

- **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

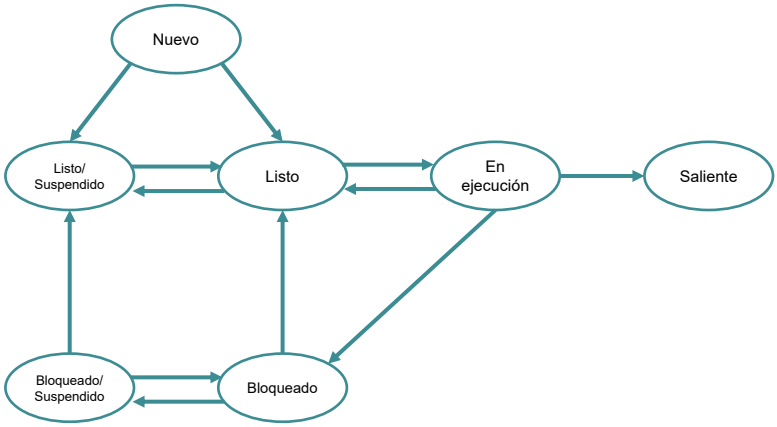
Procesos

- **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

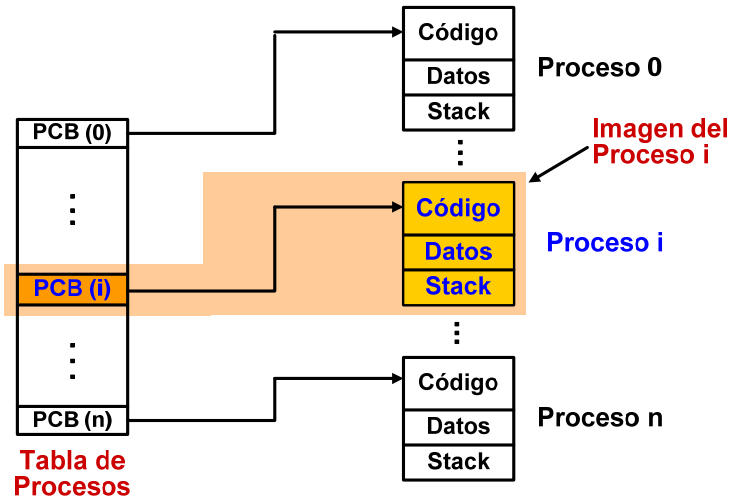
Procesos

- **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

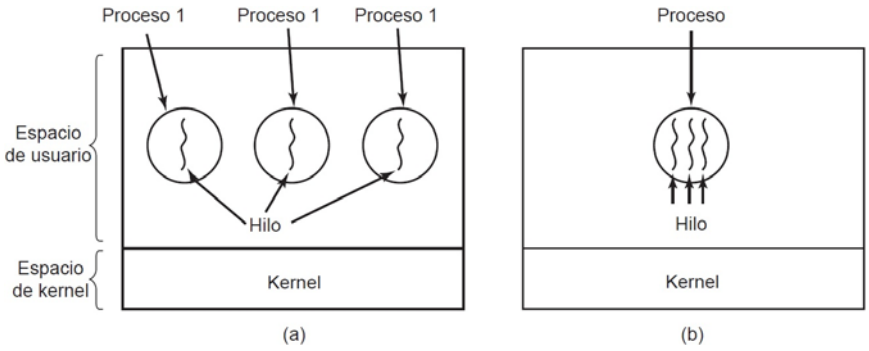
Hilos

- **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

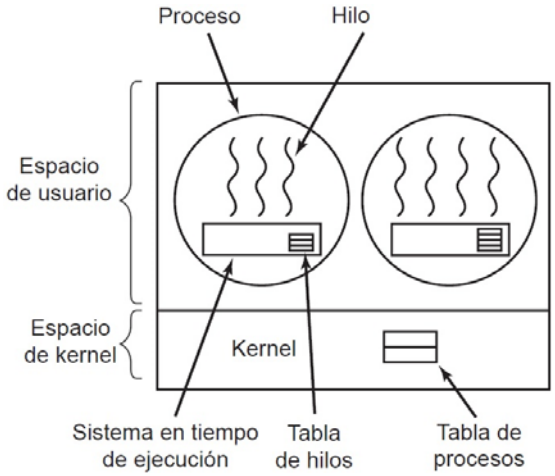
Hilos

- **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

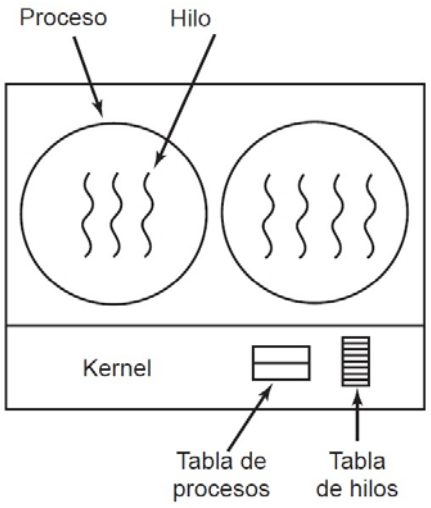
Hilos

- **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.

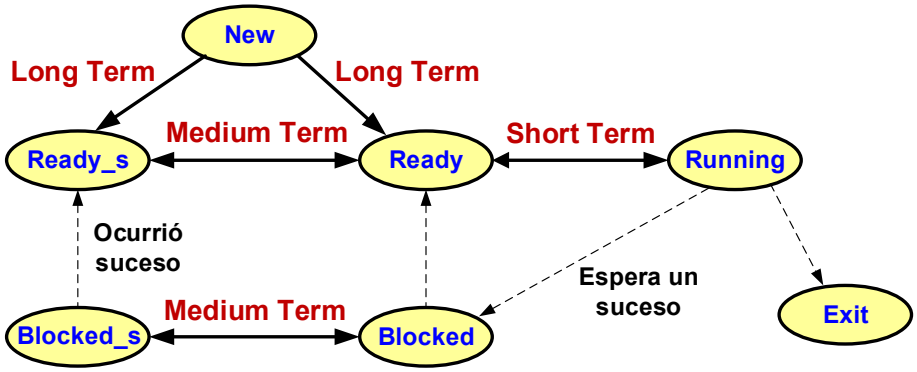
Medium Term:
Participa en la operación de swapping (en la cola Ready o en Blocked). Analiza la disponibilidad de memoria principal en relación a las prioridades de los procesos.

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



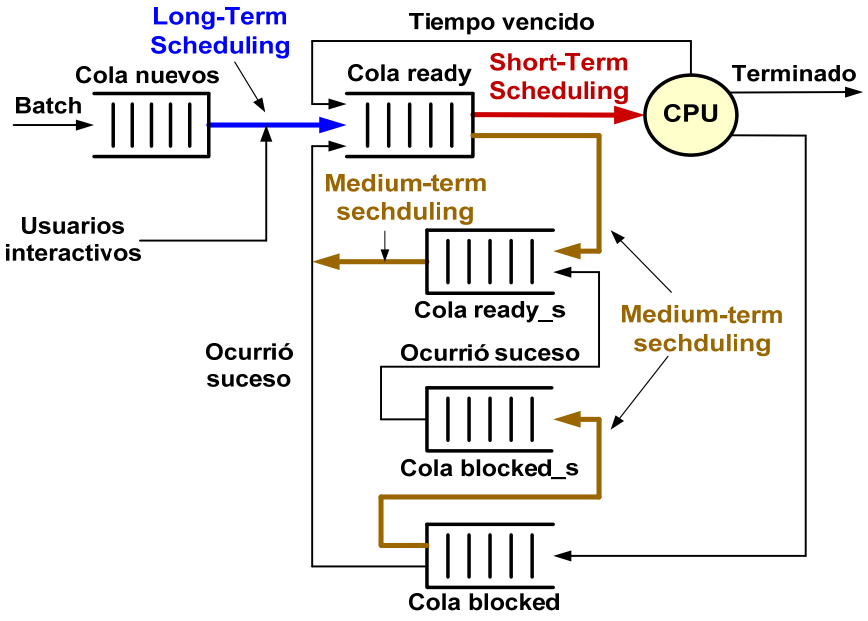
10

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

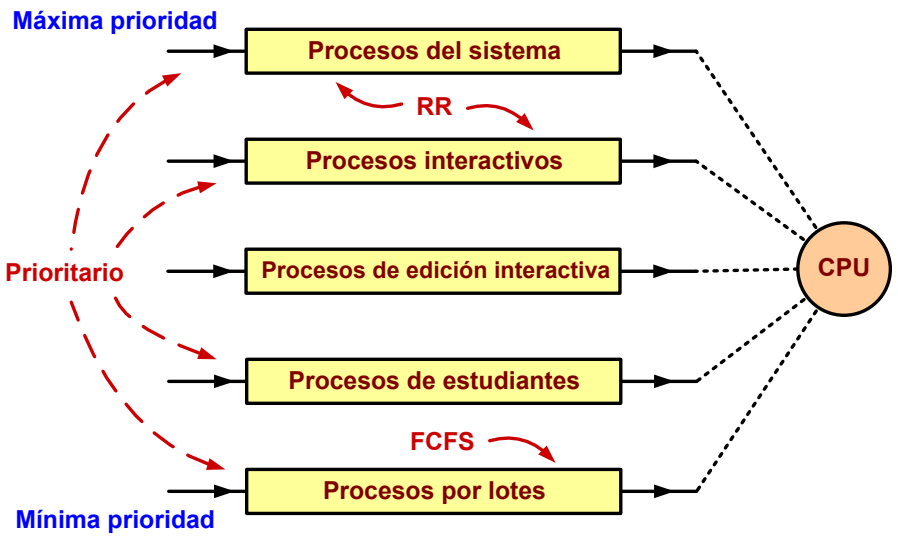
Algoritmos de planificación *(cont.)*

- **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.

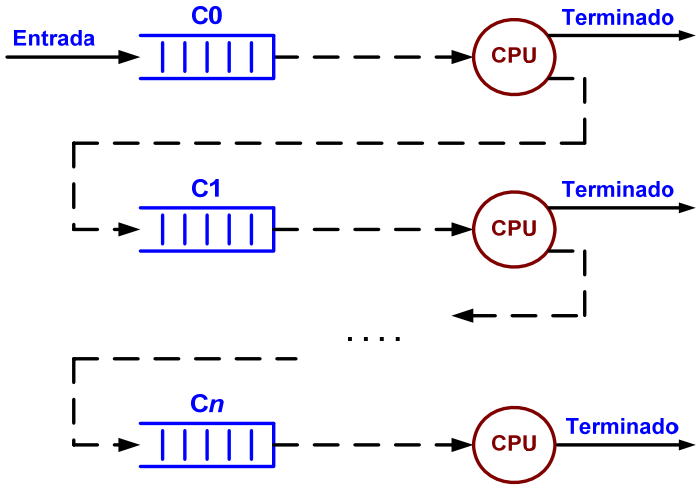
- Cuando un proceso ingresa por primera vez al sistema es puesto en la cola de prioridad C0 (máxima).
- Después de su primera ejecución, si no ha finalizado, el proceso vuelve al estado Ready y es puesto en la cola de prioridad C1 (inferior a C0).
- En resumen: después de cada ejecución, si no ha finalizado,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

Sistemas Operativos I

Módulo IV

ADMINISTRACIÓN DE LA MEMORIA

1

Temas del Módulo IV

- Introducción.
- Preparación de un programa para su ejecución.
  - Compilación.
  - Linkeado.
  - Cargado.
  - Ejecución.
- Esquemas simples de administración de memoria.
  - Particiones fijas.
  - Particiones variables. Compactación.
  - Overlay.
  - Swapping.
- Principios de memoria virtual.
  - Segmento único por proceso.
  - Múltiples segmentos por proceso.
  - Paginado.
  - Segmentación con paginado.
- Implementación de memoria virtual.
  - Tablas de segmentos.
  - Tablas de páginas.
- Esquemas de asignación de memoria.
  - First Fit, Best Fit.
  - Estático, Dinámico.

2

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.

3

Introducción

Objetivos de los Sistemas Multiprogramados.

Para que los recursos del sistema sean compartidos dinámicamente por todos los procesos de manera eficiente, es necesario:

- En general, respecto del sistema:

  mejorar la respuesta global del sistema, incrementando la utilización de estos recursos.

- En particular, respecto de la memoria:

  tener un alto grado de multiprogramación. Esto significa: mantener múltiples procesos activos en la memoria principal del sistema.

  Esta tarea es realizada por el módulo del SO denominado **Administrador de la Memoria**.

4

Introducción

Tarea de Administración de la Memoria

- Lograr un alto grado de multiprogramación implica:
  asignar espacios de memoria a múltiples procesos administrando eficientemente la memoria
- La tarea de administración de la memoria es realizada por:
  - **Módulo de Administración de Memoria (SO)**
  - **MMU o Unidad de Manejo de Memoria (Hw)**

  Ambos elementos están fuertemente relacionados.

5

Preparación de un programa para su ejecución

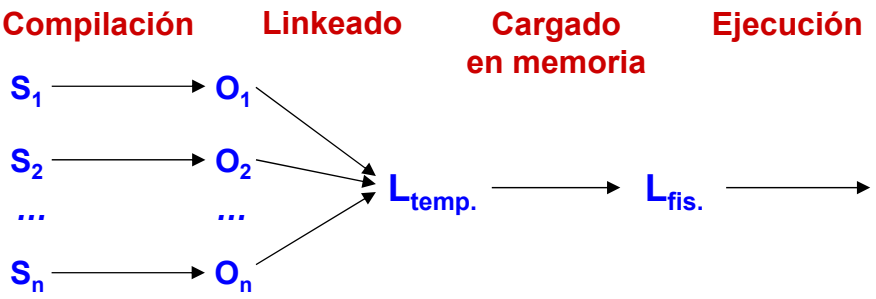
Para poder comprender los diferentes esquemas de manejo de la memoria, antes es conveniente recordar cómo se prepara un programa para su ejecución.

Téngase presente que:

- La mayoría de los programas de usuario y del Sist. Operativo están escritos en lenguajes de alto nivel (C, C++). Un programa así escrito se denomina **programa fuente (S_i)**.
- Para que el programa fuente pueda ser ejecutado, necesita pasar por una serie de transformaciones previas:

6

Preparación de un programa para su ejecución



S_i: programa fuente

O_i: módulo objeto

L_{temp}: módulo cargable (reubicable)

L_{fis}: módulo ejecutable

7

Preparación de un programa para su ejecución

Compilación.

Objetivo:

- Construir **Módulo objeto (O_i)** a partir del programa fuente **(S_i)**.

Tarea:

- Traducción de lenguaje de alto nivel a lenguaje de máquina.

Descripción de la tarea:

- Puede hacerse en más de una etapa. Implica el análisis sintáctico, semántico y léxico de las instrucciones de alto nivel. A partir de ese análisis se realizan las optimizaciones de hardware, por ejemplo, elegir qué registros del procesador utilizarán las variables.
- En una segunda etapa la compilación genera el código de máquina a partir de un archivo intermedio llamado "lenguaje ensamblador" (assembler). El resultado es un archivo binario.

8

Preparación de un programa para su ejecución

Linkeado

Objetivo:

Construir L_{temp} (módulo cargable y ejecutable en memoria)

Tarea:

- Unión de varios módulos objeto **(O_i)** en un único binario.

Descripción de la tarea:

- Normalmente, los programas complejos son escritos en varios módulos, los que deben unirse para formar un único programa.
- Función principal de esta etapa: resolver las referencias de localizaciones de operandos, instrucciones, estructuras de datos de los distintos módulos a referencias (direcciones) de memoria.
- Las referencias de memoria en L_{temp} son direcciones relativas, no absolutas (direcciones reales), debido a que el SO no sabe, en este momento, en qué lugar de la memoria será ubicado.

9

Preparación de un programa para su ejecución

Cargado en memoria

Objetivo:

- Construcción de L_{fis} (módulo cargado en memoria principal para ser ejecutado).

Tarea:

- Transformación de L_{temp} a L_{fis}

Descripción de la tarea:

- La ubicación en memoria se determina en el momento en que L_{temp} va a ser cargado desde disco para ser ejecutado.
- Programa cargador o loader: carga el programa en memoria principal y transforma las referencias relativas en absolutas.
- Referencias absolutas: direcciones de memoria principal (direcciones físicas)

10

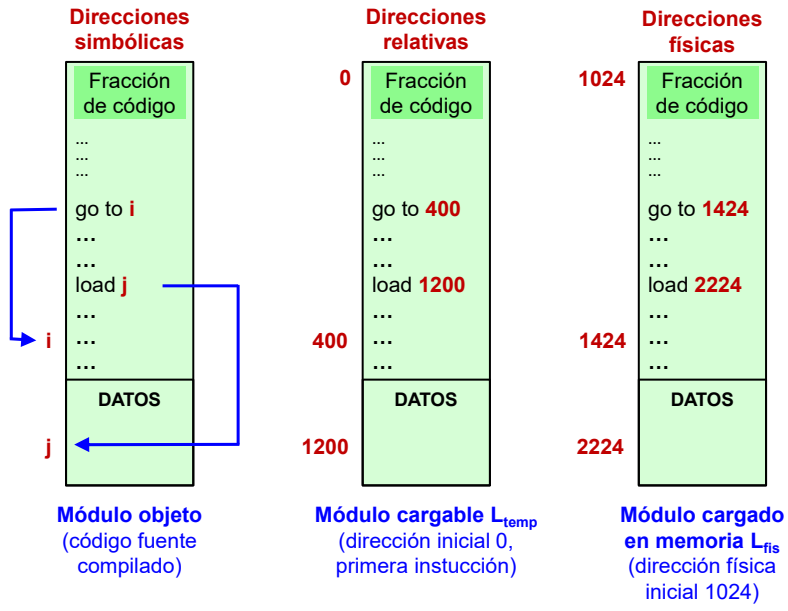
Preparación de un programa para su ejecución

Resumen de traducciones de direcciones

- Cuando se programa en alto nivel: se usan direcciones simbólicas. (el programa solo está escrito o editado).
- Cuando se construye L_{temp} se traducen las direcciones simbólicas en relativas. (varios módulos unidos en una tarea)
- Cuando se instala en memoria (L_{fis}): se traducen direcciones relativas en absolutas o físicas (direcciones reales de memoria)

11

Ejemplo de traducción de direcciones



12

Esquemas simples de administración de memoria

Planteo del problema de administración de la memoria.

La necesidad de administrar la memoria nace del objetivo de tener multiprogramación en el sistema.

(1) Esto implica que: varios procesos activos deben compartir la memoria al mismo tiempo para poder ejecutar.

Si bien el costo del hardware decrece continuamente, la memoria es un recurso siempre escaso en el computador debido, precisamente, a su costo.

Conclusión: no se puede satisfacer (1) de manera directa y simple puesto que el tamaño resultante de la suma de todos los programas en el sistema, normalmente, excede la capacidad de la memoria principal del computador.

- Se plantea así un problema a resolver: **debe administrarse la capacidad disponible de memoria principal.**

13

Esquemas simples de administración de memoria

Planteo del problema de administración de la memoria

(cont.)

- Para solucionar el problema expuesto, necesariamente debe realizarse la siguiente tarea:

  Los programas deben ser divididos de alguna forma en partes que serán cargadas desde el disco a memoria, sólo cuando esos programas vayan a ser ejecutados.

- A lo largo de la historia de la computación se han desarrollado y usado distintas técnicas para llevar a cabo la tarea mencionada.
- Se estudiarán esas técnicas, comenzando por las más simples.

14

Esquemas simples de administración de memoria

Particiones Fijas de la Memoria

- El esquema de partición de memoria más simple y antiguo.
- La memoria se divide en un número de espacios separados llamadas particiones que tienen tamaños diferentes para almacenar procesos de distintos tamaños.
- Los tamaños son fijados en el momento en que el sistema se inicializa (levanta) y no pueden ser modificados más; es decir, la memoria se divide antes de iniciarse la ejecución de los programas.
- Problema de partición fija. Hay dificultad en estimar el tamaño de las particiones: es complicado predecir el tamaño preciso de los programas que correrán en el sistema.
- Una estimación incorrecta de las particiones puede llevar a una mala utilización de la memoria.

15

Esquemas simples de administración de memoria

Particiones Fijas de la Memoria *(cont.)*

- Las particiones fijas imponen la necesidad de planificar los procesos que las van a ocupar; es decir, se debe asignar un proceso a cada partición.
- Existen dos esquemas: asignación estática y dinámica.

1 - Asignación Estática: se crea una cola de procesos por cada partición.

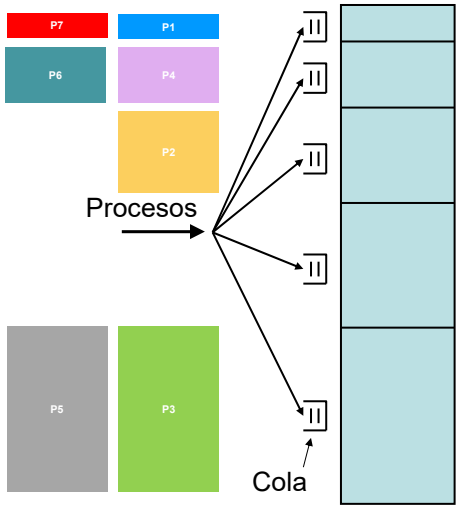
Los procesos son planificados para ocupar la partición que más se ajuste a su tamaño.

Ventaja: es la más simple de implementar.

Desventaja: algunas particiones pueden quedar vacías mientras en otras colas pueden haber procesos de tamaño menor que estas particiones. Esto lleva a una baja utilización de la memoria.

16

Esquemas simples de administración de memoria



Nunca se utilizó…

Asignación estática
Una cola de procesos
por cada partición

17

Esquemas simples de administración de memoria

Particiones Fijas de la Memoria *(cont.)*

2 - Asignación dinámica: una única cola de procesos para todas las particiones.

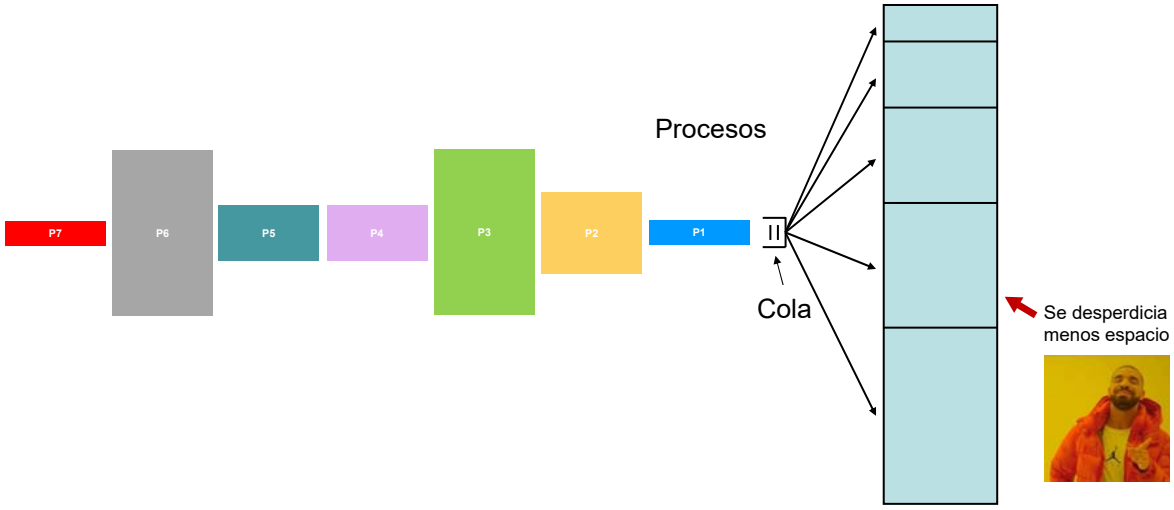
Los procesos que ingresan al sistema son puestos en una única cola. Desde esta cola cada proceso va ubicándose en la partición adecuada de acuerdo a su tamaño.

Ventaja: se obtiene una mejor utilización de la memoria, puesto que si existe un espacio desocupado en ésta que pueda albergar a un proceso listo a ser ejecutado, es usado aunque no se ajuste exactamente al tamaño del proceso.

Desventaja: complejidad en la implementación.

18

Esquemas simples de administración de memoria



Asignación dinámica
Una única cola de procesos
para todas las particiones

19

Esquemas simples de administración de memoria

Particiones Variables de la Memoria.

Este esquema se caracteriza porque la memoria se asigna según el tamaño de cada proceso y en el momento en que será cargado en memoria; es decir, es una asignación dinámica.

- Aquí se deben considerar 2 cuestiones:
  1° - los procesos no finalizan su ejecución en el mismo orden en que ingresaron al sistema, además,
  2° - los procesos tienen distinto tamaño,
  al final, la memoria es un conjunto de bloques ocupados y vacíos de diferentes tamaños.
- Esta situación lleva a un problema, llamado fragmentación externa de la memoria.

20

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. *(cont.)*

Fragmentación externa de la memoria: es cuando existen varios bloques libres de memoria de tamaño tan reducido que no pueden ser ocupados por ningún proceso.

Tarea principal del módulo Administrador de la Memoria en este esquema:

- Mantener una tabla con los espacios libres que se van asignando a los procesos nuevos a medida que ingresan al sistema.
- Los espacios liberados por los procesos que finalizaron deben ser unidos para formar grandes bloques libres de memoria y evitar así la fragmentación de la memoria.

Solución: compactar la memoria.

21

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación

- Considérese la siguiente situación: en un momento dado ingresa un proceso al sistema que no cabe en ninguno de los huecos vacíos de memoria pero, sin embargo, el total de varios huecos no contiguos es mayor que el tamaño del proceso.
- Pregunta: qué debe hacer el SO? (Ver Ejemplo en figura sigte.)

Hi = huecos (espacios vacíos de memoria)

A, B, C = Procesos instalados en memoria.

Por ejemplo, supóngase que **B** finaliza dejando un hueco de tamaño **H2** entre **A** y **C**.

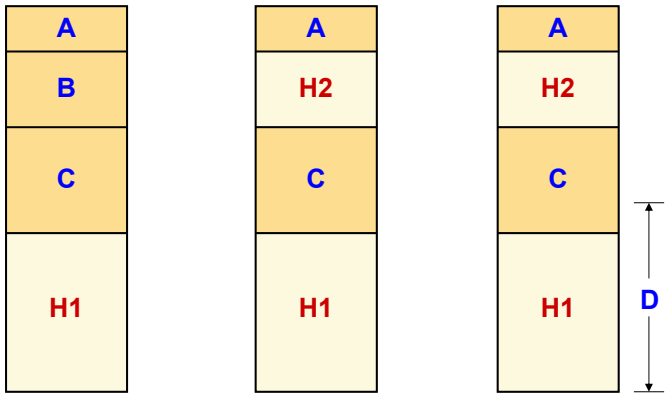
Luego el proceso **D** es elegido para ejecutar: no podrá ocupar la memoria porque su tamaño excede a **H1** y también a **H2**.

No obstante, si se juntan los huecos **H1** y **H2**, el espacio total alcanza y sobra para contener al proceso de tamaño **D**.

22

Esquemas simples de administración de memoria

Particiones Variables de Memoria. Compactación



D > H1 ; D > H2

D < H1 + H2

23

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación *(cont.)*

Solución 1: D espera hasta que finalice **C**.

Solución no óptima desde el punto de vista de performance y utilización de memoria.

Solución 2: se reubica **A** y **C** para lograr un espacio libre mayor.

Compactación de la memoria

Es la acción de desplazar bloques de datos dispersos en la memoria y agruparlos para crear espacios libres de gran tamaño.

24

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación *(cont.)*

Ejemplo.

(en la figura sigte. el número en cada bloque indica su tamaño).

Supóngase que ingresa un nuevo proceso cuyo tamaño es 10.

Evidentemente, ningún hueco puede alojar a este proceso - fig.(a) -

Solución de Compactación 1: desplazar todos los bloques de memoria a un extremo, p/ej: al principio -fig.(b)-.

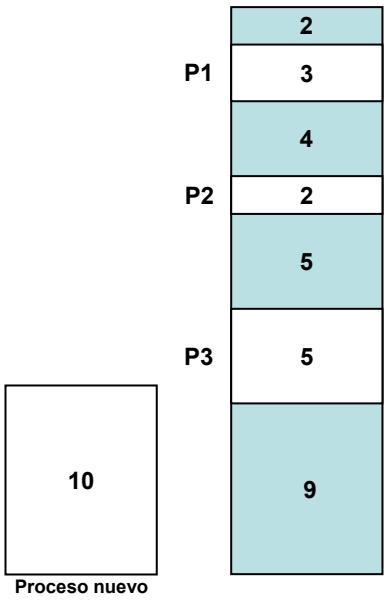
De esta manera se crea un hueco de tamaño 20 y así se puede cargar el proceso nuevo.

Ventaja: es simple.

Desventaja: movimiento excesivo de bloques ocupados. Reduce la performance global del sistema.

25

Esquemas simples de administración de memoria



Compactación de Memoria: Solución 1.

26

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación *(cont.)*

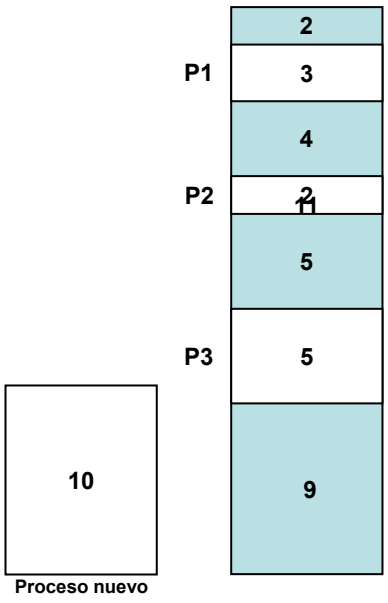
Solución de Compactación 2: desplazar sólo los bloques necesarios hasta crear un hueco igual o mayor que 10 -fig.(c)- Se movieron P1 y P2 para crear un hueco de tamaño 11 (fig. sigte.)

Ventaja: menor movimiento de bloque ocupados.

Desventaja: algoritmo no tan simple.

27

Esquemas simples de administración de memoria



Compactación de Memoria: Solución 2.

28

Esquemas simples de administración de memoria

Solución de Compactación 3: Buscar el mínimo movimiento de bloques -fig.(d)-. Se ha movido sólo el proceso P2 y con ello se ha logrado un hueco de tamaño 11.

Ventaja: Mínimo desplazamiento de bloques llenos.

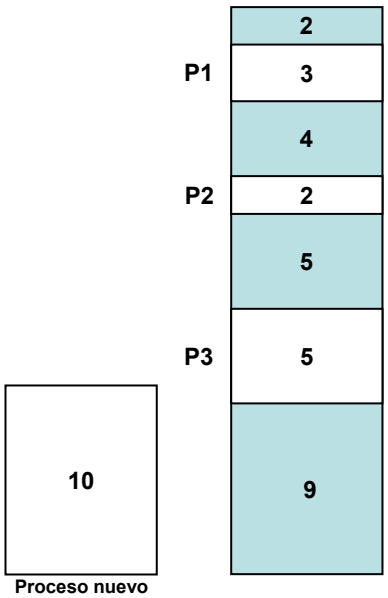
Desventaja: alta complejidad de los algoritmos encargados de determinar el movimiento mínimo de bloques.

Conclusión

Debido a la desventaja señalada se usan en mayor medida las soluciones propuestas en las figs. (b) y (c).

29

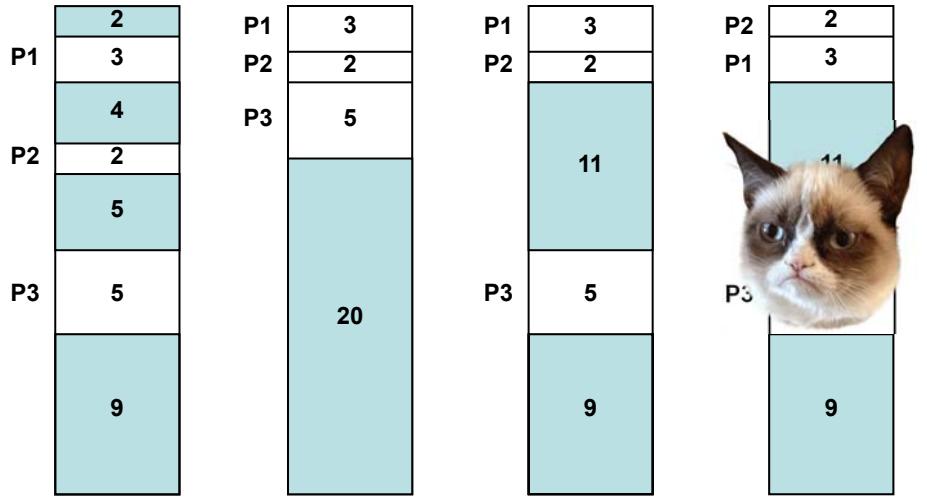
Esquemas simples de administración de memoria



Compactación de Memoria: Solución 3.

30

Esquemas simples de administración de memoria



31

Otras técnicas de administración de memoria

Técnica de Overlay

El tamaño físico - real - de la memoria principal impone varias limitaciones en sistemas mono y multiprogramas:

- **Sistemas Monoprograma.** El tamaño máximo de un proceso no debe ser mayor que el espacio disponible de memoria.
- **Sistemas Multiprograma.** El tamaño máximo posible de un proceso (en un sistema con partición fija de la memoria) es el de la mayor partición disponible.

Cuando se exceden esos límites, los programas deben ser divididos en módulos o segmentos que pasarán a ocupar la memoria en el momento en se los acceda, dejando en el disco los módulos no necesarios del programa en ejecución.

Esta técnica se denomina **overlay** o uso de overlays: significa reemplazo de módulos.

32

Otras técnicas de administración de memoria

Técnica de Overlay *(cont.)*

Esta tarea era realizada por el programador, quién debía especificar, en cada momento, cuáles de los segmentos del programa debían residir en memoria en forma simultánea para una ejecución satisfactoria.

Desventaja de la técnica de overlay: dependencia del programador en una tarea que debería ser realizada por el SO.

33

Otras técnicas de administración de memoria

Técnica de Swapping

- Es el pasaje de procesos de memoria primaria a secundaria y viceversa.
- Sea el escenario siguiente: arriba un proceso de alta prioridad al sistema requiriendo una cierta cantidad de memoria que no hay disponible.
- Ante esta situación: el SO desaloja de memoria uno o más procesos y los pasa a disco para dejar libre el espacio suficiente requerido por el proceso prioritario.
- Esta operación se denomina **swapp-out**.
- Cuando este proceso finalice su ejecución, los procesos serán pasados nuevamente de disco a memoria primaria (**swapp-in**).

34

Características de las soluciones vistas

Las soluciones anteriores no son óptimas porque:

- **Pueden llevar a una baja utilización de la memoria:**
  En asignación estática de memoria con particiones fijas: mucho desperdicio de espacio al no coincidir exactamente el tamaño de los segmentos de datos con los espacios disponibles.
- **Requieren la utilización de algoritmos de alta complejidad:**
  En asignación variable de memoria: para búsqueda de espacios libres en donde ubicar un segmento de datos y para compactación.
- **Necesitan la participación directa del programador:**
  Como en la técnica de overlay.

35

Principios de Memoria Virtual

Conceptos preliminares de memoria virtual

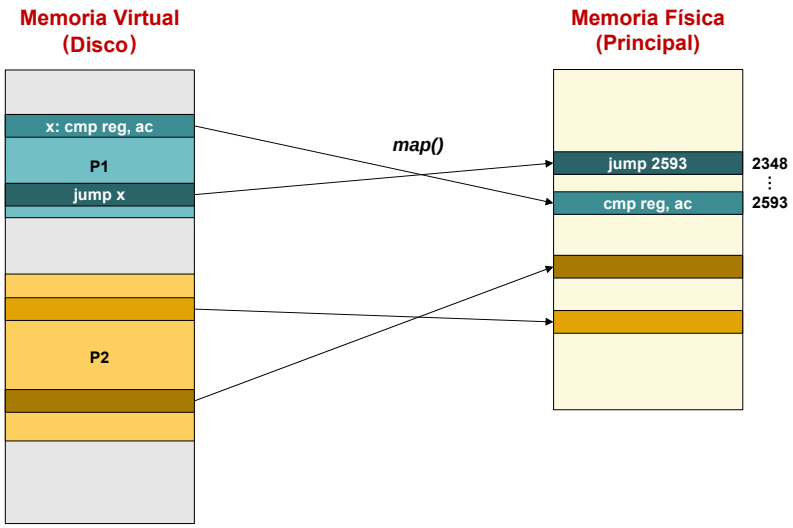
- Se trata de ocultar al usuario que la memoria principal tiene un tamaño físico limitado, de tal forma que él tenga la idea que la misma es continua e infinita.
- Esto se logra usando el disco como complemento de la memoria principal.

En concreto: al usuario de un sistema que usa memoria virtual:

1. Se le oculta que la memoria física es limitada en espacio y está compartida por varios procesos.
2. En cambio, se le presenta la visión de que a cada proceso se le provee un espacio privado e infinito de memoria.
   Estos espacios otorgados a los procesos son contiguos, es decir, están uno al lado del otro.
   ¿Cómo se materializa lo dicho?

36

Principios de Memoria Virtual



37

Principios de Memoria Virtual

Conceptos preliminares de memoria virtual *(cont.)*

En la figura anterior se muestra un ejemplo en el que se observa:

- Dos espacios de memoria virtual (MV1 y MV2) que pertenecen a un mismo proceso P
- Parte de MV1 y MV2 se han cargado en memoria física o principal
- La función map realiza el mapeo de direcciones

Mapeo

- Significado: traducción de direcciones virtuales a direcciones físicas.
- Tarea de traducción: es llevada a cabo por la función map( ).
- Implementación de la función map(): MMU (Memory Manager Unity).

38

Principios de Memoria Virtual

Conceptos preliminares de memoria virtual *(cont.)*

Direcciones virtuales

Conceptualmente, son similares a las direcciones relativas de L_{temp} vistas en el ejemplo anterior.

Direcciones físicas

Son similares a las direcciones absolutas o físicas de L_{fis}. Sus valores dependen del espacio físico en dónde está ubicado el proceso en memoria principal.

En la diapositiva que sigue se muestra el gráfico de módulo objeto, recargable y cargado en memoria.

39

Implementación de la memoria virtual

¿Cuándo debe realizarse el mapeo?

- Cuando los procesos listos para ser ejecutados o en ejecución contienen en sus códigos direcciones que referencian a alguna otra instrucción o dato, se trata de direcciones virtuales.
- Cuando una instrucción va a ser ejecutada y contiene una dirección virtual, antes debe realizarse el mapeo, es decir, debe encontrarse la dirección física correspondiente.
- Lo anterior resulta obvio, puesto que el procesador solo maneja direcciones físicas (direcciones de memoria principal)

Conclusión:

La función mapeo debe actuar cada vez que el procesador invoca una instrucción que contiene una dirección virtual.

40

Implementación de la memoria virtual

En la **implementación de la memoria virtual** se estudiarán diferentes

- **Modos de asignación de memoria virtual a los procesos** y, para cada modo de asignación, se verá la
- **Implementación del mapeo**, es decir, cómo se traducen direcciones virtuales a direcciones físicas.

41

Implementación de la memoria virtual

Modos de asignación del espacio de memoria virtual

- **Espacio de Segmento Único por Proceso.**
  Asignación de almacenamiento contiguo: cada proceso ocupa un bloque de memoria que es indivisible.
- **Paginado.**
  Asignación de páginas: cada proceso ocupa un espacio de memoria que se divide en bloques pequeños llamados páginas.
- **Espacio de Múltiples Segmentos por Proceso.**
  1. Asignación continua de segmentos: cada proceso ocupa varios segmentos indivisibles de memoria.
  2. Asignación de múltiples segmentos con paginado: cada proceso ocupa varios segmentos los que, a su vez, son divididos en páginas.

42

Espacio de segmento único por proceso

Significa que:

- Cada proceso ocupa un único segmento indivisible.
- Por lo tanto, cuando es cargado desde el disco a memoria principal, se trae todo el segmento que ocupa el proceso.

Asignación de almacenamiento contiguo

Debido a que cada proceso ocupa un segmento único e indivisible en memoria virtual, cuando un dado proceso es traído a memoria física ocupará una región contigua (bloque único) de memoria física.

43

Espacio de segmento único por proceso

Implementación del Mapeo

- Se usa un **Registro de Reubicación** (**RR**) contenido en MMU. **RR** contiene la dirección base del espacio en memoria física que ocupa el proceso que se está ejecutando en ese momento.

Secuencia de ejecución de un proceso que está en memoria virtual

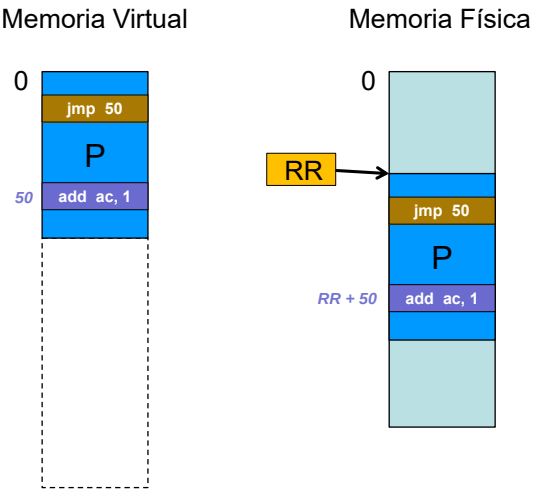
1. El SO carga el segmento que ocupa el proceso (código y datos) en un espacio libre de memoria principal.
2. A continuación carga en el **RR** la dirección física de comienzo del segmento. Si esta dirección es también el comienzo del código a ejecutar, deberá cargarse en el PC (program counter).
3. Como ya se dijo, se realiza el mapeo cada vez que se ejecuta una instrucción del programa que invoque una dirección virtual.

44

Espacio de segmento único por proceso

Implementación del Mapeo en un proceso cargado.

```
map (dir_virtual)
{
    dir_fisica = RR + dir_virtual;
    return dir_fisica;
}
```



45

Espacio de segmento único por proceso

Implementación del Mapeo en un proceso cargado.

Supóngase que se va a ejecutar un proceso y se ha realizado lo indicado en los puntos 1 y 2 anteriores.

3. El mapeo se implementa de la siguiente forma:

```
map(dir_virtual)
{
   dir_fisica = RR + dir_virtual
   return dir_fisica
}
```

(Aquí se ha supuesto que al inicio del segmento en memoria virtual se le ha otorgado la dirección virtual 0)

Ver esquema siguiente.

46

Espacio de segmento único por proceso

Implementación del Mapeo en varios procesos cargados.

La implementación del mapeo vista permite la existencia de varios procesos cargados en memoria física simultáneamente, donde cada proceso ocupa un único segmento.

En el caso de múltiples procesos activos en el sistema, el SO debe tener almacenado un **Vector de Estado** en el PCB de cada proceso **P** con la siguiente información:

- Un contador de instrucciones de memoria virtual ("program counter of virtual memory"): $\mathbf{PC_V}$.
- La dirección base ($\mathbf{DB_P}$) de la memoria física que ocupa **P**.

Vector de Estado: PC_V ; DB_P

47

Espacio de segmento único por proceso

Ejemplo

Se produce un cambio de proceso entre **P1** y **P2**. (P1 deja el procesador y P2 comienza a ejecutar en su lugar).

Pasos para realizar el mapeo de direcciones de P2

1. $\mathbf{PC_{P1} = PC_v}$; $\mathbf{DB_{P1} = RR}$ [se almacena en el PCB de P1 el vector de estado, es decir, el estado de ejecución de P1 dado por los registros PC_V(virtual) y RR]

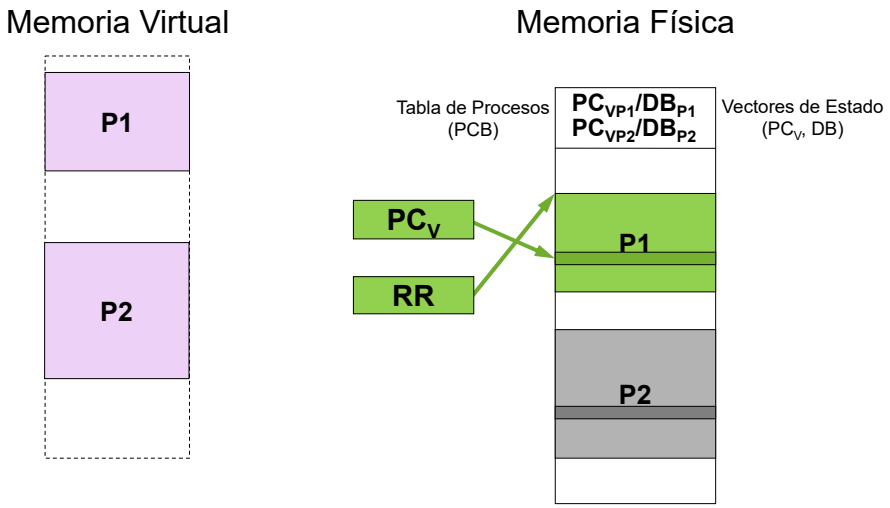
2. $\mathbf{PC_v = PC_{P2}}$; $\mathbf{RR = DB_{P2}}$ [desde el PCB de P2 se carga el vector de estado a los registros PC_V(virtual) y RR].

Una vez cargado $\mathbf{PC_v}$ desde $\mathbf{PC_{P2}}$ y **RR** desde $\mathbf{DB_{P2}}$, se obtiene la dirección física mediante el mapeo que implementó el programa expuesto anteriormente.
Si **RR** es la dirección física de inicio del código de P2, es cargada además en el $\mathbf{PC_f}$ (program counter de memoria física).

48

Espacio de segmento único por proceso

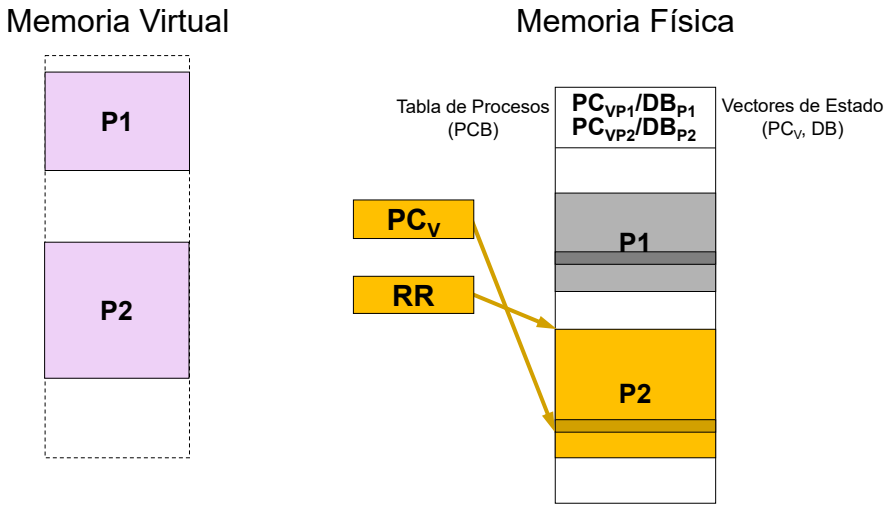
Implementación del Mapeo en varios procesos cargados.



49

Espacio de segmento único por proceso

Implementación del Mapeo en varios procesos cargados.



50

Espacio de segmento único por proceso

Ventaja y Limitación de Segmento Único por Proceso

- Ventaja: es muy fácil de implementar la ubicación y el mapeo. Para realizar el mapeo sólo se necesita especificar la dirección base (DB) del segmento que se va a ubicar en memoria física.
- Limitación: el tamaño del segmento en memoria virtual debe ser igual o menor que el espacio disponible en memoria física.

  Esta limitación, más la necesidad de compactar la memoria por la fragmentación externa, hace que sea más conveniente emplear otra técnica, que es la **técnica de paginado**.

  En este sentido, la técnica de paginado divide el espacio ocupado por los procesos en pequeños segmentos, todos de un mismo tamaño, eliminando así la fragmentación externa.

  La técnica de paginado es utilizada en los sistemas actuales, con distintas variantes.

51

Técnica de paginado

- El término **paginado** se usa para describir una particular implementación de memoria virtual y organización de memoria física.

Páginas en Memoria Física y Dirección Física

- Se divide la memoria física en un número dado de páginas que son bloques de memoria con un tamaño fijo.
  Las páginas se identifican como $\mathbf{f_0, f_1, f_2, \ldots f_{F-1}}$.
- Tamaño típico de una página: entre 1.024 y 4.096 palabras.
- Una **dirección de memoria física - df -** es un par ordenado **(f, w),** donde:

  f: es el número de página física.

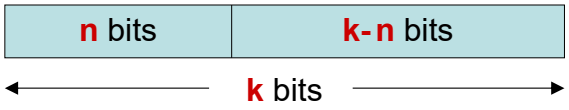
  w: es un número de palabra de la página **f**.

  (**w** se puede ver como un desplazamiento desde la dirección inicial de la página **f**).

52

Técnica de paginado

Suponiendo que una dirección **df** en memoria física tiene **k** bits, el esquema de paginado usa los primeros **n** bits para indicar el número de página **f** y los **k- n** restantes para indicar el número de palabra (desplazamiento **w**) dentro de la página.



Eso significa que está formada por f + w bits, pero para no confundirnos, cuando cuando nos refiramos a la cantidad de bits, vamos a encerrar **f** y **w** entre barras horizontales:

Entonces: **| df | = | f | + | w |**

53

Técnica de paginado

Podemos realizar los siguientes cálculos:

- Con |df| bits se pueden representar $2^{|df|}$ direcciones, por tanto, el tamaño de la memoria física es:

$$TMF = 2^{|df|}$$

- Con |f| bits se pueden representar $2^{|f|}$ páginas, es decir que la cantidad de páginas en que se puede dividir la memoria física, llamada F mayúscula, es:

$$F = 2^{|f|}$$

- Con |w| bits se pueden representar $2^{|w|}$ palabras de una página, por lo tanto, el tamaño de una página, medido en palabras o direcciones, es:

$$W = 2^{|w|}$$

54

Técnica de paginado

Páginas en Memoria Virtual y Dirección Virtual

De manera similar que para el espacio físico, un espacio virtual es dividido en un número de páginas contiguas denominadas $p_0, p_1, p_2, \ldots, p_{P-1}$,

donde: las páginas en memoria virtual son del mismo tamaño que las páginas en memoria física.

Una **dirección de memoria virtual - dv -** se considera como un par ordenado **(p, w)** donde **p** es el número de página virtual y **w** es la localización de la palabra referenciada dentro de la página **p** (desplazamiento dentro de la página).

55

Técnica de paginado

Páginas en Memoria Virtual y Dirección Virtual *(cont.)*

La correspondencia entre **dv** y **(p, w)** se obtiene de la misma manera que para la memoria física; es decir, dividiendo en dos la secuencia de bits de la dirección virtual:

- la 1ra. secuencia de **|p|** bits indica el número de página virtual.
- la 2da. secuencia de **|w|** bits indica el número de palabra dentro de una página p (desplazamiento).

Entonces, la cantidad de bits de una dirección virtual es:

$$|dv| = |p| + |w|$$

y la cantidad de páginas virtuales del proceso será:

$$P = 2^{|p|}$$

56

Técnica de paginado

- Cuando hablamos de palabras, no es otra cosa que una instrucción, y que ocupa una dirección. Por lo tanto estos tres términos van a ser equivalentes. El tamaño de la palabra dependerá de la arquitectura del hardware; hoy en día es de 64 bits.
- El número **w** es el mismo tanto en la página física como en la virtual, debido a que tienen el mismo tamaño. Esa palabra o instrucción es la que buscamos cuando hacemos el mapeo. Es decir, en alguna instrucción que ejecuta el proceso se hace referencia a otra instrucción o dato, entonces el mapeo consiste en encontrar en qué página física está.
- El valor **P** generalmente será mayor que **F**, ya que el proceso puede ser más grande que la memoria principal; veremos más adelante que esa es la principal ventaja del concepto de memoria virtual.

57

Implementación del mapeo en paginado

El objetivo de la función **map( )** es traducir una dirección virtual **(p, w)** en la correspondiente dirección física **(f, w).**

$$(p, w) \xrightarrow{map( )} (f, w)$$

Obsérvese que, en realidad, solo se debe encontrar **f** (a partir de **p**), puesto que **w** es el mismo en la dirección virtual y en la dirección física. (Ver gráfico. sigte.)

Una vez encontrada **f**, la dirección física **df** se obtiene como:

$$df = f \times 2^{k-n} + w$$

Obsérvese que: 2^{k-n} es el tamaño de una página (k-n = **|w|**)
y $f \times 2^{k-n}$ es la dirección inicial de la página **f**

58

Implementación del mapeo en paginado

|df| = 5 bits

|f| = 2 bits

|w| = 3 bits

T_{MF} = $2^{|df|}$ = 32 direcciones

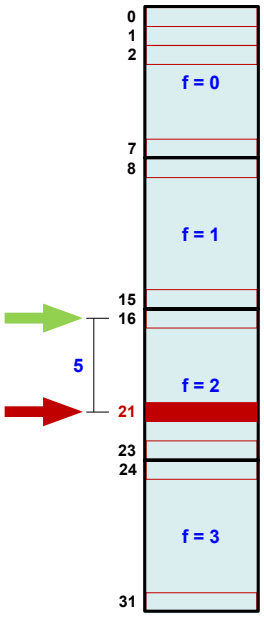
F = $2^{|f|}$ = 4 páginas

W = $2^{|w|}$ = 8 direcciones

df = (f, w) = (2, 5)

df = f x $2^{|w|}$ + w **=** 2 x 2^3 + 5 **=** 2 x 8 + 5

= 16 + 5 **= 21**



59

Implementación del mapeo en paginado

¿Cómo encontramos el valor de **f** ?

Con una **Tabla de Páginas**: arreglo que contiene las referencias entre páginas virtuales y físicas.

Esta tabla permite encontrar en qué página física se encuentra una página virtual.

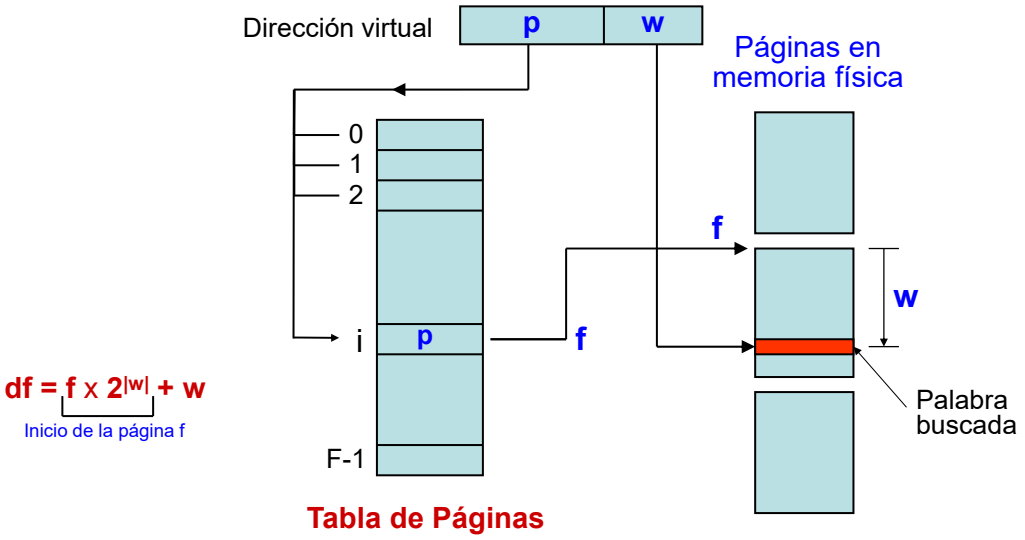
En pocas palabras, nos permite encontrar el valor de f a partir del valor de p.

La tabla tiene F entradas, igual a la cantidad de páginas físicas en memoria principal.

60

Implementación del mapeo en paginado

Implementación de la Tabla de Páginas



61

Paginado de memoria: ventajas y desventajas

Ventajas

1. La estrategia de emplazamiento o ubicación es simple.
   Esto se debe a que la unidad mínima de asignación de memoria es una página: si un proceso ocupa k páginas de memoria virtual; si hay disponibilidad en memoria física se las puede traer desde disco, no importando dónde van a ubicarse en memoria física
2. Permite la ejecución de programas de mayor tamaño que la memoria física, en forma totalmente transparente al usuario.
   Esto se implementa utilizando un esquema de asignación dinámica de almacenamiento: cuando se invoca una dirección virtual de una página que no está en memoria principal, esta página debe ser cargada desde disco.

   Este modo de operar se denomina "por demanda de páginas".
   Se necesita un mecanismo que señalice cuando una página no se encuentra en memoria principal: cuando el MMU detecta esto, se dice que hay fallo de página.

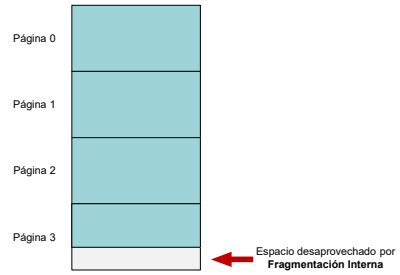
62

Paginado de memoria: ventajas y desventajas

Desventaja

Fragmentación interna: En general ningún programa ocupa con su código y sus datos el espacio disponible en la última página en forma completa. Este espacio no puede ser ocupado por ningún otro proceso ya que una página pertenece sólo a un único proceso.

Espacio promedio desperdiciado: 50% del tamaño de una página por cada proceso.



63

Espacio de múltiples segmentos por proceso

Asignación de Múltiples Segmentos a un Proceso

- Un proceso está, normalmente, compuesto por varios módulos (código, procedimientos, tablas de datos, librerías, …)
- Cuando se asigna un segmento a cada módulo, entonces cada proceso tiene asignados varios segmentos.
- Cada segmento es manejado por el SO como una unidad lógica independiente.

Ventajas:

- El SO puede acceder a estas unidades lógicas de forma más simple.
- La implementación del compartimiento y protección de acceso de estos módulos por los procesos en el sistema es natural.

64

Espacio de múltiples segmentos por proceso

La asignación de múltiples segmentos por proceso puede implementarse mediante alguna de las dos formas siguientes:

1. **Se asignan varios segmentos indivisibles a cada proceso.**

   El segmento es la mínima unidad de asignación de memoria.

   En la ejecución de un proceso, se van cargando en memoria principal los segmentos de tamaño variable a medida que se necesitan. Esta acción se denomina asignación dinámica de memoria.

2. **Se asignan segmentos a cada proceso y, a su vez, cada segmento es dividido en páginas.**

   La página es la mínima unidad de asignación de memoria

   Este esquema de asignación se denomina: paginado de segmentos.

65

Asignación de segmentos indivisibles

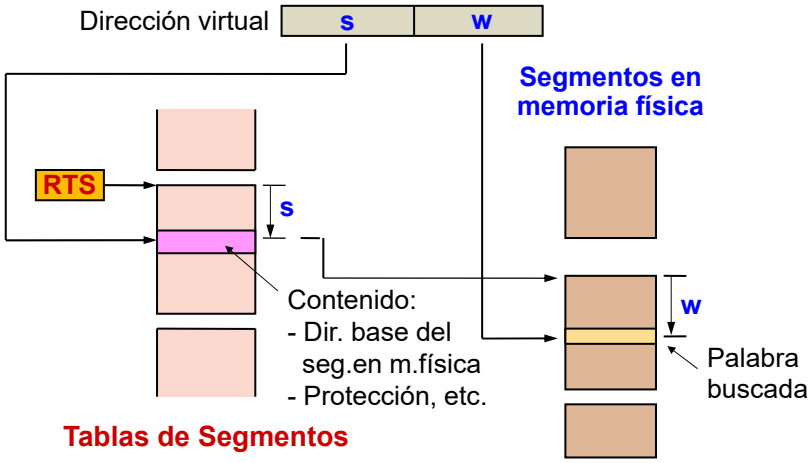
- Cada proceso consta de varios segmentos indivisibles.
- Aquí la dirección virtual que se invoca es de la forma **(s, w)** donde **s** es el número de segmento en memoria virtual y **w** la ubicación de la palabra en el segmento **s.**
- Para realizar el mapeo se utiliza una **Tabla de Segmentos** que estará almacenada en memoria principal.

  Debe existir una tabla por cada proceso activo en el sistema.
- Cada elemento o entrada de esta tabla contiene la dirección base de un segmento (en memoria física) que pertenece al proceso.
- El hardware del computador debe proveer un **Registro de Tabla de Segmentos (RTS)** que apunta al comienzo de la **Tabla de Segmentos** del proceso que se encuentra ejecutando en ese momento en la CPU.

66

Asignación de segmentos indivisibles

Tabla de Segmentos de longitud S en Memoria Principal



67

Asignación de segmentos indivisibles

Ventaja:

- Divide al programa en componentes que preservan su estructura lógica. De esta manera se simplifica el linkeado y el compartimiento y protección de procedimientos y datos.

Desventajas:

- Alta complejidad de las rutinas de ubicación y reubicación de segmentos debido a que estos tienen diferentes tamaños.
- Fragmentación externa. Se requiere compactar la memoria cada tanto.

68

Segmentación con paginado

- Significa que a cada proceso se asignan varios segmentos y, a su vez, cada segmento consta de varias páginas.
- De esta forma se combinan las ventajas de los dos esquemas vistos (segmentación simple y paginado simple).
- Para especificar una dirección virtual en este esquema, se debe agregar un nivel más de indirección que en paginado simple:

  Una **dirección virtual** consta, entonces, de una terna de la forma **(s, p, w)**, donde:

  s: número de segmento.
  p: número de página dentro del segmento **s**.
  w: desplazamiento dentro de la página **p**.

69

Segmentación con paginado

Implementación del Mapeo

La función de mapeo se puede implementar usando:

- una **Tabla de Segmentos** por proceso,
- una **Tabla de Páginas** por segmento, y
- un registro **RTS** que apunta a la Tabla de Segmentos del proceso que en ese momento está ejecutando en la CPU.

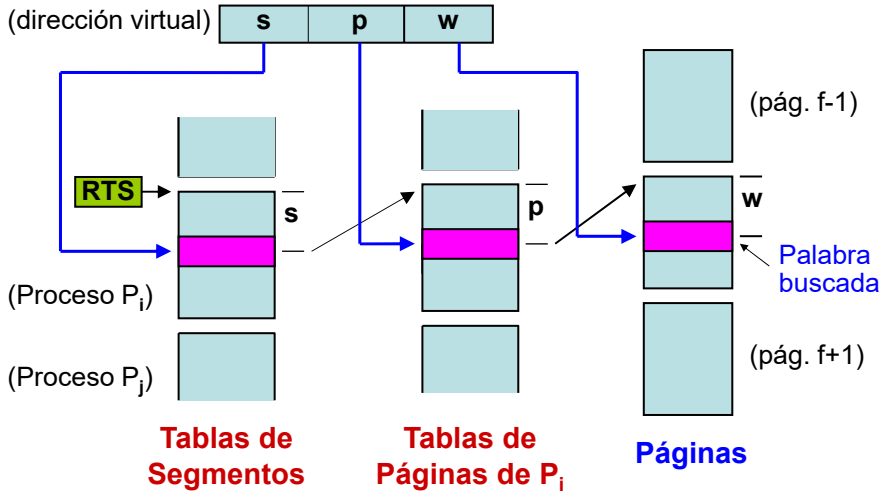
En este esquema de mapeo propuesto:

- Con el contenido de cada entrada de la Tabla de Segmentos se puede calcular la dirección de inicio de una Tabla de Páginas
- Con el contenido de cada entrada de la Tabla de Páginas se puede calcular la dirección inicio de una página en memoria física.

70

Segmentación con paginado

Implementación del Mapeo



71

Segmentación con paginado

Ventaja

Suma las ventajas que tienen por separado la segmentación y el paginado.

Desventajas

1. Se requiere un espacio considerable de memoria principal para albergar las tablas de segmentos y, especialmente, de páginas.
2. La administración de este esquema tan complejo de memoria virtual puede llevar a una disminución de la performance global del sistema.
3. Se necesitan hacer 3 referencias para acceder a una palabra en memoria física: una a la T. de Segmentos, otra a la T. de Páginas, y otra a la palabra misma.

72

Esquemas de asignación de memoria primaria

Consiste en:

Ante el pedido del SO de espacio en memoria primaria para albergar un bloque de datos que se traerá desde disco, se debe buscar un espacio igual o mayor al solicitado.

Pueden ocurrir 2 cosas:

- Que exista un espacio libre adecuado en memoria principal. En ese caso dicho espacio es asignado al bloque de datos.
- Que no exista un espacio libre adecuado en mem. principal. En ese caso, primero se debe liberar un espacio y luego asignarlo al bloque de datos (segmentos o páginas) que se traerán desde disco.

Se van a estudiar a continuación 2 esquemas de asignación:

- **Asignación de memoria en sistemas no paginados (segmentos)**
- **Asignación dinámica de memoria en sistemas paginados**

73

Asignación de memoria en sistemas no paginados

Estas operaciones pueden realizarse invocando los siguientes comandos o primitivas:

- ***Request (****Mem_Primaria, Tamaño_Pedido****)***
- ***Release (****Mem_Primaria, Tamaño_Liberado, Dir_Base****)***

Donde:

Tamaño_Xxx: número de unidades contiguas solicitadas (*Request*) y liberadas (*Release*).

Dir_Base: dirección donde comienza el bloque de celdas liberadas.

Resumen de la Operación: se solicita (*request*) un determinado espacio de memoria primaria; a partir de lo cual, se libera (*release*) una cantidad igual o mayor de memoria para asignar

74

Asignación de memoria en sistemas no paginados

Ejemplo

El módulo de administración de memoria de un SO necesita traer desde disco un bloque de datos de tamaño ***k*** de un proceso.

El comando correspondiente será:

Request *(Mem_Primaria, k)*

A partir de este pedido, pueden suceder 2 cosas:

a) **Existe** un hueco H de tamaño *h*, tal que *h* >= *k*.
En tal caso se asignan *k* unidades de H al proceso solicitante.

b) **No existe** un hueco de tamaño *h* tal que *h* >= *k* .
Aquí se pueden adoptar varios criterios:

- Bloquear el proceso solicitante hasta que se disponga de un hueco del tamaño necesario (liberación voluntaria), o
- 1) desalojar uno o más bloques de memoria, o 2) reubicar esos bloques en la misma memoria, ó 3) compactar la mem.

75

Asignación de memoria en sistemas no paginados

Algoritmos de Ubicación de Bloques de Tamaño Variable

Encuentran cuál de todos los bloques disponibles, que cumplen con la condición *h* >= *k*, se asigna al proceso que invoca el comando *Request ()*. Hay dos algoritmos: First-Fit y Best-Fit.

Antes de analizar los algoritmos mencionados, se debe asumir que la memoria está dividida en 2 conjuntos de bloques de tamaños variables:

a) Un conjunto de huecos: $H = \{H_i \mid i = 1, \ldots, n\}$, y

b) Un conjunto de bloques llenos: $F = \{F_i \mid i = 1, \ldots, m\}$.

- **Algoritmo "First-Fit":** ante la solicitud de un bloque de tamaño ***k***, recorre la memoria y asigna el primer hueco encontrado que cumple con la condición $h_i >= k$.
- **Algoritmo "Best-Fit":** entrega un hueco H_i tal que su tamaño es el mínimo posible que cumple con la condición $h_i >= k$.

76

Asignación de memoria en sistemas no paginados

Ventajas y Desventajas

Los métodos usados por los algoritmos vistos presentan las siguientes ventajas y desventajas.

	First-Fit	Best-Fit
Ventaja	Veloz	Eficiencia en la utilización de la memoria
Desventaja	Posible desperdicio de gran cantidad de memoria.	Puede resultar lento.

77

Modos de asignación de memoria principal

Asignación a Sistemas Paginados y No Paginados

¿Cuánto de memoria principal se asigna a un proceso cuando inicia su ejecución?

Existen 2 formas de asignar memoria a procesos en disco:

- **Asignación estática:** se asigna memoria física a todo el proceso al inicio de la ejecución del mismo.
- **Asignación dinámica:** se asigna la memoria física a una porción del proceso al inicio de la ejecución del mismo.
  En este caso se tiene que:
  - El resto del proceso permanece en memoria secundaria.
  - Durante la ejecución, distintas partes del proceso serán trasladadas entre memoria virtual y memoria física.

78

Modos de asignación de memoria principal

Asignación a Sistemas Paginados y No Paginados *(cont.)*

Ventajas y desventajas de las asignacˢ estática y dinámica

Asignación estática

- Es simple (ventaja)
- El espacio que utiliza el proceso debe ser menor que la memoria física disponible (desventaja).
- El grado de multiprogramación es bajo (desventaja).

Asignación dinámica

- Mejor uso del espacio de memoria primaria (ventaja)
- Mayor número de procesos activos en el sistema listos para hacer uso del procesador (mayor grado de multiprogramación) (ventaja).
- Mayor complejidad del algoritmo de asignación (desventaja).

79

Asignación dinámica de memoria en el tiempo

En sistemas Paginados y No Paginados

- El espacio que ocupan los procesos en el sistema, normalmente, es mucho mayor que el tamaño de la memoria física.
- Por ese motivo, cuando el sistema tiene muchos procesos activos es probable que la memoria principal esté casi completamente ocupada; por lo tanto, las solicitudes de espacios para páginas que están en disco no podrán ser satisfechas inmediatamente.
- En tal caso, un proceso que necesite estar en memoria principal, normalmente, entra en bloqueo hasta tanto el SO desocupe un espacio.

A continuación se verá: **Asignación Dinámica de Memoria en Sistemas Paginados**

80

Asignación dinámica en sistemas paginados

Esquema de Reemplazo de Páginas

- Es el método más complejo de implementar, pero también,
- Ofrece una mejor performance global al sistema.

Por esta segunda virtud, este método es utilizado por todos los SO´s modernos.

Situación típica en un sistema multiprogramado

- En cualquier momento, las páginas pertenecientes a distintos procesos están dispersas en distintos lugares de la memoria principal.
- Cuando ocurre un fallo de página - page fault - y no existe memoria física libre, el SO debe seleccionar una de las páginas residentes para su reemplazo.

81

Asignación dinámica en sistemas paginados

Esquema de Reemplazo de Páginas *(cont.)*

Se puede emplear alguna de las 2 estrategias de reemplazo:

1. **Estrategia local:** la página a ser reemplazada se selecciona del conjunto de páginas residentes del propio proceso que produce la falla.
2. **Estrategia global:** la página a reemplazarse puede pertenecer a cualquiera de los procesos en memoria principal.

82

Asignación dinámica en sistemas paginados

Estrategias

Cuando se debe traer desde disco a memoria física una página de un proceso, existen 2 tipos de estrategias:

- **Estrategia de emplazamiento o de ubicación**
  Determina **dónde** se debe cargar la página.
- **Estrategia de reemplazo**
  Determina **qué** página se debe eliminar de la memoria física cuando no hay espacio suficiente.

83

Algoritmos de reemplazo de páginas

Algoritmo de reemplazo más eficiente: es el que produce el menor número de fallos de página durante la ejecución de un programa. (Ver tablas a partir de pág. 273 del libro de Saade).

Se estudiarán los algoritmos considerando ejemplos, en los que:

1. Se supone que la memoria física sólo puede alojar 4 páginas que ya han sido cargadas cuando comienza la ejecución del programa.
2. **Z** representa la secuencia de las páginas referenciadas durante la ejecución del programa.
3. **a, b, …, k** son páginas virtuales referenciadas, donde **k** es la página de datos que se referencia.
4. Fallo: se marca con un asterisco **(*)** cuando ocurre un fallo de página.
5. **p** y **q** son la página cargada en memoria y la página reemplazada, respectivamente.

84

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo Óptimo

- Es el más eficiente, pues está basado en decidir el reemplazo de aquella página en memoria principal que no será accedida por mayor tiempo en el futuro.
- Es teórico. Lamentablemente, es de imposible implementación práctica, puesto que no se puede saber cuál será la página que no será accedida por mayor tiempo en el futuro.
- Sin embargo, sirve como referencia para evaluar algoritmos prácticos de reemplazo de páginas.

85

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo Óptimo

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a										
Página 1	b										
Página 2	c										
Página 3	d										
Fallo											
p											
q											

86

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo Aleatorio

- Como su nombre lo indica, se remueve cualquier página de memoria física.
- Es el esquema más simple de reemplazo. No obstante, no es eficiente porque no tiene en cuenta el

  Principio de cercanía o localidad de la referencia, que dice:

  cuando un programa referencia un lugar de la memoria, existe una alta probabilidad que referencie a la misma posición de memoria o a localidades cercanas en el futuro inmediato.

  Este principio tiene validez intuitiva si se considera que la ejecución de un programa sigue los patrones siguientes:

87

Algoritmos de reemplazo de páginas

Justificación del Principio de Localidad de la Referencia

- Excepto por las instrucciones de salto (del tipo jump que son entre un 10 y 20% del total), la ejecución de los programas es de tipo secuencial (sigue la secuencia de la ubicación de las instrucciones en memoria principal).
- La mayoría de las construcciones iterativas consisten de un número relativamente pequeño de instrucciones consecutivas repetidas un gran número de veces.
- La computación de estructuras de datos como arreglos, o secuencias de registros, constituyen la gran parte del código de un programa. Una porción significativa de esta computación requiere un procesamiento secuencial. De esta forma, instrucciones consecutivas tienden a referenciar elementos contiguos de las estructuras de datos.

88

Algoritmos de reemplazo de páginas

Algoritmo FIFO - First In, First Out -

Un puntero apunta a la página más antiguamente referenciada; en caso de fallo, se reemplaza la página apuntada y avanza el puntero a la siguiente página en orden cronológico.

Ventaja. Simplicidad de implementación: solamente se necesita una lista de ***m*** elementos y un puntero que apunta a la página más antigua en memoria física para su reemplazo. El puntero se incrementa cada vez que ocurre un fallo de página.

Desventaja. Supone que las páginas que residen en memoria por mayor tiempo son las que tienen menor probabilidad de que se referencien en el futuro. Esto no responde exactamente a lo establecido por el principio de localidad de referencia.

Puede ocurrir que algunas páginas que fueron cargadas en el pasado pueden volver a ser referenciadas con frecuencia, con lo que el "principio" puede ser violado repetidamente.

89

Algoritmos de reemplazo de páginas

Algoritmo FIFO (First In, First Out)

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a										
Página 1	b										
Página 2	c										
Página 3	d										
Fallo											
p (fifo)											
q (fifo)											

90

Algoritmos de reemplazo de páginas

Algoritmo LRU - Last Recently Used -

- Este algoritmo ha sido diseñado específicamente para cumplir con el principio de localidad de referencia.
- En este sentido, reemplaza la página que no ha sido referenciada por mayor tiempo en el pasado.
- La diferencia c/FIFO es que éste elimina la página que reside en memoria por mayor tiempo.
- **LRU** debe mantener una lista en la que las páginas se ordenan de acuerdo a la antigüedad de la referencia.

  Cuando ocurre un fallo, la página que se encuentra en la base de la lista (página referenciada en el pasado más lejano y todavía presente en memoria física) es reemplazada, luego la nueva página es ubicada al principio de la lista. (Ver ejemplo en libro)

91

Algoritmos de reemplazo de páginas

Algoritmo LRU (Last Recently Used)

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a										
Página 1	b										
Página 2	c										
Página 3	d										
Fallo											
p (lru)											
q (lru)											
Stack											

92

Algoritmos de reemplazo de páginas

Algoritmo LRU *(cont.)*

Se observa que se debe mantener un orden cronológico de las referencias a las páginas, por lo que la lista debe ser reordenada cada vez que se realiza una referencia.

Desventaja: La implementación de una lista por software es impráctica, por el tiempo que lleva reordenar la lista en cada referencia. No obstante, existen varios métodos alternativos por hardware para implementar este algoritmo.

93

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo de Reloj

- Este algoritmo trata de sustituir al algoritmo LRU cuya implementación por software es compleja e ineficiente.
- En el algoritmo de reemplazo de reloj se mantiene una lista circular de todas las páginas residentes y un puntero similar al utilizado en el algoritmo FIFO.
- Se adiciona un bit *u*, denominado bit de *uso*, asociado a cada página en memoria física y cuyo valor indica su antigüedad.
- Cuando se produce un fallo:
  - Si el puntero se encuentra posicionado en una página cuyo bit *u* es 0, entonces se reemplaza dicha página y se avanza el puntero.
  - En caso contrario (*u* = 1), el bit *u* se pone en 0, se avanza el puntero a la próxima página en la lista y se repite el mismo paso.

94

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo de Reloj

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a/1										
Página 1	b/1										
Página 2	c/1										
Página 3	d/1										
Fallo											
p											
q											

95

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo de Reloj (cont.)

Este algoritmo tiene un comportamiento aproximado al LRU, y su ventaja respecto de éste, es no se utiliza la lista donde se van ordenando las páginas, con lo que se optimiza el espacio utilizado y el tiempo que lleva el ordenamiento.

96

Estrategias de control de carga

Prepaginación

- **Un extremo:** en asignación estática, antes de ejecutar un programa son cargadas en memoria física todas las páginas del mismo.
  Consecuencia: pocos procesos cargados
- **Otro extremo:** Al inicio no se carga ninguna página. El programa comienza a ejecutar y las páginas se van cargando a medida que ocurren los fallos de página.
  Consecuencia: hasta que se estabiliza la ejecución del programa (varios fallos de página y varias páginas cargadas), hay mucha pérdida de tiempo en tareas de I/O de páginas.
- Conclusión: es necesario, entonces, buscar una solución intermedia entre los extremos; es decir, cargar en memoria física un número adecuado de páginas del programa antes de que comience a ejecutar. Esta acción se llama **prepaginación.**

97

Estrategias de control de carga

Prepaginación *(cont.)*

En la prepaginación se tienen las siguientes cuestiones a resolver:

- Proceso nuevo: es difícil tomar una decisión correcta de cuántas y qué páginas cargar inicialmente. (Ver siguiente diapositiva)
- Proceso viejo: una buena elección es cargar las páginas que tenía en memoria en el momento en que el proceso fue suspendido o bloqueado.

98

Estrategias de control de carga

Trashing

- Uno de los principales problemas a resolver en un esquema de memoria virtual es mantener un balance apropiado entre: **grado de multiprogramación** y **cantidad de prepaginado**

  Grado de multiprogramación: intervienen algoritmos de control de carga (long-term-schedulers) que determinan el número y tipos de procesos que debe ser admitidos para su ejecución.

  Cantidad de prepaginado:

1) Alto número de procesos activos→baja cantidad de prepaginado →muchos de fallos de página. Como consecuencia:
   El sistema tiende a destinar todo su tiempo sólo a mover páginas entre memoria virtual y memoria física, y viceversa.
   Este comportamiento se denomina **trashing** ← debe evitarse.
2) Alta cantidad de prepaginado→bajo grado de multiprogramación.

99

Evaluación de los sistemas de paginado

Método general para evaluar los sistemas de paginado:

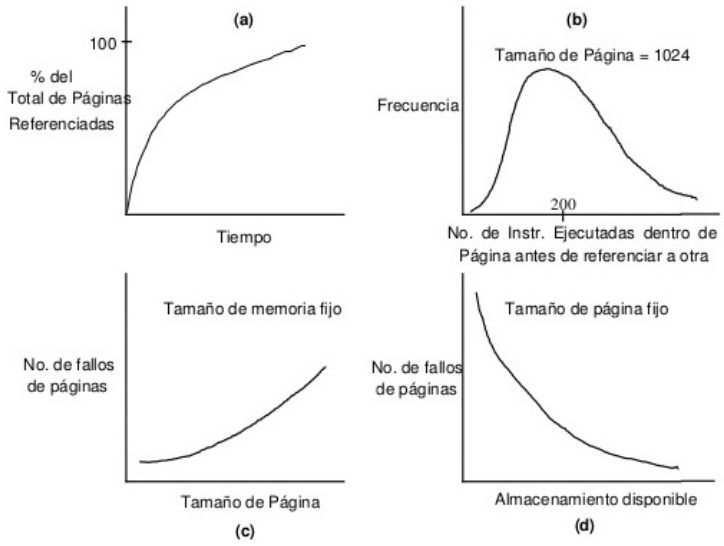
Se ejecuta una muestra de programas representativos para distintos mecanismos de paginado y variando ciertos parámetros tales como: el tamaño de la página o la cantidad de almacenamiento de páginas disponible en memoria física.

Conclusiones obtenidas:

1. La mayoría de los procesos requieren un alto porcentaje de sus páginas en un breve período, luego de su activación.
P/ejemplo: alrededor del 50% del total de las páginas de un proceso son referenciadas durante el primer quantum de tiempo de CPU, luego de la activación del proceso. [Parte (a) de la Figura 6-13] (Las figuras a que se hace referencia son del libro de Saade)

100

Evaluación de los sistemas de paginado



101

Evaluación de los sistemas de paginado

2. Un número relativamente pequeño de instrucciones son ejecutadas antes de que se referencie una instrucción en otra página. En la mayoría de las simulaciones, para páginas de 1024 palabras, fueron ejecutadas menos de 200 instrucciones antes de que otra página fuese referenciada. [Fig.(b)].
3. En el rango entre 64 y 1024 palabras por página y con espacio fijo de memoria, el número de fallos de páginas aumenta a medida que se incrementa el tamaño de la página. [Fig.(c)].
4. Cuando la cantidad de páginas en memoria (por proceso) decrece, para un tamaño de página fijo, existe un punto en que el número de fallos de página aumenta exponencialmente (trashing). [Fig.(d)].

102

Evaluación de los sistemas de paginado

Discusión sobre el Tamaño de Página más Adecuado

Tamaño Pequeño

Ventaja: menor desperdicio de memoria (fragmentación interna).

Desventaja: gran tamaño de las tablas de páginas y mucho movimiento de páginas.

Tamaño Grande

Ventajas:

- Tamaño reducido de las tablas de páginas.
- Mayor eficiencia en la transferencia de datos desde disco a memoria principal:

Eficiencia = cantidad de datos transferidos en la operación de E/S

Desventaja: mayor desperdicio de memoria (fragmentación interna)

103

Fin del Módulo IV

104

Sistemas Operativos I

Módulo V

ENTRADA/SALIDA
y
ARCHIVOS

Temas del Módulo V

- **Introducción.**
  - Conceptos generales.
  - Discos rígidos magnéticos.
- **Gestión de E/S.**
  - Dispositivos de E/S. Aspectos de diseño.
  - Estructura lógica del sistema de E/S.
- **Planificación del disco.**
  - Justificación. FIFO, SSTF, SCAN y C-SCAN.
  - Comparación de rendimientos.
- **Archivos.**
  - Descripción general.
  - Directorios.
  - Diseño del sistema de archivos.
  - Implementación de archivos.
  - Gestión del espacio libre.

Introducción

Conceptos generales.

- La E/S es quizás el aspecto más intrincado en el diseño de un SO, debido a la gran variedad de dispositivos y aplicaciones de éstos.
- Veremos en éste módulo cómo el SO debe gestionar, por un lado los dispositivos de E/S, en particular los de almacenamiento secundario, repasando cómo funciona un disco rígido magnético y sus parámetros de rendimiento.
- Por otro lado, veremos qué es un archivo y un sistema de archivos, de qué manera se organizan y almacenan los archivos, y cómo se gestiona el espacio libre en disco.

Introducción

Discos rígidos magnéticos.

- Son los dispositivos de almacenamiento secundario más populares y económicos. Consisten en uno o varios discos de metal o plástico, a los que se denomina platos, cubiertos con un material magnético.
- Los datos se leen y escriben con un dispositivo llamado cabezal, que se mantiene estacionario mientras el plato gira por debajo.
- El cabezal es, básicamente, una bobina conductora por la que circula electricidad y genera un campo magnético. Los datos en el plato se escriben magnetizando su superficie, emitiendo pulsos magnéticos a través del cabezal.
- La lectura de los datos se realiza “capturando” la magnetización de la superficie del plato con cabezal.

Introducción

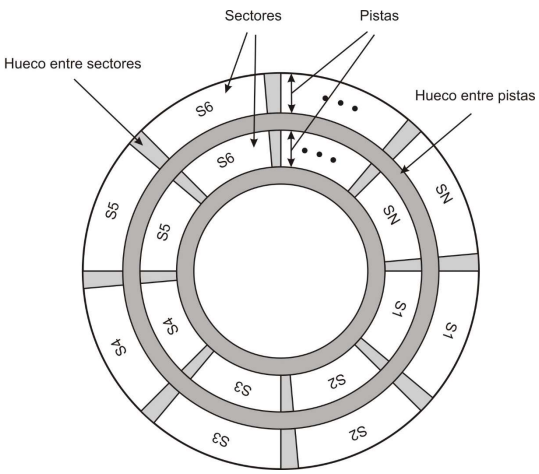
Discos rígidos magnéticos. *(cont.)*

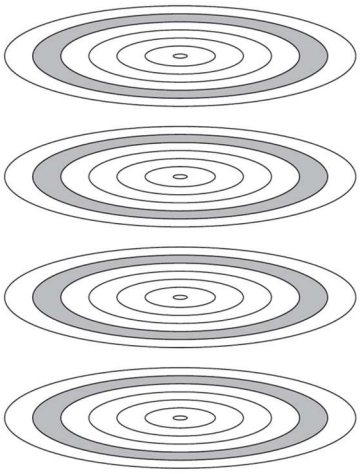
- **Organización y formato de los datos.**
- Los datos en el disco se organizan dividiendo su superficie en:
  - Pistas: es un conjunto de anillos concéntricos, del mismo ancho que el cabezal. Hay miles de pistas en cada superficie. Están separadas por huecos, que sirven para minimizar los errores de lectura/escritura.
  - Sectores: son subdivisiones de una pista. Es la unidad básica de datos, su longitud es normalmente fija, y tiene un tamaño típico de 512 Bytes, siendo éste el tamaño universal de sector. También están separados por huecos.
  - Cilindros: si el disco cuenta con más de un plato, un cilindro está formado por todas las pistas en la misma posición relativa.

Introducción

Discos rígidos magnéticos. *(cont.)*

- **Organización y formato de los datos.** *(cont.)*





Stallings, 2005

Introducción

Discos rígidos magnéticos. *(cont.)*

- **Parámetros de rendimiento.**
- A la hora de la gestionar el disco, el rendimiento del mismo se mide en términos de los siguientes parámetros:
  - Tiempo de búsqueda: es el que tarda el cabezal en posicionarse sobre una pista.
  - Latencia: también llamada *retardo rotacional*, es el tiempo que tarda en llegar el comienzo de un sector por debajo del cabezal. La suma del tiempo de búsqueda y la latencia es el **tiempo de acceso**.
  - Tiempo de transferencia: una vez posicionado el cabezal en la pista, los datos se leen o escriben mientras el sector pasa por debajo, y el tiempo requerido para esa lectura o escritura es el tiempo de transferencia.

Gestión de E/S

Dispositivos de E/S.

- Se pueden clasificar, a grandes rasgos, en tres categorías:
  - Legibles para el usuario: adecuados para la comunicación con el usuario, tales como teclados, monitores, mouse, impresoras, etc.
  - Legibles para la máquina: diseñados para la comunicación con equipamiento electrónico, tales como discos, cintas, sensores, etc.
  - Comunicación: utilizados en la comunicación con equipos remotos, tales como las placas de red, módems, etc.
- Hay grandes diferencias entre las distintas categorías:
  - Velocidad: pueden ser del orden de 10^1 bps como en los teclados, hasta 10^9 bps como en las placas Gigabit Ethernet.
  - Aplicación: un disco, además de almacenar archivos, también se utiliza para implementar esquemas de memoria virtual (Módulo IV). Otro ejemplo es una terminal, que puede ser utilizada por un usuario normal o un administrador.

Gestión de E/S

Dispositivos de E/S. *(cont.)*

- Hay grandes diferencias entre las distintas categorías: *(cont.)*
  - Complejidad de control: una impresora requiere una interfaz relativamente sencilla, mientras que un disco una mucho más compleja.
  - Unidad de transferencia: los datos pueden transferirse como un flujo bytes o caracteres (teclados, pantallas), o bien en bloques de varios bytes (discos).
  - Representación de datos: los dispositivos pueden utilizar distintos esquemas de codificación, por ejemplo, distintos códigos de caracteres (ASCII, UTF-8).
  - Condiciones de error: los tipos de errores, cómo se notifican, etc.
- Las técnicas básicas de comunicación de dispositivos de E/S son tres, y las planteamos en el Módulo Introductorio:
  - E/S programada / Polling.
  - Interrupciones.
  - DMA.

Gestión de E/S

Aspectos de diseño.

- Dos aspectos son muy importantes en el diseño del módulo de E/S de un SO:
  - Eficiencia: los dispositivos de E/S son muy lentos respecto de la memoria y el procesador, por lo que suelen ser un cuello de botella. Una manera de afrontar este problema es la multiprogramación, que como vimos, permite que algunos procesos esperen la finalización de operaciones de E/S mientras que la CPU ejecuta otro proceso.
  - Generalidad: para simplicidad y minimización de errores es deseable manejar todos los dispositivos de manera uniforme. En la práctica esto es muy difícil de lograr por la gran diversidad de características de los dispositivos. Lo que se hace es utilizar una estrategia modular jerárquica para diseñar las funciones de E/S, esto es, modelos de abstracción de varias capas para ocultar las distintas complejidades subyacentes.

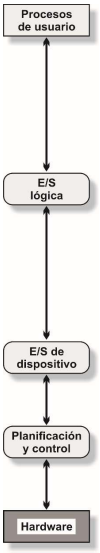
Gestión de E/S

Estructura lógica del sistema de E/S.

- El sistema de E/S se plantea como un modelo jerárquico, y si bien cada fabricante de SO tiene su propio diseño, en términos generales todos contemplan la siguiente organización en tres capas básicas:
  - E/S lógica: es la capa más cercana al usuario. Trata los dispositivos como un recurso lógico sin tener en cuenta los detalles de control real. Gestiona las tareas de E/S para los procesos de usuario en términos de ID de dispositivo, con operaciones sencillas: abrir, cerrar, leer y escribir.
  - E/S de dispositivo: las operaciones y los datos se convierten en instrucciones de E/S, es decir, se encarga de dar las órdenes específicas a los controladores de los dispositivos.
  - Planificación y control: realiza la gestión real de la cola y la planificación de las operaciones de E/S, como también el control de las operaciones. También maneja las IRQ y el estado de la E/S. Esta capa interactúa realmente con el hardware del dispositivo.

Gestión de E/S

Estructura lógica del sistema de E/S. *(cont.)*



Stallings, 2005

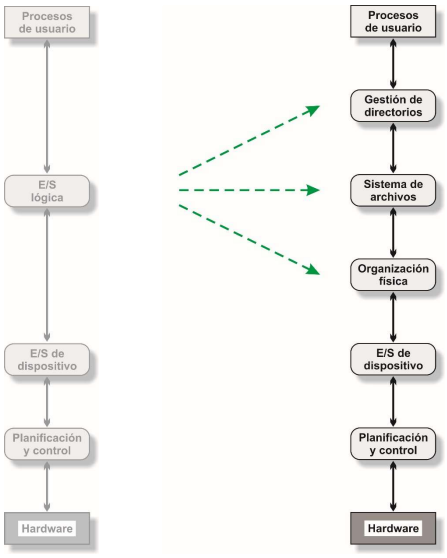
Gestión de E/S

Estructura lógica del sistema de E/S. *(cont.)*

- En el caso de los dispositivos de almacenamiento secundario que dan soporte a un sistema de archivos, la E/S lógica se divide en:
  - Gestión de directorios: en este nivel se convierten los nombres simbólicos en identificadores de archivos, que pueden referenciarlos directamente, o bien a través de un descriptor o una tabla de índices. También se gestiona las operaciones de usuario, tales como añadir, borrar y reorganizar.
  - Sistema de archivos: trata con la estructura lógica de los archivos y con las operaciones que pueden hacer los usuarios, como abrir, cerrar, leer y escribir. Los derechos de acceso también se gestionan en este nivel.
  - Organización física: se gestiona la traducción de las referencias lógicas a los archivos y registros, en direcciones físicas del almacenamiento secundario, teniendo en cuenta las pistas y los sectores. También se realiza la gestión de la asignación de espacio en disco.

Gestión de E/S

Estructura lógica del sistema de E/S. *(cont.)*



Stallings, 2005

Planificación del disco

Justificación.

- Como sabemos, los dispositivos de E/S son muy lentos respecto del procesador y la memoria. Por si fuera poco, en las últimas décadas esa brecha de velocidades pasó, aproximadamente, de dos a cuatro órdenes de magnitud.
- Si bien la multiprogramación es un primer paliativo, para mejorar la performance se hace necesario optimizar las operaciones de E/S a fin de minimizar los tiempos de acceso al disco.
- En este sentido, los parámetros de rendimiento del disco vistos son la herramienta fundamental para el diseño de las técnicas de planificación del disco.
- Ejemplos de estas optimizaciones serían minimizar la cantidad de movimientos del cabezal y acceder a los sectores según su cercanía.

Planificación del disco

FIFO (First In-First Out).

- Es la planificación más sencilla, donde se procesan las peticiones de manera secuencial.
- Tiene la ventaja de ser equitativa porque todas las peticiones serán atendidas, siguiendo el orden en que fueron recibidas.
- Si son pocos los accesos a disco, y además las peticiones se corresponden con sectores agrupados de archivos, se puede prever un buen rendimiento.
- Sin embargo, esta técnica tiene en general un rendimiento muy pobre, sobre todo cuando la cola de peticiones es muy grande, fundamentalmente porque no optimiza la cantidad de movimientos del cabezal.

Planificación del disco

SSTF (Shortest Service Time First).

- Consiste en seleccionar la petición de E/S del disco que requiera un menor movimiento del cabezal desde su posición actual. Entonces, siempre se realiza una selección de manera que se produzca un tiempo de búsqueda mínimo.
- Seleccionar siempre el tiempo de búsqueda mínimo no garantiza que sea mínimo el tiempo de búsqueda medio correspondiente a varios movimientos del cabezal. Sin embargo, este esquema proporciona un rendimiento superior al algoritmo FIFO.
- Como el cabezal puede moverse en dos direcciones, se puede utilizar un algoritmo aleatorio para resolver aquellos casos en que la distancia a recorrer hacia ambos lados sea igual.
- Una desventaja importante es que cabe la posibilidad de tener inanición entre las peticiones que requieren más tiempo.

Planificación del disco

SCAN.

- Impide la inanición anterior. Con este algoritmo, el cabezal sólo debe moverse en una dirección, satisfaciendo las peticiones que va encontrando en su camino hasta que llega a la última pista.
- Una mejora, llamada política LOOK, es no llegar hasta la última pista, sino hasta que no haya más peticiones en esa dirección.
- Una vez completadas las peticiones en una dirección, la búsqueda y servicio de las peticiones continúa en la dirección opuesta.
- Este algoritmo se comporta prácticamente igual a SSTF, aún cuando la cola de peticiones cambia dinámicamente.
- La desventaja de este algoritmo es que no aprovecha la proximidad de sectores tanto como SSTF, y puede provocar retardos a las peticiones más recientes.

Planificación del disco

C-SCAN.

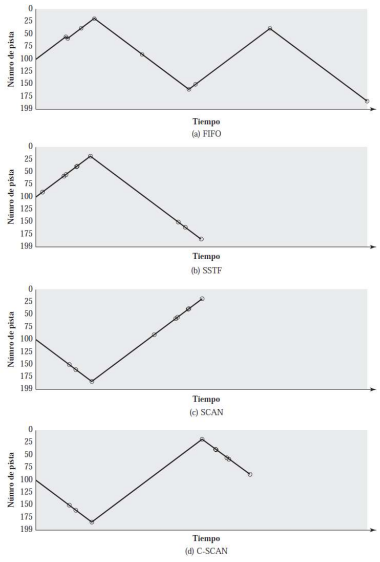
- La política C-SCAN restringe la búsqueda en una sola dirección; después de visitar la última pista, el cabezal vuelve al extremo opuesto del disco y la búsqueda comienza de nuevo.
- Esto reduce el retardo máximo que pueden experimentar las nuevas peticiones. Por otro lado, también reduce el tiempo de espera de las pistas recientemente visitadas.
- Este algoritmo no tiene tan buen rendimiento como SCAN, ya que tiene en promedio más movimientos del cabezal, lo que se puede mejorar utilizando también la política LOOK.

Planificación del disco

Comparación de rendimientos.

Secuencia de pistas solicitadas:
55, 58, 39, 18, 90, 160, 150, 38, 184

Pista inicial: 100



Stallings, 2005

Archivos

Descripción general.

- Todas las aplicaciones requieren almacenar y recuperar información. Mientras un proceso está en ejecución, puede almacenar una cantidad limitada de información dentro de su espacio de direcciones, en memoria principal.
- Un segundo problema es que al almacenar información dentro de su espacio de direcciones, cuando el proceso termina, ésta se pierde, lo que es inaceptable en la gran mayoría de las aplicaciones, como es el caso de las bases de datos.
- Un tercer problema es que suele ser necesario que varios procesos accedan a (partes de) la información al mismo tiempo; esto se resuelve haciendo que la información en sí sea independiente de cualquier proceso.

Archivos

Descripción general. *(cont.)*

- En consecuencia, tenemos tres requerimientos esenciales para el almacenamiento de información a largo plazo:
  - Debe ser posible almacenar una cantidad muy grande de información.
  - La información debe sobrevivir a la terminación del proceso que la utilice.
  - Múltiples procesos deben ser capaces de acceder a la información concurrentemente.
- Así como el SO provee las abstracciones de proceso y memoria virtual, también presenta una abstracción para el almacenamiento: los **archivos**.
- Los archivos son unidades lógicas de información creada por los procesos. La información que se almacena debe ser persistente, es decir, no debe ser afectada por la creación y terminación de los procesos. La parte del SO que los administra se conoce como **sistema de archivos**.

Archivos

Descripción general. *(cont.)*

- Los archivos son un mecanismo de abstracción para proteger al usuario de los detalles acerca de cómo y dónde se almacena y lee la información, y de cómo funcionan los discos en realidad.
- En esta sección, primero analizaremos los archivos desde el punto de vista del usuario, esto es, cómo se utilizan y que propiedades tienen.
- **Nomenclatura de archivos.**
- Cuando un proceso crea un archivo, le otorga un nombre.
- Las reglas exactas para denominar archivos varían de un sistema a otro, pero todos los SO actuales permiten cadenas de una a ocho letras como nombres de archivos legales.
- Algunos sistemas distinguen mayúsculas de minúsculas, y otros no; ejemplos de los primeros, típicamente los sistemas compatibles con POSIX, y de los segundos, MS-DOS, Windows, etc.

Archivos

Descripción general. *(cont.)*

- **Nomenclatura de archivos.** *(cont.)*
- Muchos sistemas operativos aceptan nombres de archivos en dos partes, separadas por un punto. La parte que va después del punto se conoce como la extensión del archivo y por lo general indica algo acerca de su naturaleza.
- En algunos sistemas, como los basados en POSIX, la extensión es sólo una convención y no son impuestas por el SO. Sin embargo, las aplicaciones pueden requerirlas, como el compilador *gcc* exige que el código fuente tenga la extensión *.c* para programas en C.
- Por el contrario, Windows está consciente de las extensiones y les da un significado. Lo usuarios/procesos pueden registrar extensiones en el SO y especificar qué aplicación “posee” esa extensión.

Archivos

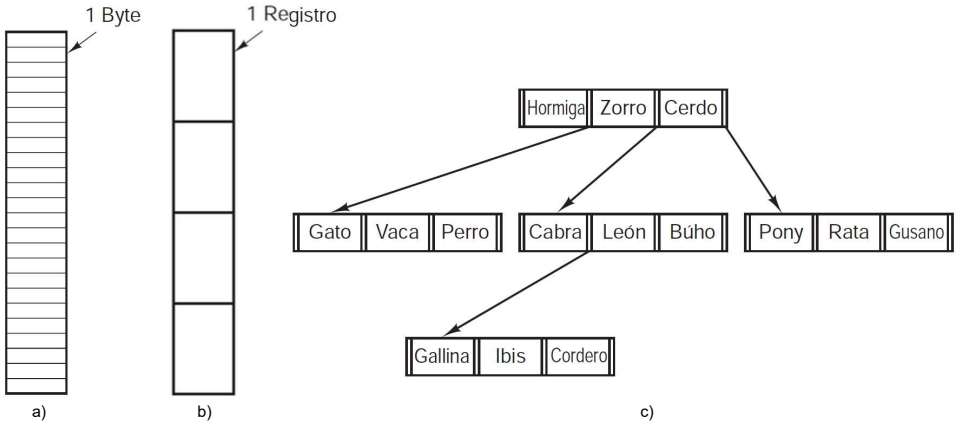
Descripción general. *(cont.)*

- **Estructura de archivos.**
- Existen básicamente tres maneras de estructurar los archivos:
  - Secuencia de bytes: en sí, no hay estructura, el SO no sabe, ni le importa, qué hay en el archivo. Todo lo que ve son bytes. Cualquier significado debe ser impuesto por los programas a nivel usuario. Tanto POSIX como Windows utilizan esta metodología.
  - Secuencia de registros: en este modelo se tiene una secuencia de registros de longitud fija, cada uno con una cierta estructura interna. Este modelo dejó de utilizarse en sistemas de propósito general.
  - Árbol: en esta estructura, el archivo es un árbol de registros, no necesariamente todos de la misma longitud. Cada registro contiene una *clave*, la cual se utiliza para ordenar el archivo y permitir una búsqueda rápida según este campo. Este tipo de estructura es muy utilizada en los SO de tipo mainframe.

Archivos

Descripción general. *(cont.)*

- **Estructura de archivos.** *(cont.)*



a) Secuencia de bytes. b) Secuencia de registros. c) Árbol.

Tanenbaum, 2009

Archivos

Descripción general. *(cont.)*

- **Acceso a archivos.**
- Hay dos modos elementales para acceder a los archivos:
  - Acceso secuencial: es el más antiguo; los bytes o registros son leídos en orden, empezando desde el principio, y no es posible saltear o leer fuera de orden.
  - Acceso aleatorio: es posible leer los bytes o registros fuera de orden, cualquiera sea la posición del byte o registro.
- En la actualidad los archivos se pueden acceder combinando los dos tipos de acceso.
- Una operación especifica la posición donde se debe empezar a leer, y otra operación establece al posición actual dentro del archivo, para luego realizar lecturas secuenciales.
- Estos métodos son utilizados tanto en POSIX como en Windows.

Archivos

Descripción general. *(cont.)*

- **Atributos de archivos.**
- Todo archivo tiene un nombre y sus datos. Además, todos los SO asocian otra información con cada archivo. A estos elementos adicionales les llamaremos atributos del archivo.
- Ejemplos de atributos de un archivo son:
  - Protección: quién puede acceder al archivo y de qué forma, contraseña, creador, propietario.
  - Banderas: sólo lectura, oculto, del sistema, ASCII/binario, etc.
  - Longitudes: cantidad de bytes por registro, del campo clave.
  - Fechas y horas: de creación, último acceso, última modificación.
  - Tamaños: actual, máximo.

Archivos

Descripción general. *(cont.)*

- **Operaciones de archivos.**
- Las operaciones de archivos están relacionadas con las system calls; las más comunes implementadas por la mayoría de los SO son:
  - Create: crea un archivo sin datos.
  - Delete: elimina el archivo y libera espacio en el almacenamiento.
  - Open: abre un archivo, llevando a memoria los atributos y la lista de direcciones.
  - Close: cierra el archivo, eliminando sus datos de acceso de la tabla en memoria.
  - Read: lee datos, especificando la cantidad de datos a leer y desde qué posición.
  - Write: escribe datos, especificando desde qué posición.
  - Append: similar a write, pero escribe los datos siempre al final del archivo.
  - Seek: posiciona el puntero al archivo, respecto del inicio, final o la posición actual.
  - Get attributes: lee los atributos de un archivo.
  - Set attributes: modifica los atributos de un archivo.
  - Rename: cambia el nombre del archivo.

Archivos

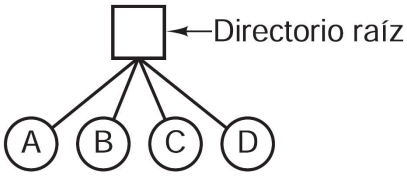
Directorios.

- Para llevar el registro de los archivos, los sistemas de archivos por lo general tienen directorios o carpetas.
- En muchos sistemas operativos, típicamente en aquellos basados en POSIX, los directorios son tratados también como archivos, y al igual que éstos, tienen operaciones implementadas como llamadas al sistema.
- En esta sección hablaremos sobre los directorios, su organización, sus propiedades y las operaciones que pueden realizarse con ellos.

Archivos

Directorios. *(cont.)*

- **Directorios de un solo nivel.**
- La forma más simple de un sistema de directorios es tener un único directorio que contenga todos los archivos.
- En las primeras computadoras personales, este sistema era común, en parte debido a que sólo había un usuario.
- Las ventajas de este esquema son su simpleza y la rapidez para localizar archivos. Se utiliza típicamente en sistemas embebidos.

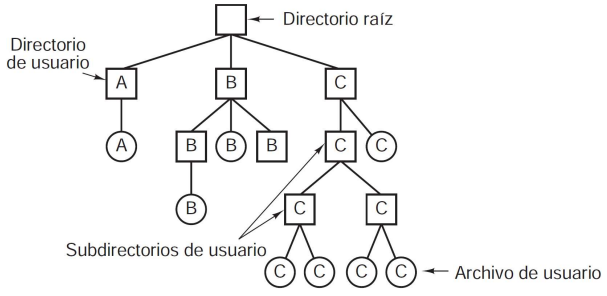


Tanenbaum, 2009

Archivos

Directorios. *(cont.)*

- **Directorios jerárquicos.**
- Con este esquema, puede haber tantos directorios como se necesite para agrupar los archivos en formas naturales.
- Además, si varios usuarios comparten un servidor de archivos, cada usuario puede tener un directorio raíz privado con su propia jerarquía. Casi todos los sistemas modernos se organizan de esta manera.



Tanenbaum, 2009

Archivos

Directorios. *(cont.)*

- **Nombre de rutas.**
- Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos. Por lo general se utilizan dos métodos distintos.
- En el primer método, cada archivo recibe un nombre de ruta absoluto que consiste en la ruta desde el directorio raíz al archivo.
- Por ejemplo, la ruta */usr/ast/mailbox* significa que el directorio raíz contiene un subdirectorio llamado *usr*, que a su vez contiene un subdirectorio *ast*, el cual contiene el archivo *mailbox*.
- Los nombres de ruta absolutos siempre empiezan en el directorio raíz y son únicos. En POSIX, los componentes de la ruta van separados por /; en Windows el separador es \.

Archivos

Directorios. *(cont.)*

- **Nombre de rutas.** *(cont.)*
- El otro tipo de nombre es el nombre de ruta relativa. Éste se utiliza en conjunto con el concepto del directorio de trabajo (también llamado directorio actual).
- Un usuario designa un directorio como el directorio de trabajo actual, y todos los nombres de las rutas que no empiecen en el directorio raíz se toman en forma relativa al directorio de trabajo.
- Por ejemplo, si el directorio de trabajo actual es */usr/ast*, entonces el archivo cuya ruta absoluta sea */usr/ast/mailbox* se puede referenciar simplemente como *mailbox*.

Archivos

Directorios. *(cont.)*

- **Operaciones de directorios.**
- Las operaciones de directorio difieren mucho más que las de archivos de un SO a otro; a modo de ejemplo, veremos algunas en POSIX:
  - Create: crea un directorio vacío.
  - Delete: elimina el directorio; sólo se puede eliminar un directorio vacío.
  - Opendir: abre un directorio, por ejemplo, para listar su contenido.
  - Closedir: cierra el directorio, y libera el espacio en la tabla interna.
  - Readdir: devuelve la siguiente entrada en un directorio abierto.
  - Rename: cambia el nombre del directorio.
  - Link: permite que un mismo archivo aparezca en más de un directorio.
  - Unlink: elimina una entrada en un directorio; si está presente en más de uno, solo se elimina del directorio actual.

Archivos

Diseño del sistema de archivos.

- Cambiamos ahora el punto de vista del usuario sobre el sistema de archivos, al punto de vista del que lo implementa.
- Los sistemas de archivos se almacenan en discos. Éstos se pueden dividir en una o más particiones, cada una con sistemas de archivos independientes.
- El sector 0 del disco se denomina MBR (Master Boot Record, registro maestro de arranque), y se utiliza para arrancar la computadora.
- El final del MBR contiene la **tabla de particiones**, la cual proporciona las direcciones de inicio y fin de cada partición.
- Una de las particiones en la tabla se marca como activa. Cuando se arranca la computadora, el BIOS lee y ejecuta el MBR.

Archivos

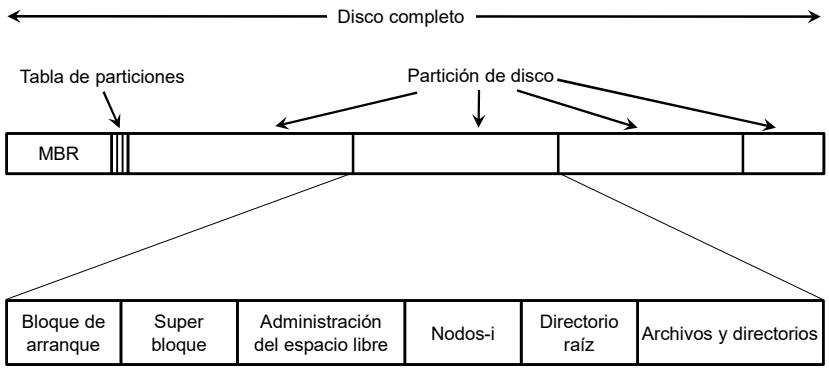
Diseño del sistema de archivos. *(cont.)*

- Lo primero que hace el programa MBR es localizar la partición activa, leer su primer bloque, conocido como **bloque de arranque**, y ejecutarlo. El programa en el bloque de arranque carga el sistema operativo contenido en esa partición.
- La distribución de una partición de disco varía mucho de un sistema de archivos a otro. A menudo contendrá algunos de estos elementos:
  - Superbloque: se lleva a memoria cuando la computadora inicia, y contiene información para identificar el tipo de sistema de archivo, el número de bloques que contiene, etc.
  - Administración del espacio libre: información acerca del espacio libre en el sistema de archivos, con alguna de las técnicas que veremos más adelante.
  - Nodos-i: arreglo de estructuras de datos, uno por archivo, con la información completa del archivo.

Archivos

Diseño del sistema de archivos. *(cont.)*

- Después de esto, podría venir el directorio raíz, y por último, el resto del disco contiene todos los otros directorios y archivos.



Tanenbaum, 2009

Archivos

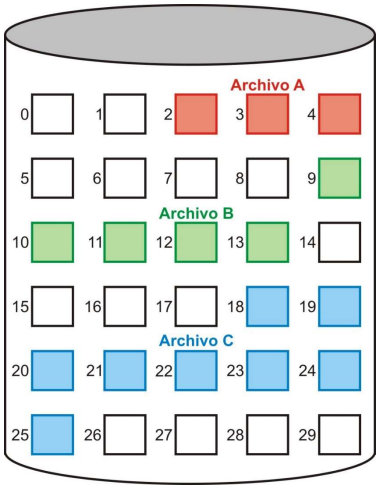
Implementación de archivos.

- Tal vez la cuestión más importante al implementar el almacenamiento de archivos sea mantener un registro acerca de qué bloques de disco van con cuál archivo.
- Se utilizan varios métodos en distintos SO.
- **Asignación contigua.**
- El esquema de asignación más simple es almacenar cada archivo como una serie contigua de bloques de disco.
- Así, en un disco con bloques de 1 KB, a un archivo de 50 KB se le asignarían 50 bloques consecutivos. Con bloques de 2 KB, se le asignarían 25 bloques consecutivos.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación contigua.** *(cont.)*



Entradas en el directorio

Nombre de archivo	Bloque inicial	Longitud
Archivo A	2	3
Archivo B	9	5
Archivo C	18	8

Archivos

Implementación de archivos. *(cont.)*

- **Asignación contigua.** *(cont.)*
- Este método tiene dos ventajas significativas:
- En primer lugar es simple de implementar, ya que llevar un registro de la ubicación de los bloques de un archivo se reduce a recordar dos números: la dirección de disco del primer bloque y el número de bloques en el archivo. Dado el número del primer bloque, se puede encontrar el número de cualquier otro bloque con una simple suma.
- En segundo lugar, el rendimiento de lectura es excelente; el archivo completo se puede leer del disco en una sola operación. Sólo se necesita una búsqueda (para el primer bloque); después de eso, no son necesarias más búsquedas ni retrasos por rotación. Por ende, la asignación contigua tiene un alto rendimiento.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación contigua.** *(cont.)*
- La desventaja de este método es ligeramente significante: con el transcurso del tiempo, los discos se fragmentan, tal como vimos que pasa con la memoria principal.
- Cuando se quita un archivo, sus bloques se liberan naturalmente, dejando una serie de bloques libres en el disco.
- El disco no se compacta al momento para quitar el hueco, ya que eso implicaría tener que copiar todos los bloques que van después del hueco, que podrían ser millones.
- Como resultado, el disco al final consiste de archivos y huecos, difícil de mantener, por lo que esta forma de asignación se dejó de utilizar en discos magnéticos, pero es utilizada en discos ópticos (CD, DVD).

Archivos

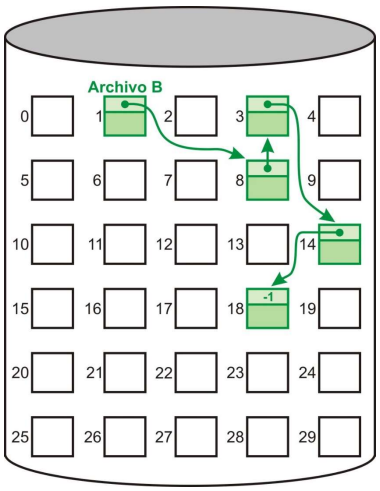
Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada.**
- Con este método, un archivo es una lista enlazada de bloques en el disco. La primera palabra de cada bloque se utiliza como puntero al siguiente; el resto del bloque es para los datos. En el último bloque, la primera palabra contendrá un valor numérico que indicará que es el último (cero o negativo).
- Con este método se puede utilizar cada bloque del disco. No se pierde espacio debido a la fragmentación del disco (excepto por la fragmentación interna en el último bloque).
- Para la entrada del directorio se almacena la dirección de disco del primer bloque; el resto se puede encontrar a partir de ella.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada.** *(cont.)*



Entradas en el directorio

Nombre de archivo	Bloque inicial
…	…
Archivo B	1
…	…

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada.** *(cont.)*
- Si bien la lectura secuencial de un archivo es directa, el acceso aleatorio es en extremo lento. Para llegar al bloque *n*, el SO tiene que empezar desde el principio y leer los *n* – 1 bloques anteriores, uno a la vez.
- Por otro lado, la cantidad de datos en un bloque ya no es una potencia de dos, debido a que el puntero ocupa unos cuantos bytes. Aunque no es fatal, tener un tamaño peculiar es menos eficiente debido a que muchos programas leen y escriben en bloques, cuyo tamaño es una potencia de dos.
- Además, no hay principio de proximidad de sectores, lo que provoca que leer varios bloques cuesta acceder a diferentes partes del disco, lo que resulta en una operación de lectura de muy bajo rendimiento.

Archivos

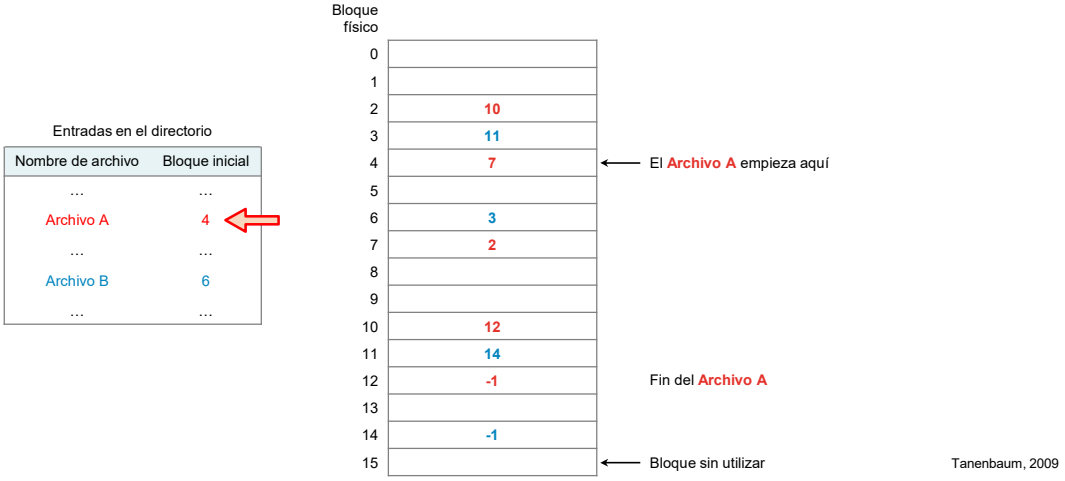
Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.**
- Ambas desventajas de la asignación de lista enlazada se pueden eliminar si tomamos la palabra del puntero de cada bloque de disco y la colocamos en una tabla en memoria principal.
- En la entrada del directorio se almacena la dirección del primer bloque, que coincide con el número de entrada en la tabla, y cada entrada en la tabla contiene la dirección del próximo bloque.
- El último bloque se indica con un valor especial en esa entrada de la tabla, que será un número de bloque no válido, por ejemplo, -1.
- De esta manera, un archivo se recorre siguiendo los valores de cada entrada de la tabla, hasta encontrar el indicador de fin del archivo.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.** *(cont.)*



Tanenbaum, 2009

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.** *(cont.)*
- Esta tabla en memoria principal se conoce como FAT (File Allocation Table, tabla de asignación de archivos). Con esta organización, el bloque completo está disponible para los datos.
- El acceso aleatorio es mucho más sencillo, y si bien aún se tiene que seguir la lista, al estar en memoria es mucho más rápido al no tener que hacer las sucesivas referencias a disco.
- Al igual que el método anterior, la entrada de directorio necesita sólo un entero (el número de bloque inicial) y aún así se puede localizar todos los bloques, sin importar qué tan grande sea el archivo.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación de lista enlazada con tabla en memoria.** *(cont.)*
- La principal desventaja de este método es que es necesario que la tabla completa esté en memoria todo el tiempo para que funcione.
- Con un disco de 200 GB y un tamaño de bloque de 1 KB, la tabla necesita 200 millones de entradas, una para cada uno de los 200 millones de bloques de disco.
- Con 4 bytes por entrada, el tamaño ocupado por la tabla ronda los 800 MB de memoria, lo cual no es tan práctico.
- Esto hace inviable esta metodología para discos grandes.

Archivos

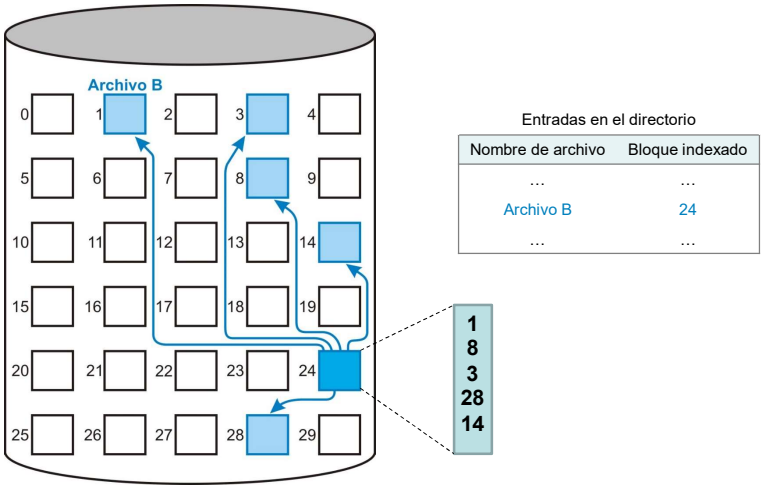
Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).**
- Cada archivo tiene asociada una estructura de datos, llamada nodo-i (nodo-índice), la cual lista los atributos y las direcciones de disco de los bloques del archivo.
- Esta estructura se almacena en el primer bloque del archivo, por lo que la entrada del directorio sólo necesita ésta dirección.
- Los bloques del archivo se pueden acceder entonces a partir de las direcciones almacenadas en el nodo-i.
- La gran ventaja de este esquema, en comparación con los archivos enlazados que utilizan una tabla en memoria, es que el nodo-i sólo necesita estar en memoria cuando está abierto el archivo.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).** *(cont.)*



Archivos

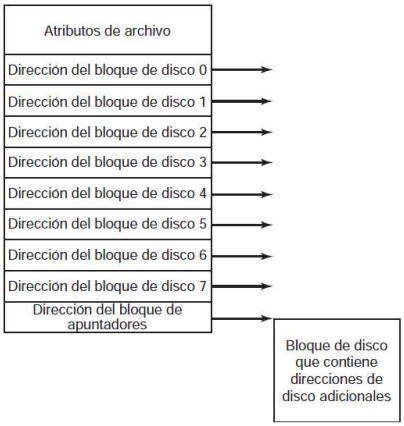
Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).** *(cont.)*
- Un problema con los nodos-i es que éstos tienen un número fijo de direcciones de disco (limitado por el tamaño del bloque), por lo tanto, el archivo tendrá una cantidad acotada de bloques.
- Si el archivo tiene que crecer más allá de esa cantidad de bloques, la solución es reservar la última dirección, no para un bloque de datos, sino para un bloque que contiene más direcciones de disco.
- A su vez, esos bloques pueden apuntar a otros bloques con más direcciones, permitiendo que el archivo pueda tener tamaños más grandes.
- Esta metodología es ampliamente utilizada en los sistemas POSIX.

Archivos

Implementación de archivos. *(cont.)*

- **Asignación indexada (Nodos-i).** *(cont.)*



Tanenbaum, 2009

Archivos

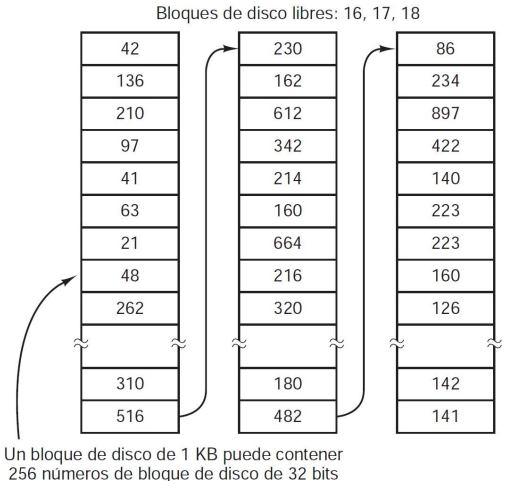
Gestión del espacio libre.

- Hay dos métodos ampliamente utilizados para la gestión del espacio libre en disco: lista enlazada de bloques libres, y mapa de bits.
- **Lista enlazada de bloques libres.**
- Cada bloque contiene tantos números de bloque de disco libres como pueda, dejando libre una palabra dentro del bloque para un puntero al siguiente bloque.
- Con un bloque de 1 KB y un número de bloque de disco de 32 bits, cada bloque en la lista contiene los números de 255 bloques libres.
- Con un disco de 500 GB, unos 488 millones de bloques, se requiere una cantidad aproximada de 1.9 millones de bloques.
- En general se utilizan bloques libres para mantener esta lista, por lo que en esencia su almacenamiento es gratuito.

Archivos

Gestión del espacio libre. *(cont.)*

- **Lista enlazada de bloques libres.** *(cont.)*



Tanenbaum, 2009

Archivos

Gestión del espacio libre. *(cont.)*

- **Lista enlazada de bloques libres.** *(cont.)*
- Este método requiere un solo bloque de punteros en memoria. Al crear un archivo, los bloques necesarios se toman de este bloque de punteros. Cuando se agotan, se lee del disco un nuevo bloque de punteros.
- Una desventaja es que, en ciertas circunstancias, se pueden producir operaciones de E/S innecesarias. Por ejemplo, cuando el bloque de punteros tiene dos direcciones disponibles y se elimina un archivo de tres bloques.
- Entonces el bloque de punteros se desborda, se escribe en disco y se lee uno a memoria. Si inmediatamente se crea un archivo nuevo de tres bloques, se vuelve a la situación anterior, y si ese archivo es temporal, se repiten nuevamente todas las operaciones de E/S.

Archivos

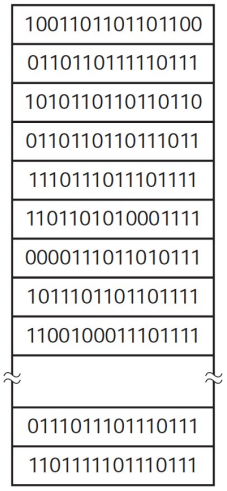
Gestión del espacio libre. *(cont.)*

- **Mapa de bits.**
- Un disco de *n* bloques requiere un mapa de *n* bits. Los bloques libres se representan con un 1, y los ocupados con un 0 (o viceversa).
- Con el disco de 500 GB de ejemplo, necesitamos 488 millones de bits para el mapa, que requieren poco menos de 60.000 bloques, apenas un poco más del 3% requerido por la lista enlazada.
- Con un mapa de bits también es posible mantener sólo un bloque en memoria, usando el disco para obtener otro sólo cuando el primero se llena o se vacía.

Archivos

Gestión del espacio libre. *(cont.)*

- **Mapa de bits.** *(cont.)*



Tanenbaum, 2009

Archivos

Gestión del espacio libre. *(cont.)*

- **Mapa de bits.** *(cont.)*
- Otra ventaja de este método es que al realizar toda la asignación de un solo bloque del mapa de bits, los bloques de disco estarán cerca uno del otro, con lo cual se minimiza el movimiento del cabezal.
- Como el mapa de bits es una estructura de datos de tamaño fijo, si el kernel está paginado, el mapa de bits puede colocarse en memoria virtual y hacer que se paginen sus páginas según se requiera.
- Esto soluciona una posible desventaja, que es que ocupe más lugar en memoria de lo que realmente es necesario.

Fin del Módulo V