

Resumen Introducción a los Sistemas Operativos

Sistema Operativo

Un sistema operativo es un software que actúa como intermediario entre el usuario, una computadora y su hardware

Es software

- Necesita un procesador y memoria para ejecutarse

Cosas que hace:

- Gestiona el HW
- Controla la ejecución de los procesos
- Interfaz entre aplicaciones y HW
- Actúa como intermediario entre un usuario de una computadora y el HW de la misma

Dos perspectivas:

- Desde el usuario
- Desde el sistema

Perspectiva del usuario:

- Abstracción con respecto a la arquitectura
 - Arquitectura: conjunto de instrucciones, organización de memoria, E/S,, estructura del bus
- El S.O “oculta” el HW y presenta a los programas abstracciones más simples de manejar
- Los programas de aplicación son los “clientes” del S.O

- Comparación: uso de escritorio y uso de comando de texto

Perspectiva desde la administración de recursos:

- Administra los recursos de HW de uno o mas procesos
- Provee un conjunto de servicios a los usuarios del sistema
- Maneja la memoria secundaria y dispositivos de I/O(input/output-entrada/salida)
- Ejecución simultánea de procesos
- Multiplexación de tiempo(CPU) y espacio(memoria)

Objetivos del S.O:

- Comodidad
 - Hacer más fácil el uso del hardware
- Eficiencia
 - Hacer un uso más eficiente del sistema
- Evolución
 - Permitir la introducción de nuevas funciones al sistema sin interferir con funciones anteriores

Componentes de un S.O:

- Kernel
- Shell
- Herramientas
 - Editores, compiladores, librerías, etc.

Kernel (Núcleo):

- “Porción de código”
 - Que se encuentra en memoria principal
 - Que se encarga de la administración de los recursos
- Implementa servicios:
 - Manejo de memoria

- Manejo de CPU
- Administración de procesos
- Comunicación y concurrencia
- Gestión de E/S

Servicios del S.O:

- Administración y planificación del procesador
 - Multiplexación de la carga de trabajo
 - Imparcialidad, "justicia" en la ejecución
 - Que no haya bloqueos
 - Manejo de prioridades
- Administración de memoria
 - Administración de memoria eficientemente
 - Memoria física vs memoria virtual. Jerarquías de memoria
 - Protección de programas que compiten o se ejecutan concurrentemente
- Administración del almacenamiento-Sistema de archivos
 - Acceso de medios de almacenamiento externos
- Administración de dispositivos
 - Ocultamiento de dependencias de HW
 - Administración de accesos simultáneos
- Detección de errores y respuestas
 - Errores de HW internos y Externos
 - Errores de memoria/cpu
 - Errores de dispositivos
 - Errores de SW
 - Errores Aritméticos
 - Acceso no permitido a direcciones de memoria
 - Incapacidad del S.O para conceder una solicitud de una aplicación

- Interacción del Usuario(shell)
- Contabilidad
 - Recoger estadísticas del uso
 - Monitorear parámetros de rendimiento
 - Anticipar necesidades de mejoras futuras
 - Dar elementos si es necesario facturar tiempo de procesamiento

Complejidad

- Un S.O es un software extenso y complejo
- Es desarrollado por partes
- Cada una de estas partes deben ser analizadas y desarrolladas entendiendo su función, cuáles son sus entradas y sus salidas

Funciones principales de un S.O

- Brindar abstracciones de alto nivel a los procesos de usuario
- Administrar eficientemente el uso de:
 - La CPU
 - La memoria
 - Otros dispositivos
- Brindar asistencia para la realización de Entrada-salida por parte de los procesos

Problemas que un S.O debe evitar

- Que un proceso se apropie de la CPU
- Que un proceso intente ejecutar instrucciones "importantes"
 - E/s - flags del procesador
- Que un proceso intente acceder a una posición de memoria fuera de su espacio declarado

- Proteger los espacios de direcciones
 - Gestionando el uso de la cpu
 - Detectando intentos de ejecución de instrucciones de E/S ilegales
 - Detectar accesos ilegales a memoria
 - Proteger el vector de interrupciones

Apoyo del hardware

- Modos de ejecución: Define limitaciones en el conjunto de instrucciones que se puede ejecutar en cada modo
- Interrupción de Clock: Se debe evitar que un proceso se apropie de la CPU
- Protección de la Memoria: Se deben definir límites de memoria a los que puede acceder cada proceso(registros base y límite)

Modos de ejecución:

- El bit en la CPU indica el modo actual
- Las instrucciones en modo Supervisor o Kernel
 - Necesitan acceder a estructuras del Kernel, o ejecutar código que no es del proceso
- En modo Usuario, el proceso puede acceder sólo a su espacio de direcciones, es decir a las direcciones “propias”
- El Kernel del S.O se ejecuta en modo supervisor
- El resto del S.O y los programas de usuario se ejecutan en modo usuario

Tener en cuenta que...

- Cuando se arranque el sistema, arranca con el bit en modo supervisor
- Cada vez que comienza a ejecutarse un proceso de usuario, este bit se DEBE PONER en modo usuario
 - Mediante una instrucción especial

- Cuando hay un trap o una interrupción, el bit de modo se pone en modo Kernel
 - Única forma de pasar a modo Kernel
 - No es el proceso de usuario quien hace el cambio explícitamente

Cómo actúa...

- Cuando el proceso de usuario intenta por sí mismo ejecutar instrucciones que pueden causar problemas (llamadas instructivas privilegiadas), el HW lo detecta como una operación ilegal y produce un trap al S.O

En windows...

- En WIN2000 el modo núcleo ejecuta los servicios ejecutivos. El modo usuario ejecuta los procesos de usuario
- Cuando un programa se bloquea en modo usuario, a lo sumo se describe un suceso en el registro de sucesos. Si el bloqueo se produce estando en modo supervisor se genera la BSOD(pantalla azul)

Resumiendo...

- Modo Kernel:
 - Gestión de procesos: Creación y terminación, planificación, intercambio, sincronización y soporte para la comunicación entre procesos
 - Gestión de memoria: Reserva de espacio de direcciones para los procesos, Swapping, gestión y páginas de segmentos.
 - Gestión E/S: gestión de buffers, reserva de canales de E/S y de dispositivos de los procesos
 - Funciones de soporte: Gestión de interrupciones, auditoría, monitoreo
- Modo usuario
 - Debug de procesos, definición de protocolos de comunicación gestión de aplicaciones(compilador, editor, aplicaciones de usuario)

- En este modo se llevan a cabo todas las tareas que no requieran accesos privilegiados
- En este modo no se puede interactuar con el hardware
- El proceso trabaja en su propio espacio de direcciones

Protección de la memoria

- Delimitar el espacio de direcciones del proceso
- Poner límites a las direcciones que puede utilizar un proceso

La memoria principal aloja al S.O y a los procesos de usuario.

- El kernel debe proteger para que los procesos de usuario no puedan acceder donde no les corresponde
- El kernel debe proteger el espacio de direcciones de un proceso del acceso de otros procesos

Protección de la E/S

- Las instrucciones de E/S se definen como privilegiadas
- Deben ejecutarse en Modo Kernel
 - Se deberían gestionar en el kernel del sistema operativo
 - Los procesos de usuario realizan E/S a través de llamadas al SO(es un servicio del SO)

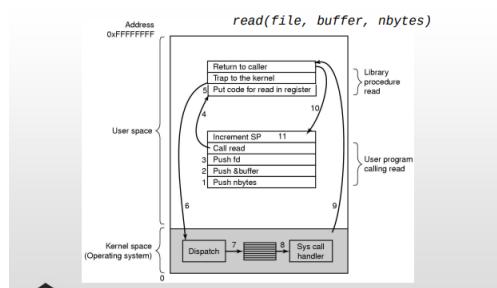
Protección de la CPU

- Uso de interrupción por clock para evitar que un proceso se apropie de la CPU
- Se implementa normalmente a través de un clock y un contador

- El kernel le da valor al contador que se decrementa con cada tick de reloj y al llegar a cero puede expulsar al proceso para ejecutar otro
- Las instrucciones que modifican el funcionamiento del reloj son privilegiadas
- Se le asigna al contador el valor que se quiere que se ejecute un proceso
- Se la usa también para el cálculo de la hora actual, basándose en cantidad de interrupciones ocurridas cada tanto tiempo y desde una fecha y hora determinada

System calls

- Es la forma en que los programas de usuario acceden a los servicios del S.O
- Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques o tablas de memoria o la pila
- Se ejecutan en modo Kernel o superviso



- Categorías de system calls:
 - Control de procesos
 - Manejo de archivos
 - Manejo de dispositivos
 - Mantenimiento de información del sistema
 - Comunicaciones

Process management	
Call	Description
pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &status, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

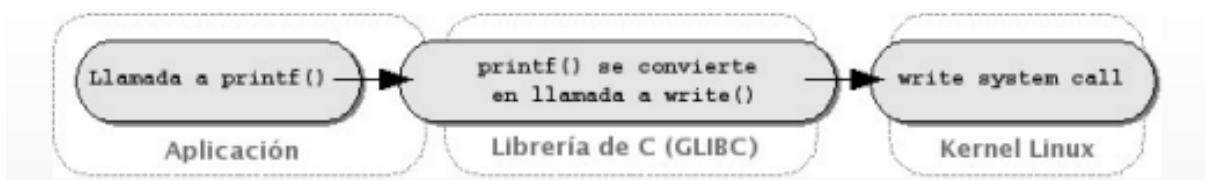
File management	
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970



- Para iniciar la system call se indica:
 - Numero de syscall que se quiere ejecutar
 - los parametros de la misma
- Luego se emite un aviso al SO para pasar a modo kernel y gestionar la system call
- Se evalúa al system call deseada y se ejecuta

Ejemplos:

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Procesos

Definición de proceso

- Es un programa en ejecución
- Para nosotros serán sinónimos: tarea, job y proceso

Diferencias entre un programa y un proceso

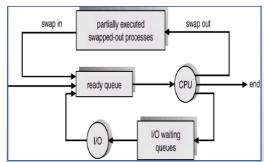
Programa

- Estático
- No tiene program counter

Proceso

- Dinámico
- Tiene program counter

- Existe desde que se edita hasta que se borra
- Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina



Componentes de un proceso

Proceso: Entidad de abstracción

Un proceso(para poder ejecutarse) incluye como mínimo:

- Sección de código(texto)
- Sección de Datos(variables globales)
- Stack(s)(datos temporarios: parámetros, variables temporales y direcciones de retorno)

Stacks

- Un proceso cuenta con 1 o más stacks
 - En general: modo usuario y modo kernel
- Se crean automáticamente y su medida se ajusta en run-time
- Está formado por stack frames que son pushed(al llamar a una rutina) y popped(cuando se retorna de ella)
- El stack frame tiene los parámetros de la rutina(variables locales), y los datos necesarios para recuperar el stack frame anterior(el contador de programa y el valor del stack pointer en el momento del llamado)

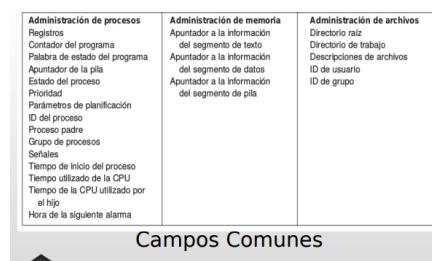
Atributos de un proceso

- Identificación del proceso, y del proceso padre
- Identificación del usuario que lo “disparó”

- Si hay estructura de grupos, grupo que lo disparó
- En ambientes multiusuario, desde que terminal y quien lo ejecuto

Process Control Block(pcb)

- Estructura de datos asociada al proceso(abstracción)
- Existe una por procesos
- Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina
- Contiene la información asociada con cada proceso:
 - PID, PPID, etc
 - Valores de los registros de la CPU(PC, AC, etc)
 - Planificación(estado, prioridad, tiempo consumido, etc)
 - Ubicación(representación)
 - Accounting
 - Entrada salida(estado, pendientes, etc)



¿Qué es el espacio de direcciones de un proceso?

- Es el conjunto de direcciones de memoria que ocupa el proceso
 - Stack, text y datos

- No incluye su PCB o tablas asociadas
- Un proceso en modo usuario puede acceder sólo a su espacio de direcciones;
- En modo Kernel, se puede acceder a estructuras internas, o a espacios de direcciones de otros procesos

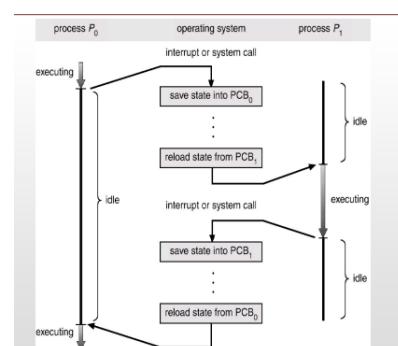
El contexto de un proceso

- Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente.
- Son parte del contexto, los registros de CPU, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.

Cambio de contexto(Context Switch)

- Se produce cuando la CPU cambia de un proceso a otro
- Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornará después a la CPU
- Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto
- Es tiempo no productivo de CPU
- El tiempo que consume depende del soporte de HW

Ejemplo:



Sobre el Kernel del Sistema Operativo

- Es un conjunto de módulos de software
- Se ejecuta en el procesador como cualquier proceso
- Entonces:
 - ¿El Kernel es un proceso? Y si es así, quién lo controla?
- Diferentes enfoques de diseño

Enfoque 1 - El Kernel como entidad independiente

- El Kernel se ejecuta fuera de todo proceso
- Es una arquitectura utilizada por los primeros SO
- Cuando un proceso es “interrumpido” o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del sistema operativo
- El Kernel tiene su propia región de memoria
- El Kernel tiene su propio Stack
- Finalizada su actividad, le devuelve el control al proceso (o a otro diferente)
- Importante:
 - El Kernel NO es un proceso. El concepto de proceso solo se asocia a programas de usuario
 - Se ejecuta como entidad independiente en modo privilegiado

Enfoque 2 - El Kernel “dentro” del proceso

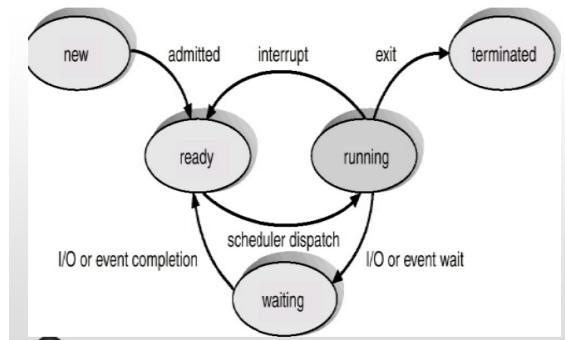
- El “código” del Kernel se encuentra dentro del espacio de direcciones de cada proceso

- El Kernel se ejecuta en el MISMO contexto que algún proceso de usuario
- El Kernel se puede ver como una colección de rutinas que el proceso utiliza
- Dentro de un proceso se encuentra el código del programa (user) y el código de los módulos de SW del SO
- Cada proceso tiene su propio stack (uno en modo usuario y otro en modo Kernel)
- El proceso es el que se Ejecuta en Modo Usuario y el Kernel del SO se ejecuta en Modo Kernel (Cambio de modo)
- El código del Kernel es compartido por todos los procesos
 - En administración de memoria veremos "como"
- Cada interrupción(incluyendo las System Call) es atendida en el contexto del proceso que se encontraba en ejecución
 - Pero en modo Kernel!!!(se pasa a este modo sin necesidad de hacer un cambio de contexto completo)
 - Si el SO determina que el proceso debe seguir ejecutándose luego de atender la interrupción, cambia a modo usuario y devuelve el control. Es más económico y performante

Estados de un proceso

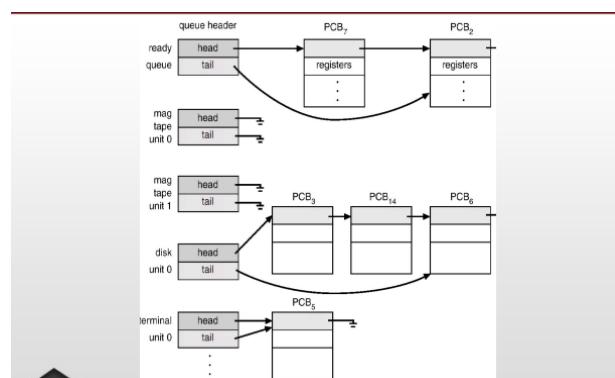
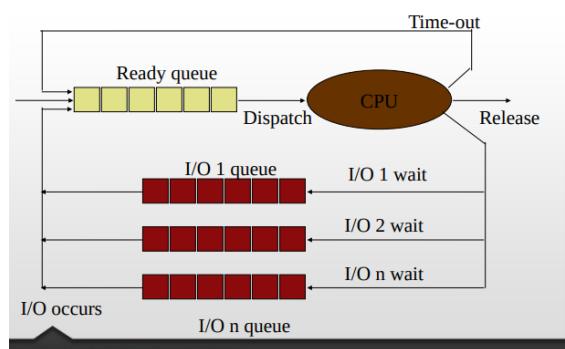
En su ciclo de vida, un proceso pasa por diferentes estados

- Nuevo (new)
- Listo para ejecutar (ready)
- Ejecutándose (running)
- En espera (waiting)
- Terminado (terminated)



Colas en la planificación de procesos

- Para realizar la planificación, el SO utiliza la PCB de cada proceso como una abstracción del mismo
- Las PCB se enlazan en Colas siguiendo un orden determinado
- Ejemplos
 - Cola de trabajos o procesos
 - Contiene todas las PCB de procesos en el sistema
 - Cola de procesos listos
 - PCB de procesos residentes en memoria principal esperando para ejecutarse



Módulos de planificación

- Son módulos (SW) del Kernel que realizan distintas tareas asociadas a la planificación
 - Se ejecutan ante determinados eventos que así lo requieren:
 - Creación/Terminación de procesos
 - Eventos de Sincronización o de E/S
 - Finalización de lapso de tiempo
 - Etc
 - Scheduler (planificador) de long term
 - Scheduler de short term
 - Scheduler de medium term
- Su nombre proviene de la frecuencia de ejecución

Existen otros módulos: Dispatcher y Loader

Pueden no existir como módulos separados de los schedulers vistos, pero la función debe cumplirse

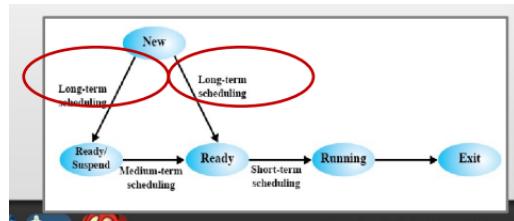
Dispatcher: hace cambio de contexto, cambio de modo de ejecución..."despacha" el proceso elegido por el Short Term (es decir, salta a la instrucción a ejecutar)

Loader: Carga en memoria el proceso elegido por el Long term

Long term Scheduler

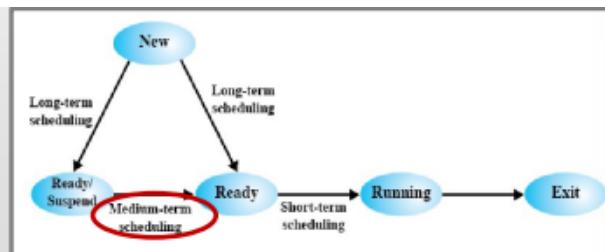
Controla el grado de multiprogramación, es decir la cantidad de procesos en memoria

Puede no existir este scheduler y absorber esta tarea del short term



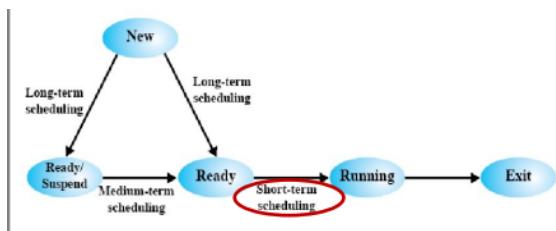
Medium Term Scheduler (swapping)

- Si es necesario, reduce el grado de multiprogramación
- Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema
- Términos asociados: swap out (sacar de memoria), swap in (Volver a memoria)

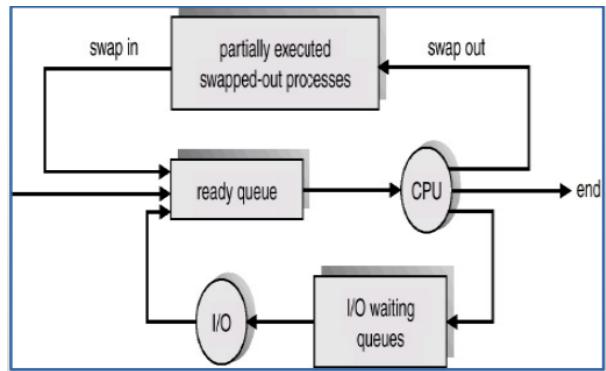


Short Term Scheduler

- Decide a cuál de los procesos en la cola de listos se elige para que use la CPU
- Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling



Procesos en espera y swapeados



Sobre el estado nuevo (new)

- Un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre
- En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria

Sobre el estado listo (ready)

- Luego que el scheduler de largo plazo eligió al proceso para cargarlo en memoria, el proceso queda en estado listo
- El proceso sólo necesita que se le asigne CPU
- Está en la cola de procesos listos (ready queue)

Sobre el estado en ejecución (running)

- El scheduler de corto plazo lo eligió para asignarle CPU
- Tendrá la CPU hasta que el período de tiempo asignado (quantum o time slice), termine o hasta que necesite realizar alguna operación de E/S

Sobre el estado de espera (waiting)

- El proceso necesita que se cumpla el evento esperado para continuar
- El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso
- Sigue en memoria, pero no tiene la CPU
- Al cumplirse el evento, pasará al estado listo

Transiciones

New-Ready: Por elección del scheduler de largo plazo (carga en memoria)

Ready-Running: Por elección del scheduler de corto plazo (asignación de CPU)

Running-Waiting: el proceso “se pone a dormir”, esperando por un evento.

Waiting-Ready: Terminó la espera y compite nuevamente por la CPU.

Caso especial: Running-ready

- Cuando el proceso termina su quantum (franja de tiempo) sin haber necesitado ser interrumpido por un evento, pasa al estado de ready, para competir por CPU, pues no está esperando por ningún evento...
- Se trata de un caso distinto a los anteriores, porque el proceso es expulsado de la CPU contra su voluntad

- Esta situación se da en algoritmos apropiativos

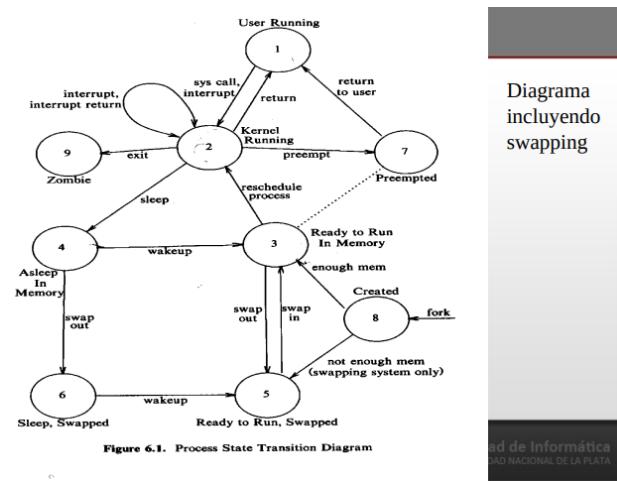
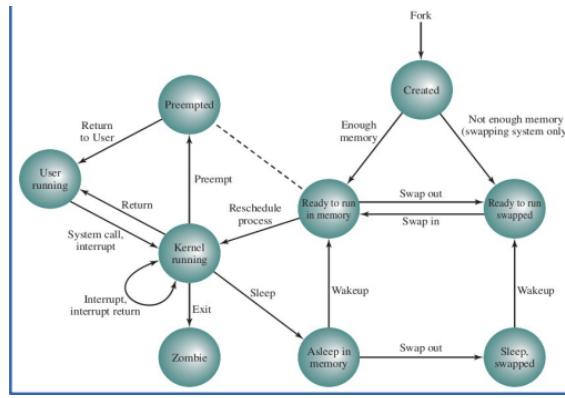


Diagrama incluyendo swapping

Explicación por estado

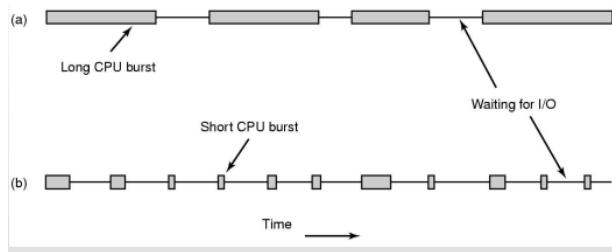
1. Ejecución en modo usuario
2. Ejecución en modo Kernel
3. El proceso está listo para ser ejecutado cuando sea elegido
4. Proceso en espera en memoria principal
5. Proceso listo, pero el swaper debe llevar al proceso a memoria principal antes que el kernel lo pueda elegir para ejecutar
6. Proceso en espera en memoria secundaria
7. Proceso retornando desde el modo Kernel al user. Pero el kernel se apropiá, hace un context switch para darle a la cpu a otro proceso
8. Proceso recientemente creado en transición: existe, pero aun no está listo para ejecutar, ni está dormido
9. Proceso ejecutó la system call exit y está en estado zombie. Ya no existe más, pero se registran datos sobre su uso, código resultante del exit. Es el estado final

Diagrama de transiciones UNIX



Comportamiento de los procesos

Procesos alternan ráfagas de CPU y de I/O



- CPU-bound
 - Mayor parte del tiempo utilizando la CPU
- I/O bound
 - Mayor parte del tiempo esperando por I/O
- La velocidad de la CPU es mucho más rápida que la de los dispositivos de E/S
 - Pensar: Necesidad de atender rápidamente procesos I/O-bound para mantener el dispositivo ocupado y aprovechar la CPU para los procesos CPU-bound

Planificación

- Planificación
 - Necesidad de determinar cual de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado
- Algoritmo de planificación
 - Algoritmo utilizado para realizar la planificación del sistema

Algoritmos apropiativos y No apropiativos

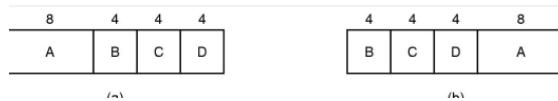
- En los algoritmos apropiativos existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU
- En los algoritmos no apropiativos los procesos se ejecutan hasta que el mismo (por si propia cuenta) abandone la CPU
 - Se bloquea por E/S o finaliza
 - No hay decisiones de planificación durante las interrupciones de reloj

Categorías de los Algoritmos de planificación

- Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:
 - Equidad: Otorgar una parte justa de la CPU a cada proceso
 - Balance: Mantener ocupadas todas las partes del sistema
- Ejemplos:
 - Procesos por lotes (Batch)
 - Procesos Interactivos
 - Procesos en Tiempo real

Procesos Batch

- No existen usuarios que esperen una respuesta en una terminal
- Se pueden utilizar algoritmos no apropiativos
- Metas propias de este tipo de algoritmos:
 - Rendimiento: Maximizar el número de trabajos por hora
 - Tiempo de retorno: Minimizar los tiempos entre el comienzo y la finalización
 - El tiempo en espera se puede ver afectado
 - Uso de la CPU: mantener la CPU ocupada la mayor cantidad de tiempo posible
- Ejemplos de Algoritmos
 - FCFS - First Come First Served
 - SJF - Shortest Job First



Procesos interactivos

- No solo interacción con los usuarios
 - Un servidor, necesita de varios procesos para dar respuesta a diferentes requerimientos
- Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU
- Metas propias de este tipo de algoritmos:
 - Tiempo de respuesta: Responder a peticiones con rapidez
 - Proporcionalidad: Cumplir con expectativas de los usuarios

- Si el usuario le pone STOP al reproductor de música, que la música deje de ser reproducida en un tiempo considerablemente corto
- Ejemplos de algoritmos:
 - Round Robin
 - Prioridades
 - Colas multinivel
 - SRTF - Shortest remaining time first

Política Versus Mecanismo

- Existen situaciones en las que es necesario que la planificación de uno o varios procesos se comporte de manera diferente
- El algoritmo de planificación debe estar parametrizado, de esta manera los procesos/usuarios pueden indicar los parámetros para modificar la planificación
- El Kernel implementa el mecanismo
- El usuario/proceso/administrador utiliza los parámetros para determinar la Política
- Ejemplo:
 - Un algoritmo de planificación por prioridades y una System Call que permite modificar la prioridad de un procesos
 - Un proceso puede determinar las prioridades de los procesos que el crea, según la importancia de los mismos

Creación de procesos

- Un proceso es creado por otro proceso
- Un proceso padre tiene uno o más procesos hijos
- Se forma un árbol de procesos

Actividades en la creación

- Crear la PCB
- Asignar PID(process identification) único
- Asignarle memoria para regiones
 - Stack, text y datos
- Crear estructuras de datos asociadas
 - Fork (copiar el contexto, regiones de datos, text y stack)

Relación entre procesos Padre e hijo

Con respecto a la Ejecución:

- El padre puede continuar ejecutándose concurrentemente con su hijo
- El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución

Con respecto al espacio de direcciones:

- El hijo es un duplicado del proceso padre (caso Unix)
 - Se crea un nuevo espacio de direcciones copiando el del padre
- Se crea el proceso y se le carga adentro el programa (caso Windows)
 - Se crea un nuevo espacio de direcciones vacío

Creación de procesos(continuación)

- En UNIX: (2 System Calls)

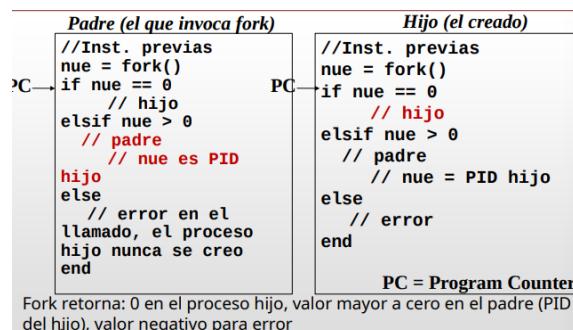
- system call fork() crea un nuevo proceso igual al llamador
- system call execve(), generalmente usada después del fork, carga un nuevo programa en el espacio de direcciones
- En Windows:(1 system call)
 - system call CreateProcess() crea un nuevo proceso y carga el programa para la ejecución

¿Cómo funciona el Fork?

Padre

```
//Instrucciones previas
```

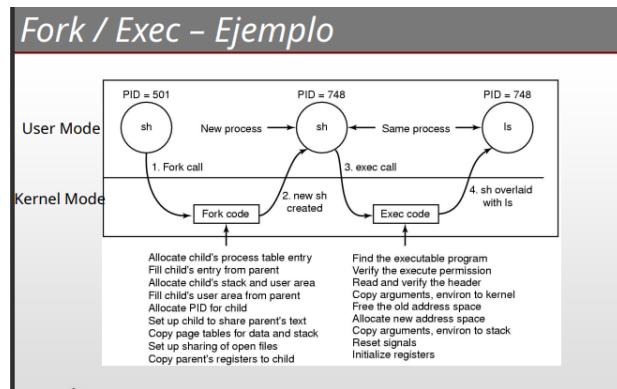
```
nue = fork()
if nue == 0
    // hijo
elseif nue > 0
    // padre
    // nue es el PID del hijo
else
    // error
end
```



Terminación de procesos

- Ante un (exit), se retorna el control al sistema operativo
 - El proceso padre puede esperar recibir un código de retorno(via wait). Generalmente se lo usa cuando se requiere que el padre espere a los hijos
- Proceso padre puede terminar la ejecución de sus hijos(kill)

- La tarea asignada al hijo se terminó
- Cuando el padre termina su ejecución
 - Habitualmente no se permite a los hijos continuar, pero existe la opción
 - Terminación en cascada



System Calls - Unix

System call	Description
pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause()	Suspend the caller until the next signal

Syscalls de Procesos

System calls - Windows

Win32 API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section

Syscalls de Procesos

Memoria

- La organización y administración de la “memoria principal” es uno de los factores mas importantes en el diseño de los S.O
- Los programas y datos deben estar en el almacenamiento principal para:
 - Poderlos ejecutar
 - Referenciarlos directamente
- El SO debe
 - Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no
 - Asignar espacio en memoria principal a los procesos cuando estos la necesitan
 - Libera espacio de memoria asignada a procesos que han terminado
- Se espera de un S.O un uso eficiente de la memoria con el fin de alojar el mayor número de procesos
- El S.O. debe:
 - Lograr que el programador se abstraiga de la alocación de los programas
 - Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros
 - Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria (librerías, código en común, etc.)
 - Garantizar la performance del sistema

Administración de la memoria

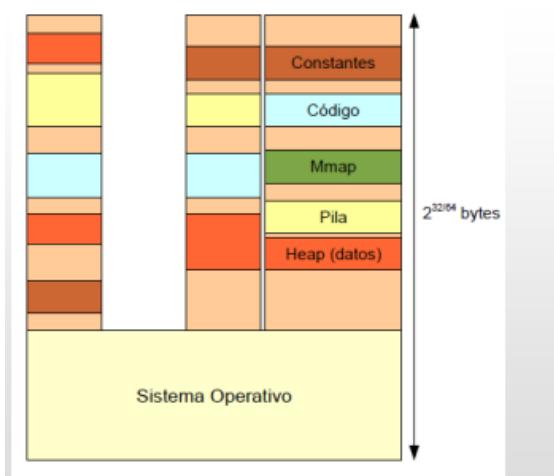
- División lógica de la memoria física para alojar múltiples procesos
 - Garantizando protección
 - Depende del mecanismo previsto por el HW
- Asignación eficiente
 - Contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos

Requisitos

- Reubicación
 - El programador no debe ocuparse de conocer donde será colocado en la Memoria RAM
 - Mientras un proceso se ejecuta puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones
 - Las referencias a la memoria deben "traducir" según ubicación actual del proceso
- Protección
 - Los procesos NO deben referenciar-acceder a direcciones de memoria de otros procesos
 - Salvo que tengan permiso
 - El chequeo se debe realizar durante la ejecución:
 - NO es posible anticipar todas las referencias a memoria que un proceso puede utilizar
- Compartición
 - Permitir que varios procesos accedan a la misma porción de memoria.
 - Ej: rutinas comunes, librerías, espacios explícitamente compartidos
 - Permite un mejor uso-aprovechamiento - de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones

Abstracción - Espacio de Direcciones

- Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos
- El tamaño depende de la Arquitectura del procesador
 - 32 bits: $0..(2^{32})-1$
 - 64 bits: $0..(2^{64})-1$
- Es independiente de la ubicación “real” del proceso en la Memoria RAM



Direcciones

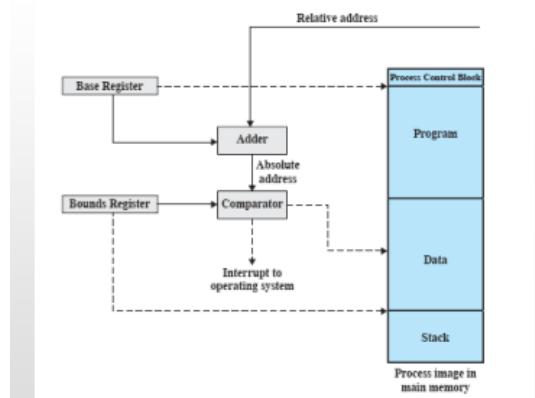
- Lógicas
 - Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria
 - Representa una dirección en el “Espacio de Direcciones del proceso”
- Físicas
 - Referencia una localidad en la Memoria Física (RAM)
 - Dirección absoluta

En caso de usar direcciones lógicas, es necesaria algún tipo de conversión a direcciones Físicas

Conversión de Direcciones

Una forma simple de hacer esto es utilizando registros auxiliares

- Registro Base
 - Dirección de comienzo del Espacio de Direcciones del proceso en la RAM
- Registro límite
 - Dirección final del proceso o medida del proceso - Tamaño de su Espacio de direcciones
- Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria
- Varían entre procesos (Context Switch)



Dir. Lógicas vs. Físicas

- Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal, se deben transformar en direcciones físicas
 - Resolución de direcciones (address-binding):
 - transformar la dirección lógica en la física correspondiente
- Resolución en momento de compilación (Archivos .com de DOS) y en tiempo de carga
 - Direcciones lógicas son idénticas

- Para reubicar un proceso es necesario recompilarlo o recargarlo
- Resolución en tiempo de ejecución
 - Direcciones lógicas y físicas son diferentes
 - Direcciones lógicas son llamadas “Direcciones virtuales”
 - La reubicación se puede realizar fácilmente
 - El mapeo entre “Virtuales” y “Físicas” es realizado por hardware
 - Memory Management Unit (MMU)

Memory Management Unit (MMU)

- Dispositivo de Hardware que mapea direcciones virtuales a físicas
 - Es parte del Procesador
 - Re-programar el MMU es una operación privilegiada
 - solo puede ser realizada en Kernel Mode
- El valor en el registro “registro de realocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria
 - Los procesos usan direcciones físicas



Mecanismo de asignación de memoria

- Particiones fijas: El primer esquema implementado

- La memoria se divide en particiones o regiones de tamaño Fijo (Pueden ser todas del mismo tamaño o no)
- Alojan un proceso en cada una
- Cada proceso se coloca de acuerdo a algún criterio (Primer ajuste, Mejor ajuste, peor ajuste, Siguiente ajuste)
- Particiones dinámicas: La evolución del esquema anterior
 - Las particiones varían en tamaño y en número
 - Alojan un proceso en cada una
 - Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso

Fragmentación

- La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse de forma contigua
- **Fragmentación Interna:**
 - Se produce en el esquema de particiones fijas
 - Es la porción de la partición que queda sin utilizar
- **Fragmentación Externa:**
 - Se produce en el esquema de particiones dinámicas
 - Son huecos que van quedando en la memoria a medida que los procesos finalizan
 - Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alojar un proceso, pero que no la podamos utilizar
 - Para solucionar el problema se puede acudir a la compactación, pero es muy costosa

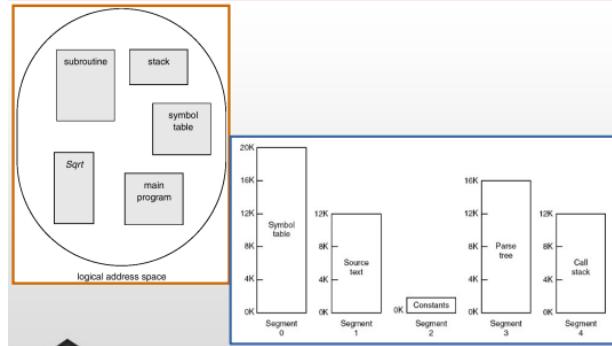
Problemas del esquema

- El esquema de Registro Base + Límite presenta problemas:
 - Necesidad de almacenar el Espacio de Direcciones de forma continua en la memoria física
 - Los primeros SO definían particiones fijas de memoria, luego evolucionaron a particiones dinámicas
 - Fragmentación
 - Mantener “partes” del proceso que no son necesarias
 - Los esquemas de particiones fijas y dinámicas no se usan hoy en día
- Solución:
 - Segmentación
 - Paginación

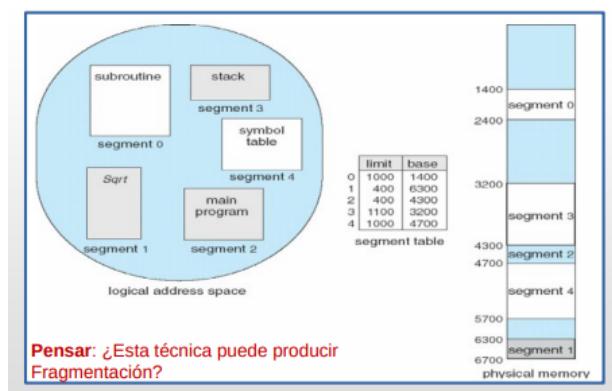
Segmentación

- Esquema que se asemeja a la “visión del usuario”. El programa se divide en partes/secciones
- Un programa es una colección de segmentos. Un segmento es una unidad lógica como:
 - Programa principal, procedimientos y funciones, variables locales y globales, stack, etc.
- Puede causar Fragmentación

Programa desde la visión del usuario



Ejemplo de segmentación



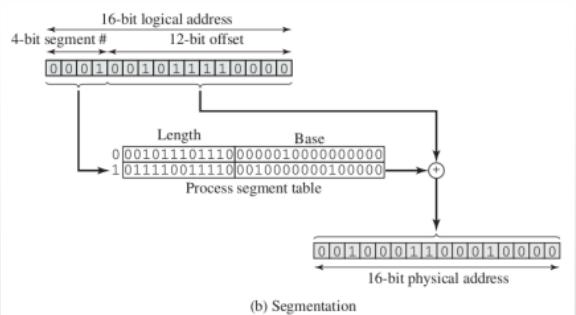
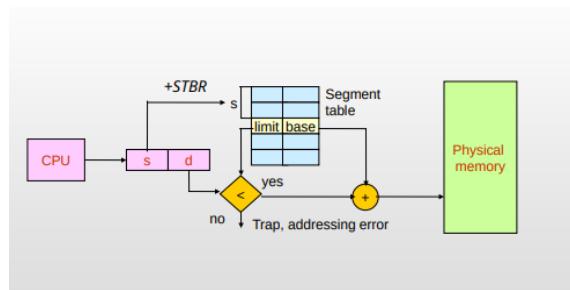
Continuación Segmentación

- Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas)
- Las direcciones lógicas consisten en 2 partes:
 - Selección de segmento
 - Desplazamiento dentro del segmento

Arquitectura de la segmentación

- Tabla de segmentos

- Permite mapear la dirección lógica en física. Cada entrada contiene:
 - Base: Dirección física de comienzo del segmento
 - Limit: Longitud del Segmento
- Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos
- Segment-table length register (STLR): cantidad de segmentos de un programa

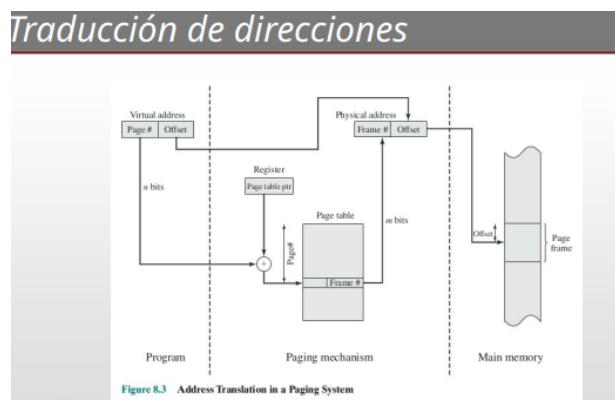
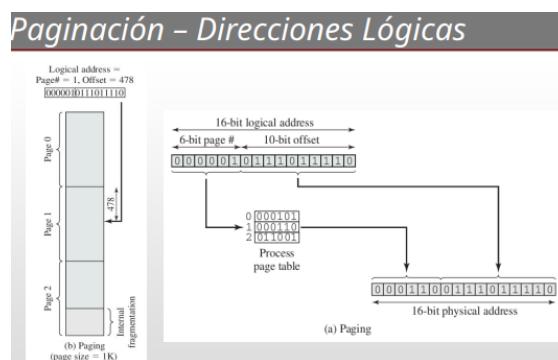
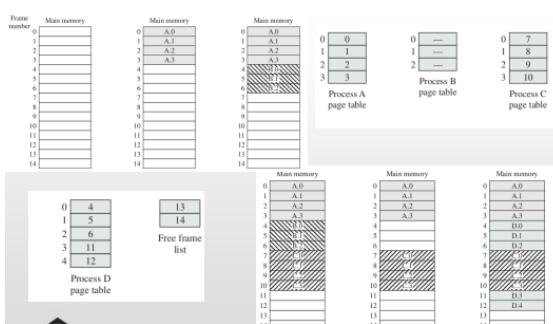
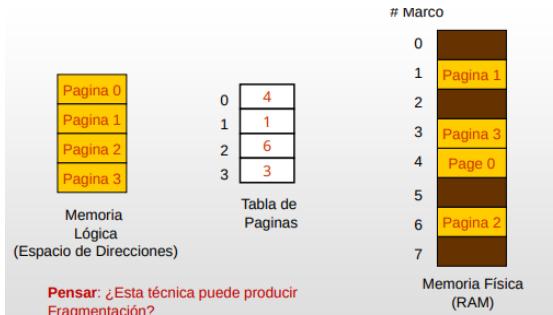


Paginación

- Memoria física es dividida lógicamente en pequeños trozos de igual tamaño (Marcos)
- Memoria Lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos (Páginas)
- El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras) el Marco en la que se coloca cada página
- La dirección lógica se interpreta como:

- Un número de página y un desplazamiento dentro de la misma

Ejemplos:



Ventajas sobre Segmentación

- Compartir
- Proteger

Segmentación paginada

- La paginación
 - Transparente al programador
 - Elimina Fragmentación externa
- Segmentación
 - Es visible al programador
 - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección
- Segmentación paginada: Cada segmento es dividido en páginas de tamaño fijo

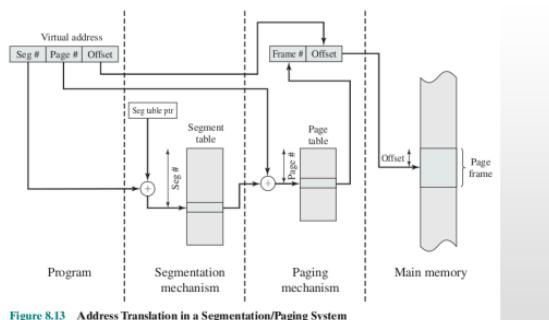
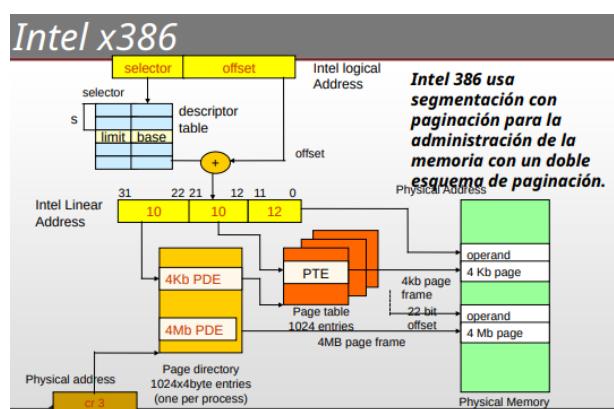


Figure 8.13 Address Translation in a Segmentation/Paging System



Hasta ahora

- Con paginación vimos que el espacio de direcciones de un proceso no necesariamente debe estar “contiguo” en la memoria para ejecutarse
 - El hardware traduce direcciones lógicas a físicas utilizando las tablas de páginas que el SO administra

Motivación para Memoria Virtual

- Podemos pensar también que, no todo el espacio de direcciones del proceso se necesitó en todo momento:
 - Rutinas o librerías que se ejecutan una única vez (o nunca)
 - Partes del programa que no vuelven a ejecutarse
 - Regiones de memoria alocadas dinámicamente y luego liberadas
 - Etc.
- No hay necesidad que la totalidad de la imagen del proceso sea cargada en memoria

Como se puede trabajar...

- El SO puede traer a memoria las “piezas” de un proceso a medida que éste las necesita.
- Definiremos como “Conjunto Residente” a la porción del espacio de direcciones del proceso que se encuentra en memoria
 - Alguna bibliografía lo llama “Working set”
- Con el apoyo del HW:
 - Se detecta cuando se necesita una porción del proceso que no está en su conjunto residente

- Se debe cargar en memoria dicha sección para continuar con la ejecución

Ventajas

- Más procesos pueden ser mantenidos en memoria
 - Sólo son cargadas algunas secciones de cada proceso
 - Con más procesos en memoria principal es más probable que existan más procesos Ready
- Un proceso puede ser más grande que la memoria principal
 - El usuario no se debe preocupar por el tamaño de sus programas
 - La limitación la impone el HW y el bus de direcciones

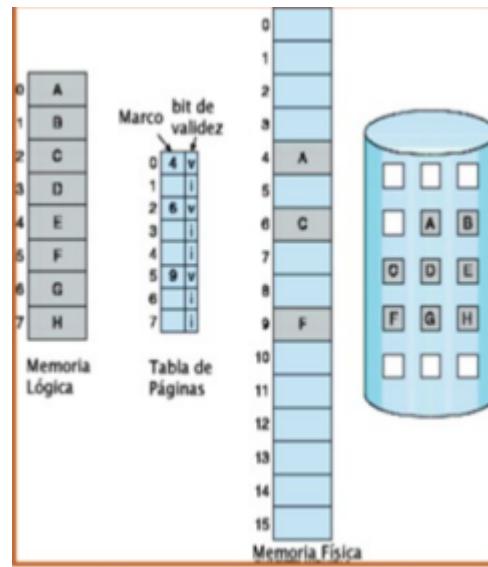
¿Qué se necesita para Memoria Virtual?

- El hardware debe soportar paginación por demanda(y/o segmentación por demanda)
- Un dispositivo de memoria secundaria (disco) que dé el apoyo para almacenar las secciones del proceso que no estén en memoria principal (área de intercambio)
- El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria

Memoria Virtual con Paginación

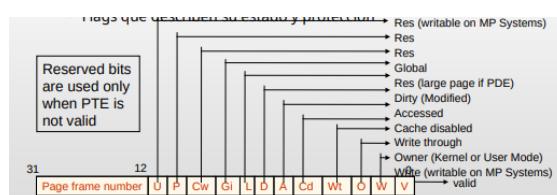
- Cada proceso tiene su tabla de páginas
- Cada entrada en la tabla referencia al frame o marco en el que se encuentra la página en la memoria principal
- Cada entrada en la tabla de página tiene bits de control (entre otros):
 - Bit V: Indica si la página está en memoria (Lo activa/desactiva el SO, lo consulta el HW)

- Bit M: Indica si la página fue modificada. Si se modificó, en algún momento, se deben refejar los cambios en Memoria Secundaria (lo activa el HW, lo consulta y desactiva el SO)



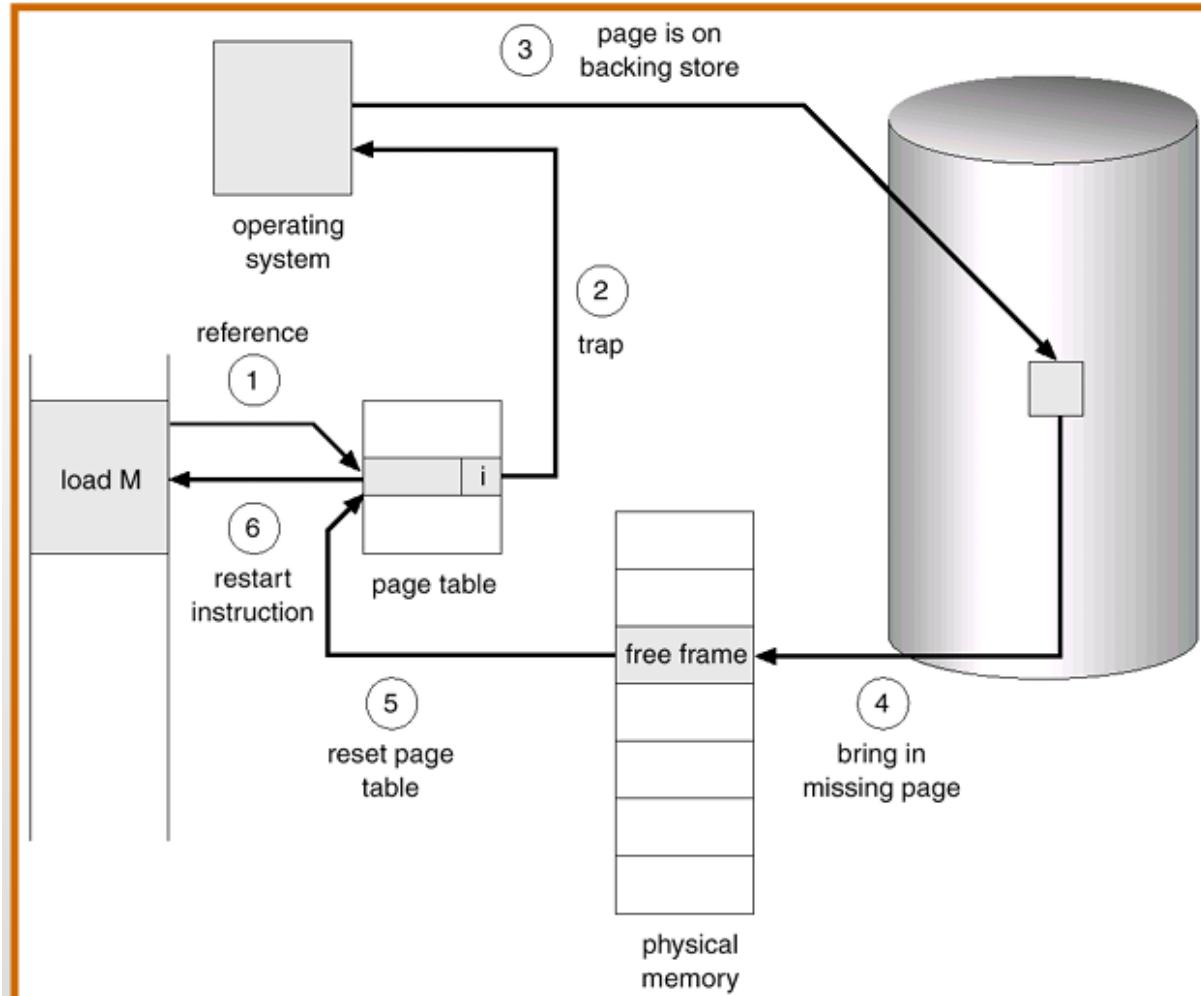
Entrada en la Tabla de páginas de x86 (32 bits)

- El HW define el formato de la tabla de páginas y el SO se adapta a él
- Una entrada válida tiene:
 - Bit V = 1
 - Page Frame Number (PFN0:) - Marco de memoria asociado
 - Flags que describen su estado y protección



Fallo de páginas (Page Fault)

- Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en memoria principal. Bit V = 0(Tambien marcado con i = inválido)
 - La página no se encuentra en su conjunto residente
- El HW detecta la situación y genera un trap al S.O
- El S.O podrá colocar al proceso en estado de “Blocked” (espera) mientras gestiona que la página que se necesite se cargue
- El S.O. busca un “Frame o Marco Libre” en la memoria y genera una operación de E/S al disco para copiar en dicho Frame la página del proceso que se necesita utilizar.
- El SO puede asignarle la CPU a otro proceso mientras se completa la E/S
- La E/S se realizará y avisará mediante interrupción su finalización.
- Cuando la operación de E/S finaliza, se notifica al SO y este:
 - Actualiza la tabla de páginas del proceso
 - Coloca el Bit V en 1 en la página en cuestión
 - Coloca la dirección base del Frame donde se colocó la página
 - El proceso que generó el Fallo de Página vuelve a estado de Ready (listo)
 - Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes generó el fallo de página



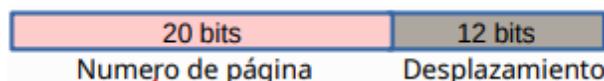
- La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles
- Cada vez que hay que alocar una página en un marco, se produce un fallo de página
- ¿Qué sucede si es necesario alocar una página y ya no hay marcos disponibles?
- Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos (FIFO, Óptimo, LRU, etc.)
- ¿Cuál es el mejor algoritmo?:
- El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro próximo

Tabla de Páginas

- Cada proceso tiene su tabla de páginas
- El tamaño de la tabla de páginas depende del espacio de direcciones del proceso.
- Puede alcanzar un tamaño considerable
- Formas de organizar:
 - Tabla de 1 nivel: Tabla única lineal
 - Tabla de 2 niveles (o más, multinivel)
 - Tabla invertida: Hashing
- La forma de organizarla depende del HW subyacente

Tabla de 1 nivel - 32 bits

- Direcciones de 32bits

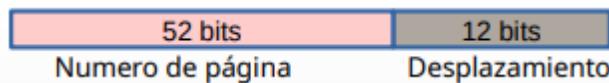


- Ejemplo:
 - Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso = 2^{20} (1.048.576)
 - El tamaño de cada página es de 4KB (2^{12})
 - El tamaño de cada PTE es de 4 bytes
 - Cantidad de PTEs que entran en un marco: $4KB/4B = 2^{10}$
 - Tamaño de tabla de páginas
 - Cantidad de marcos necesarios para todas las PTEs de la tabla de páginas de un proceso = $2^{20}/2^{10} = 2^{10}$

- Tamaño tabla de páginas del proceso:
2
 $10 * 4\text{bytes} = 4\text{MB}$ por proceso

Tabla de 1 nivel - 64 bits

- Direcciones de 64 bits



- Ejemplo
 - Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso = 252
 - El tamaño de cada página es de 4KB
 - El tamaño de cada PTE es de 4 bytes
 - Cantidad de PTEs que entran en un marco: $4\text{KB}/4\text{B} = 210$
 - Tamaño de tabla de páginas
 - Cantidad de marcos necesarios para todas las PTEs de la tabla de páginas de un proceso = $252/210 = 242$
 - Tamaño tabla de páginas del proceso = $242 * 4\text{bytes} = 254$
Más de 16.000GB por proceso!!!

Tabla de páginas - Tabla de 2 niveles

- El propósito de la tabla de páginas multinivel es dividir la tabla de páginas lineal en múltiples tablas de páginas
- Cada tabla de páginas suele tener el mismo tamaño pero se busca que tengan un menor número de páginas por tabla
- La idea general es que cada tabla sea más pequeña
- Se busca que la tabla de páginas no ocupe demasiada memoria RAM

- Además solo se carga una parcialidad de la tabla de páginas (solo que se necesite resolver)
- Existe un esquema de direcciones indirectos

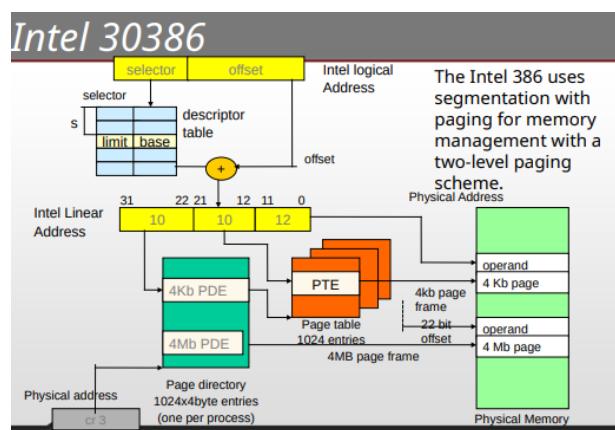
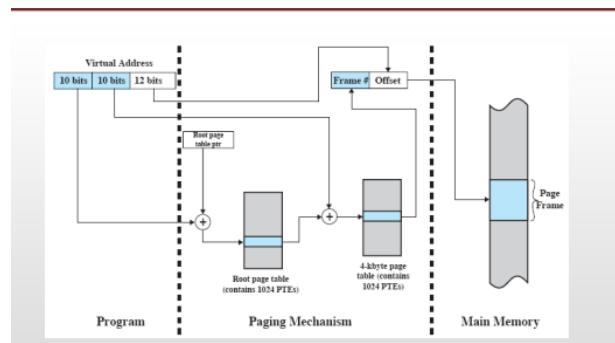


Tabla de Páginas - x64

- Se usan 47 bits para direccionar
- Usa tabla de páginas de 4 niveles

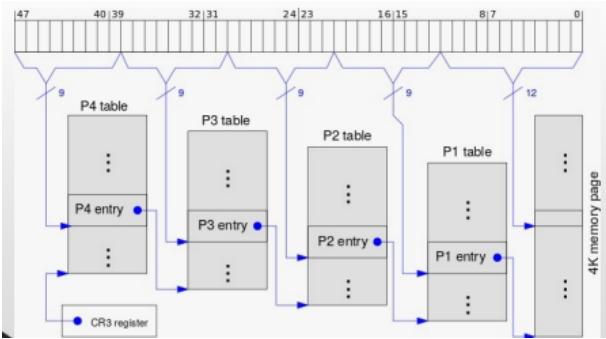
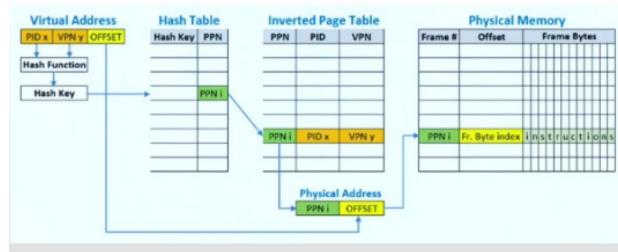


Tabla de Páginas - Tabla invertida

- Utilizada en Arquitecturas donde el espacio de direcciones es muy grande
 - Las tablas de páginas ocuparían muchos niveles y la traducción sería costosa
 - Por esta razón se adopta esta técnica
- Por ejemplo si el espacio de direcciones es de 2 a la 64 bytes, con páginas de 4 KB, necesitamos una tabla de páginas con 252 entradas
- Si cada entrada es de 8 bytes, la tabla es de más de 30 millones de Gigabytes (30 PB)
- Hay una entrada por cada marco de página en la memoria real. Es la visión inversa a la que veníamos viendo
- Hay una sola tabla para todo el sistema
- El espacio de direcciones de la tabla se refiere al espacio físico de la RAM, en vez del espacio de direcciones virtuales de un proceso
- Usada en PowerPC, UltraSPARC, y IA-64
- El número de página es transformado en un valor de HASH
- El HASH se usa como índice de la tabla invertida para encontrar el marco asociado
- Se define un mecanismo de encadenamiento para solucionar colisiones (cuando el hash da igual para 2 direcciones virtuales)
- Solo se mantiene los PTEs de páginas presentes en memoria física

- La tabla invertida es organizada como tabla hash en memoria principal
 - Se busca indexadamente por número de página virtual
 - Si está presente en tabla, se extrae el marco de página y sus protecciones
 - Si no está presente en tabla, corresponde a un fallo de página



Tamaño de la Pagina

- Pequeño
 - Menor Fragmentación Interna
 - Más páginas requeridas por proceso, tablas de páginas más grandes
 - Más páginas pueden residir en memoria
- Grande
 - Mayor Fragmentación Interna
 - La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente, más rápido se mueven páginas hacia la memoria principal
- Relación con la E/S
 - Vel. de transferencia: 2Mb/s
 - Latencia: 8ms
 - Búsqueda: 20ms
- Página de 512 bytes
 - 1 página total: 28,2 ms →

- Solo 0,2 ms de transferencia (1%)
- 2 paginas 56,4 ms →
- Pagina de 1024 bytes
 - total: 28,4 ms
 - Solo 0,4 ms de transferencia

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Translation Lookaside Buffer

- Cada referencia en el espacio virtual puede causar 2 (o más) accesos a la memoria física
 - Uno(o más) para obtener la entrada en la tabla de páginas
 - Uno para obtener los datos
- Para solucionar este problema, una memoria cache de alta velocidad es usada para almacenar entradas de páginas
 - TLB
- El buffer de traducción adelantada contiene las entradas de la tabla de páginas que fueron usadas más recientemente
- Dada una dirección virtual, el procesador examina la TLB

- Si la entrada de la tabla de páginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física
- Si la entrada no es encontrada en la TLB (miss), el número de página es usado como índice en la tabla de páginas del procesos
- Se controla si la página está en memoria
 - Si no está, se genera un Page Fault
- La TLB es actualizada para incluir la nueva entrada
- El cambio de contexto genera la invalidación de las entradas de la TLB. Analizar. ¿Qué ocurre si el Quantum en Round Robin es chico? ¿y al contrario?

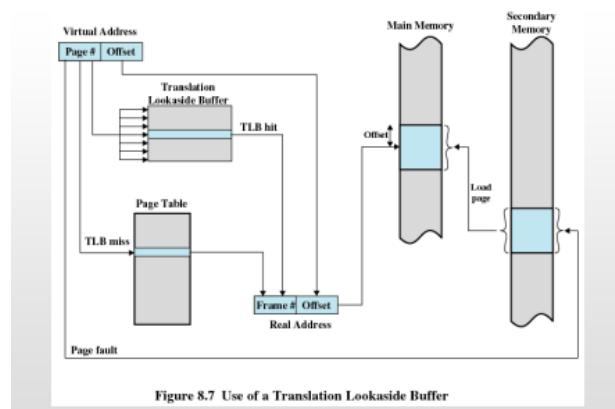


Figure 8.7 Use of a Translation Lookaside Buffer

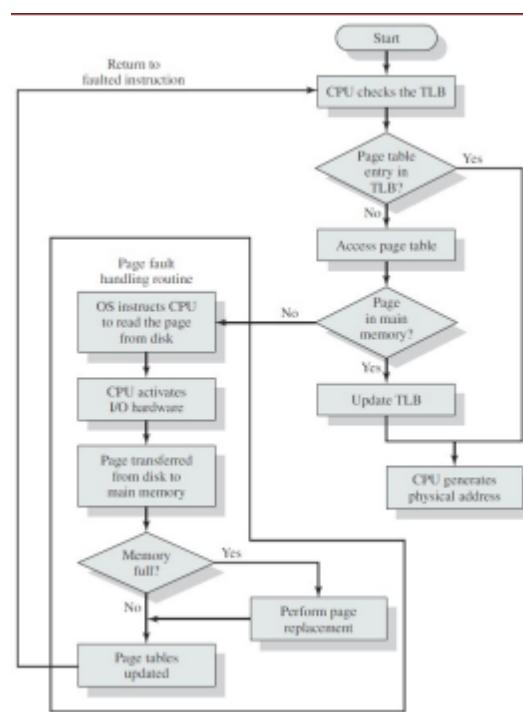


Figure 8.8 Generation of Page Faults and Translation Lookaside Buffer (TLB)

Políticas de manejo de MV

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy	Cuando una página debe ser llevada a la memoria
Demand paging	
Prepaging	
Placement Policy	Donde ubicarla (best-fit, first-fit, etc...)
Replacement Policy	
Basic Algorithms	Elección de víctima
Optimal	
Least recently used (LRU)	
First-in-first-out (FIFO)	
Clock	
Page Buffering	
Resident Set Management	
Resident set size	
Fixed	Cuántas páginas se traen a memoria
Variable	
Replacement Scope	
Global	
Local	
Cleaning Policy	
Demand	Cuando una página modificada debe llevarse a disco
Precleaning	
Load Control	# de procesos en memoria
Degree of multiprogramming	

Asignación de Marcos - Asignación Fija

- Asignación equitativa - Ejemplo: Si tengo 100 frames y 5 procesos, 20 frames para cada proceso
- Asignación proporcional: Se asigna acorde al tamaño de cada proceso

$$\begin{aligned}
 s_i &= \text{size of process } p_i \\
 S &= \sum_i s_i \\
 m &= \text{total number of frames} \\
 a_i &= \text{allocation for } p_i = \frac{s_i}{S} \times m
 \end{aligned}
 \quad
 \begin{aligned}
 m &= 64 \\
 s_1 &= 10 \\
 s_2 &= 127 \\
 a_1 &= \frac{10}{137} \times 64 \approx 5 \\
 a_2 &= \frac{127}{137} \times 64 \approx 59
 \end{aligned}$$

Reemplazo de páginas

- Que sucede si ocurre un fallo de página y todos los marcos están ocupados: "Se debe seleccionar una página víctima"
- ¿Cuál sería el reemplazo óptimo?
 - Que la página a ser removida no sea referenciada en un futuro próximo

- La mayoría de los reemplazos predicen el comportamiento futuro mirando el comportamiento pasado

Alcance de reemplazo

- Reemplazo global
 - El fallo de página de un proceso puede reemplazar la página de cualquier proceso
 - El SO no controla la tasa de page-faults de cada proceso
 - Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él
 - Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad
- Reemplazo Local
 - El fallo de página de un proceso solo puede reemplazar sus propias páginas – De su Conjunto Residente
 - No cambia la cantidad de frames asignados
 - El SO puede determinar cuál es la tasa de pagefaults de cada proceso
 - Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

Asignación y alcance

Table 8.5 Resident Set Management

	Local Replacement	Global Replacement
Fixed Allocation	<ul style="list-style-type: none"> • Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process. 	<ul style="list-style-type: none"> • Not possible.
Variable Allocation	<ul style="list-style-type: none"> • The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process. 	<ul style="list-style-type: none"> • Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Algoritmos de reemplazo

- óptimo: Es sólo teórico
- FIFO: Es el más sencillo
- LRU (Least Recently Used): Requiere soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas más recientemente accedidas
- 2da. Chance: Un avance del FIFO tradicional que beneficia a las páginas más referenciadas
- NRU (Non Recently Used):
 - Utiliza bits R y M
 - Favorece a las páginas que fueron usadas recientemente

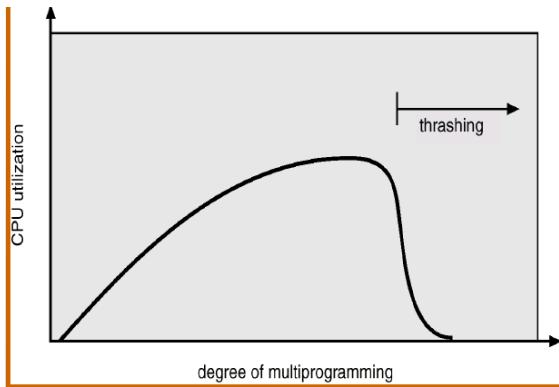
Thrashing (hiperpaginación)

- **Concepto:** decimos que un sistema está en thrashing cuando pasa más tiempo paginando que ejecutando procesos.
- Como consecuencia, hay una baja importante de performance en el sistema

Ciclo del thrashing

1. El kernel monitorea el uso de la CPU
2. Si hay una baja utilización aumenta el grado de multiprogramación
3. Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos
4. Un proceso necesita más frames. Comienzan los page-faults y robo de frames a otros procesos

5. Por swapping de páginas, y encolamiento en dispositivos, baa el uso de la cpu
6. Vuelve a 1



El scheduler de CPU y el thrashing

1. Cuando se decrementa el uso de la CU, el scheduler long term aumenta el grado de multiprogramación
2. El nuevo proceso inicia nuevos page-faults, y por lo tanto, más actividad de paginado
3. Se decrementa el uso de la CPU
4. Vuelve a 1

Control del thrashing

- Se puede limitar el thrashing usando algoritmos de reemplazo local
- Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos
- Si bien perjudica la performance del sistema, es controlable

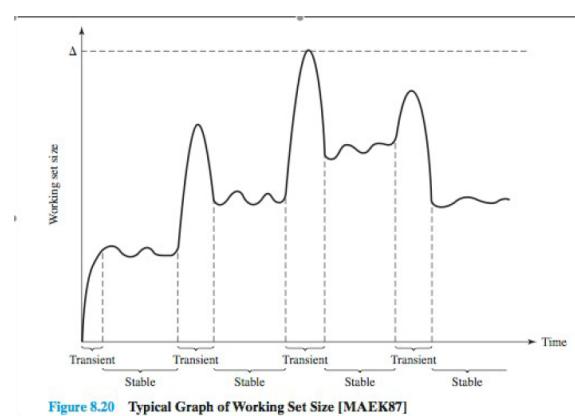
Conclusión sobre thrashing

- Si un proceso cuenta con todos los frames que necesita, no habría thrashing
- Una manera de abordar esta problemática es utilizando la estrategia de Working Set, la cual se apoya en el modelo de localidad

- Otra estrategia con el mismo espíritu es la del algoritmo PFF (Frecuencia de Fallos de Página)

Modelo de localidad

- Cercanía de referencias o principio de cercanía
- Las referencias a datos y programa dentro de un proceso tienden a agruparse
- La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento
- En cortos períodos de tiempo, el proceso necesitará pocas "piezas" del proceso (por ejemplo, una página de instrucciones y otra de datos...)



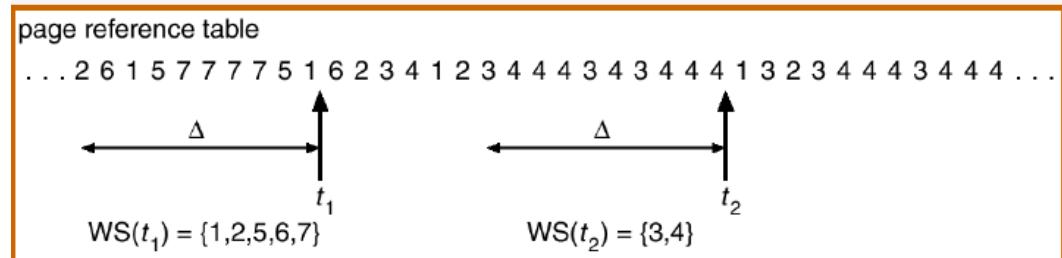
- Un programa se compone de varias localidades
- Ejemplo: Cada rutina será una nueva localidad: se refieren sus direcciones (cercanas) cuando se está ejecutando
- Para prevenir la hiperactividad, un proceso debe tener en memoria sus páginas más activas (menos page faults)

Modelo de working set

- Se basa en el modelo de localidad.
- Ventana del working set (Δ): las referencias a memoria más recientes.

- Working set: es el conjunto de páginas que tienen las más recientes Δ referencias a páginas
- Ejemplo:

$$\Delta = 10$$



- La selección del Δ
 - Δ chico: no cubrirá la localidad
 - Δ grande: puede tomar varias localidades

Medida del working set

- m = cantidad de frames disponibles
- WSS_i = tamaño del working set del proceso p_i
- $WSS_i = D$;
- D = demanda total de frames
- Si $D > m$, habrá thrashing

Prevención del thrashing

- SO monitorea c/ proceso, dándole tantos frames hasta su WSS_i (medida del working set del proceso p_i)
- Si quedan frames, puede iniciar otro proceso
- Si D crece, excediendo m , se elige un proceso para suspender, reasignándose sus frames...

Así, se mantiene alto el grado de multiprogramación optimizando el uso de la CPU.

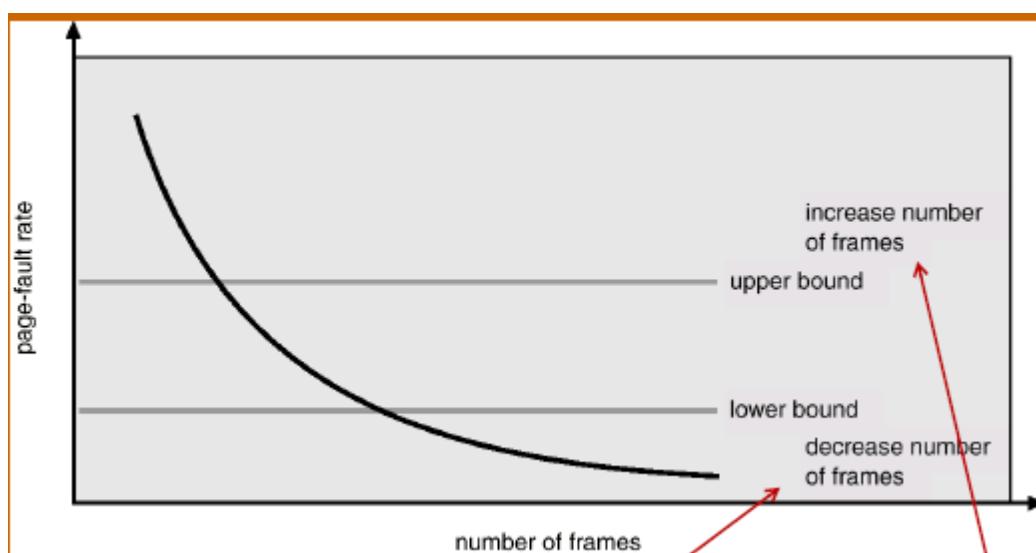
Problema del modelo del WS

- Mantener un registro de los WSS;
- La ventana es móvil

Prevención del thrashing por PFF

- La técnica PFF (Page Fault Frequency o Frecuencia de Fallo de Páginas), en lugar de calcular el WS de los procesos, utiliza la tasa de fallos de página para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS.
- PFF: Frecuencia de page faults
- PFF alta: se necesitan más frames
- PFF baja: Los procesos tienen frames asignados que le sobran

Esquema de PFF

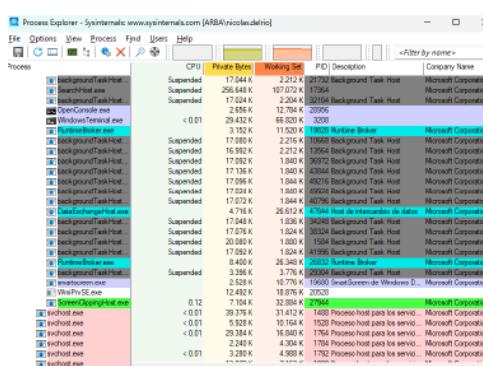


- Establecer tasa de PF aceptable
 - Si la tasa actual es baja, el proceso puede perder frames
 - Si la tasa actual es alta, el proceso gana frames

PFF (Continuación)

- Establecer límites superior e inferior de las PFF's deseadas.
- Si se Excede PFF max. Se le asignan mas frames al proceso, ya que el mismo genera muchos PF y probablemente los esté necesitando.
- Si la PFF está por debajo del mínimo. Se le pueden quitar frames al proceso para ser utilizados en los que necesitan mas.
- Se puede llegar a suspender un proceso si no hay más frames libres y todos los procesos están por arriba de PFF max

Herramientas



Permiten manipular:

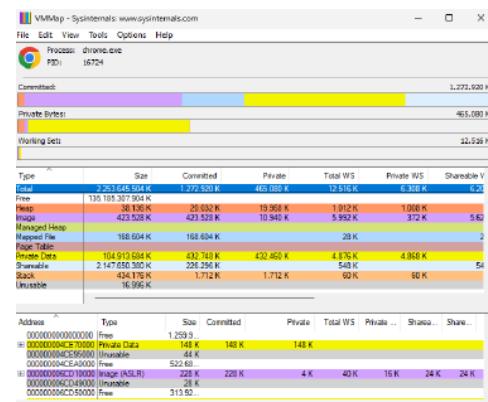
- Working Set
- Otros parámetros

En GNU/Linux:

- Cat /proc/<PID>/statm
- Cat /proc/<PID>/status

Permiten analizar información de los procesos:

- Working Set, Delta, Fallos de página, espacio virtual vs espacio físico ocupado en memoria, entre otros parámetros



PFF y estructura de un programa

- Ejemplo:
 - int A [][] = new int [1024][1024];
 - Cada fila se almacena en una página

- Programa 1:

```
for (j = 0; j < A.length; j++)
    for (i = 0; i < A.length; i++)
        A[i,j] = 0;
```

1024 X 1024 Page faults

- Programa 2

```
for (i = 0; i < A.length; i++)
    for (j = 0; j < A.length; j++)
        A[i,j] = 0;
```

1024 page faults

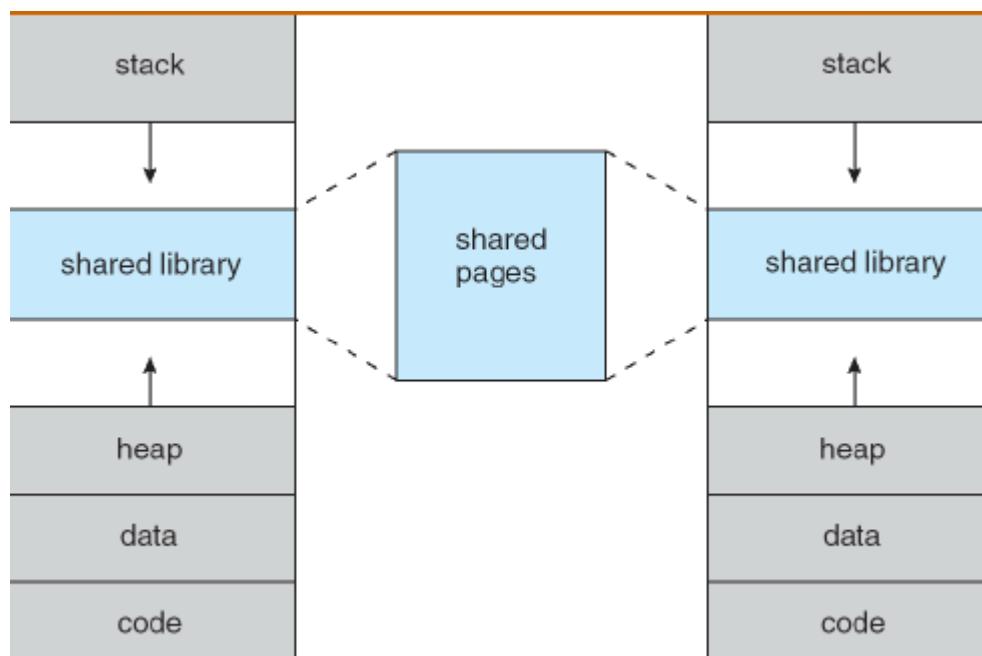
Demonio de Paginación

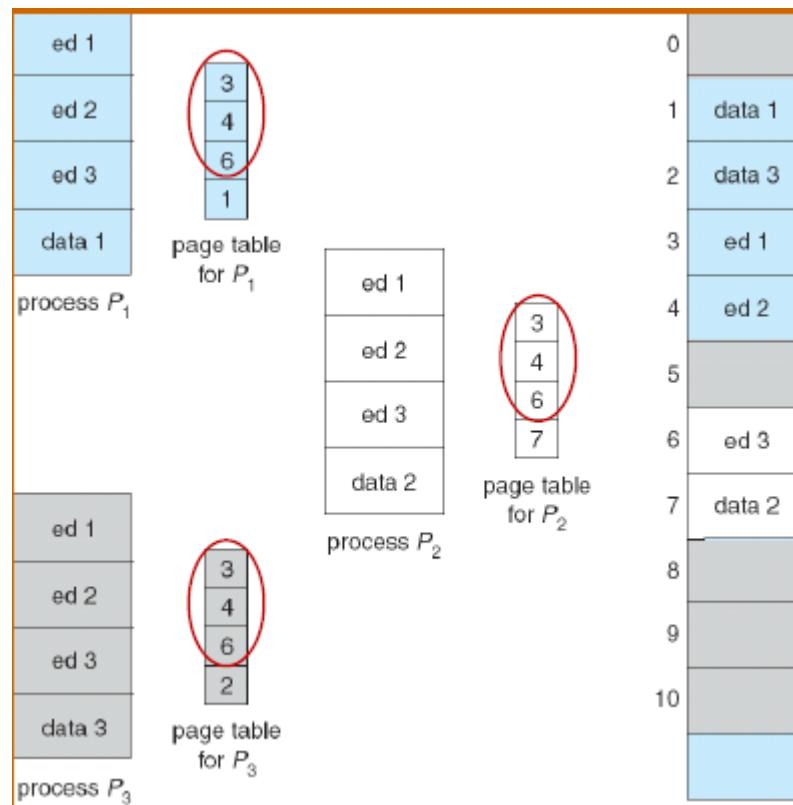
- Proceso creado por el SO durante el arranque que apoya a la administración de la memoria
- Se ejecuta cuando el sistema tienen una baja utilización o algún parámetro de la memoria lo indica
 - Poca memoria libre
 - Mucha memoria modificada
- Tareas:
 - Limpiar páginas modificadas sincronizándolas con el swap
 - Reducir el tiempo de swap posterior ya que las páginas están "limpias"
 - Reducir el tiempo de transferencia al sincronizar varias páginas contiguas
 - Mantener un cierto número de marcos libres en el sistema

- Demorar la liberación de una página hasta que haga falta realmente
- Ejemplos:
 - En linux: proceso "kswapd"
 - En windows: proceso "system"

Memoria compartida

- Gracias al uso de la tabla de páginas varios procesos pueden compartir un marco de memoria; para ello ese marco debe estar asociado a una página en la tabla de páginas de cada proceso
- El número de página asociado al marco puede ser diferente en cada proceso
- Código compartido
 - Los procesos comparten una copia de código (sólo lectura) por ejemplo: editores de texto, compiladores, etc
 - Los datos son privados a cada proceso y se encuentran en páginas no compartidas





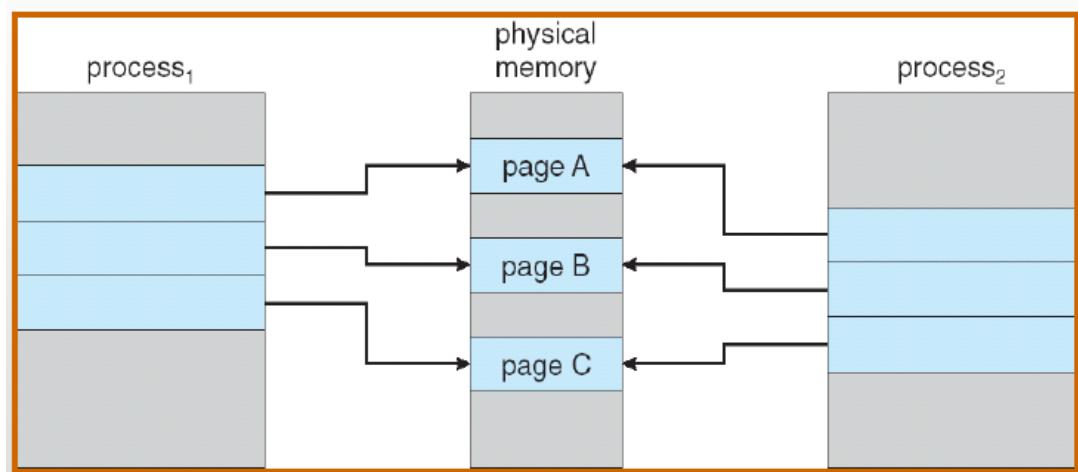
Mapeo de Archivo en Memoria

- Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales
- El contenido del archivo no se sube a memoria hasta que se generan Page Faults
- Cualquier modificación a la dirección de memoria es una modificación indirecta al archivo
- El contenido de la página que genera el PF es obtenido desde el archivo asociado
 - No del área de intercambio
- Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente
- Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el Sistema de Archivos
- Es utilizado comúnmente para asociar bibliotecas compartidas o DLLs

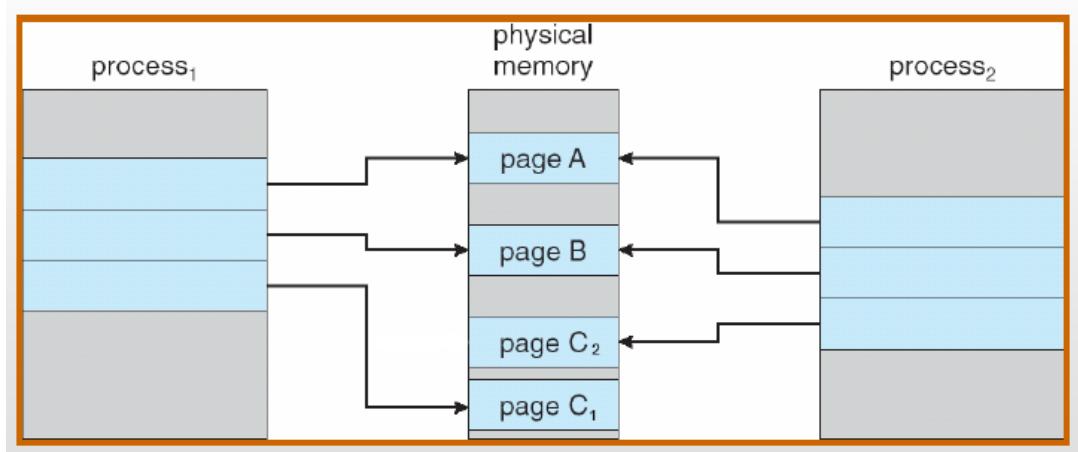
Copia en Escritura

- La copia en escritura (Copy-on-Write, COW) permite a los procesos compartir inicialmente las mismas páginas de memoria
 - Si uno de ellos modifica una página compartida la página es copiada
 - En fork, permite inicialmente que padre e hijo utilicen las mismas páginas sin necesidad de duplicación.
- COW permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas

El Proceso 1 Modifica la Página C (Antes)

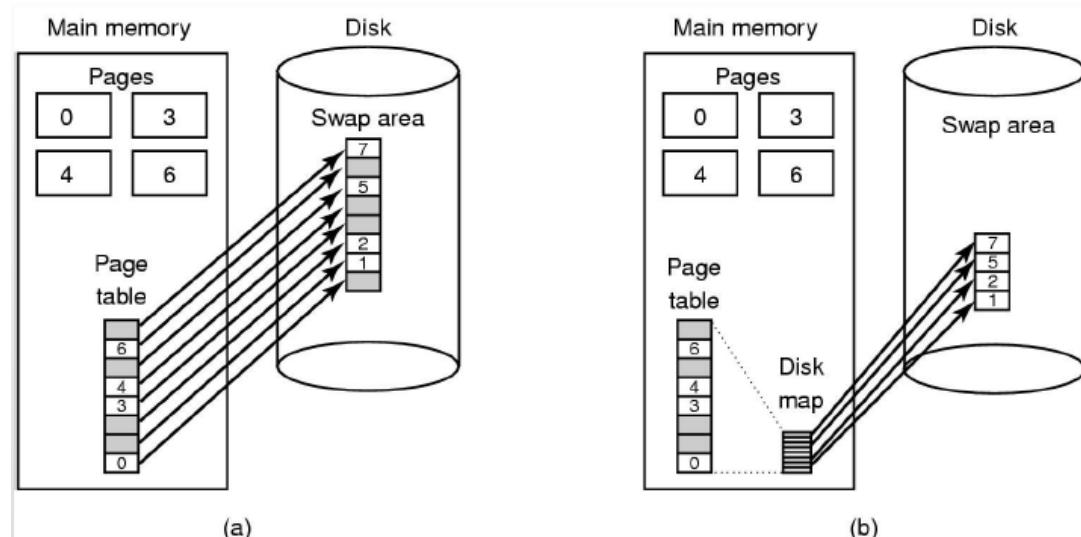


El Proceso 1 Modifica la Página C (Después)

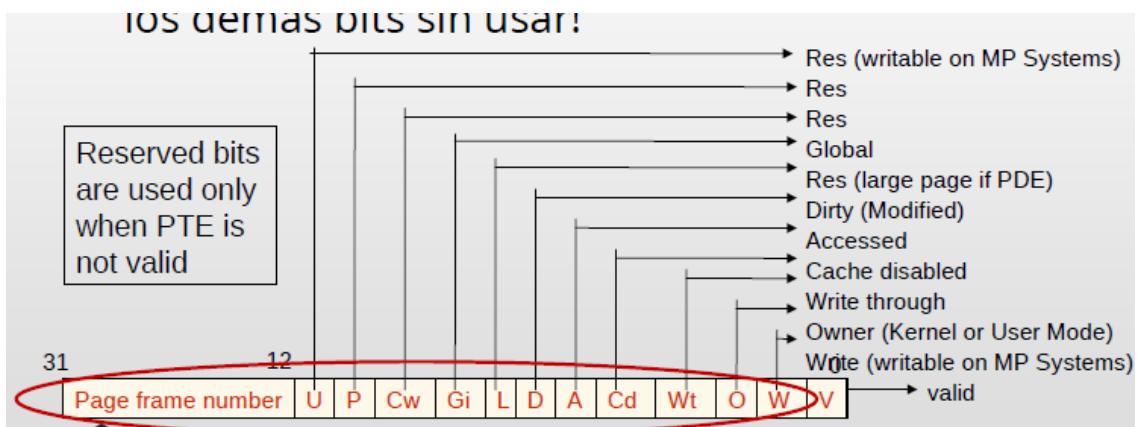


Área de Intercambio

- Sobre el área utilizada
 - Área dedicada, separada del Sistema de Archivos (por ejemplo, en Linux)
 - Un archivo dentro del Sistema de Archivos (por ejemplo, Windows)
- Técnicas para la Administración
 - a. Cada vez que se crea un proceso se reserva una zona del área de intercambio igual al tamaño de imagen del proceso. A cada proceso se le asigna la dirección en disco de su área de intercambio. La lectura se realiza sumando el número de página virtual a la dirección de comienzo del área asignada al proceso
 - b. No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la página vuelve a memoria. Problema: se debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco



- Cuando una página no está en memoria, sino en swap, como podríamos saber en que parte del área de intercambio está?
 - RTA: El PTE de dicha página tiene el bit v=0 y todos los demás bits sin usar



Área de Intercambio - Linux

- Permite definir un número predefinido de áreas de Swap
- swap_info es un arreglo que contiene estas estructuras

<linux/swap.h>

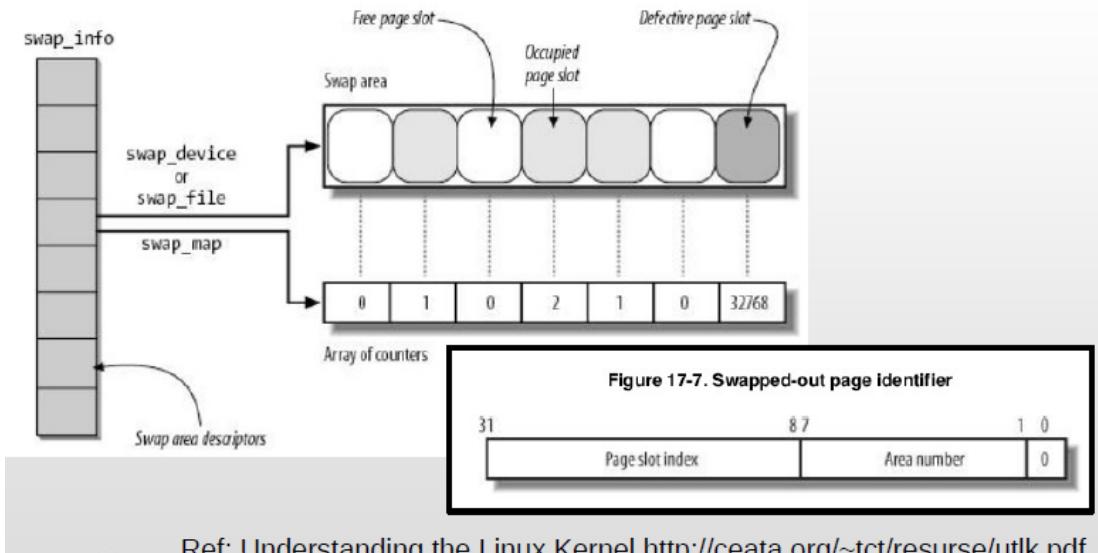
```

64 struct swap_info_struct {
65     unsigned int flags;
66     kdev_t swap_device;
67     spinlock_t sdev_lock;
68     struct dentry * swap_file;
69     struct vfsmount *swap_vfsmnt;
70     unsigned short * swap_map;
71     unsigned int lowest_bit;
72     unsigned int highest_bit;
73     unsigned int cluster_next;
74     unsigned int cluster_nr;
75     int prio;
76     int pages;
77     unsigned long max;
78     int next;
79 };

```

- Cada área es dividida en un número fijo de slots según el tamaño de la página
- Cuando una página es llevada a disco, Linux utiliza el PTE para almacenar 2 valores:
 - El número de área
 - El desplazamiento en el área (24 bits, lo que limita el tamaño máximo del área a 64gb)

Figure 17-6. Swap area data structures



Ref: Understanding the Linux Kernel <http://ceata.org/~tct/resource/utlk.pdf>

Subsistema de Entrada / Salida

Responsabilidades del SO

- Controlar dispositivos E/S
 - Generar comandos
 - Manejar interrupciones
 - Manejar errores
- Proporcionar una interfaz de utilización

Problemas

- Heterogeneidad de dispositivos
- Características de los dispositivos
- Velocidad
- Nuevos tipos de dispositivos

- Diferentes formas de realizar E/S

Aspectos de los dispositivos de I/O

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read&write	CD-ROM graphics controller disk

- Unidad de transferencia
 - Dispositivos por bloques (discos):
 - Operaciones: Read, Write, Seek
 - Dispositivos por Carácter (teclados, mouse, serial ports)
 - Operaciones: get, put
- Formas de Acceso
 - Secuencial o Aleatorio
 - Tipo de Acceso
 - Acceso compartido: Disco Rígido

- Acceso Exclusivo: Impresora
- Tipo de Acceso:
 - Read Only: CDROM
 - Write only: Pantalla
 - Read/Write: Disco
- Velocidad

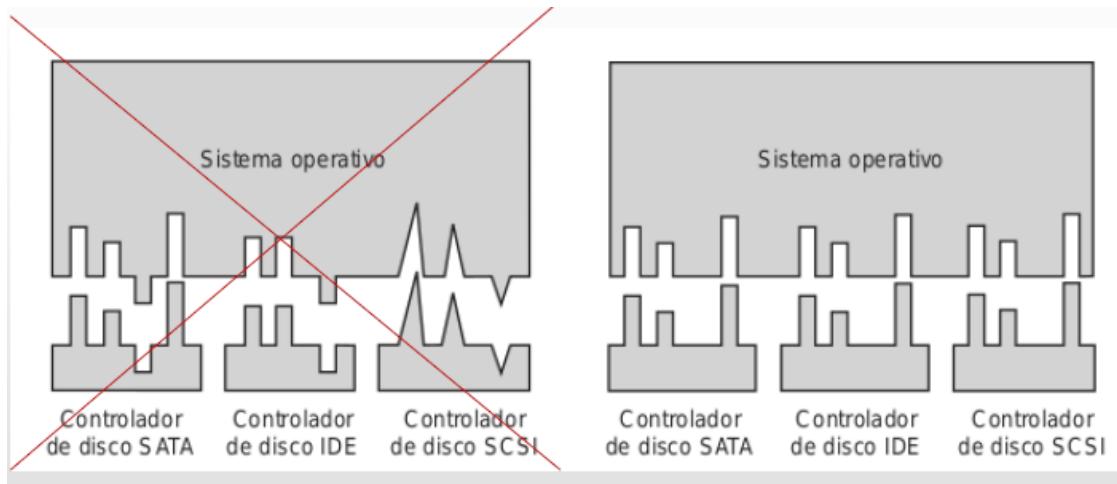
Dispositivo	Velocidad de transferencia de datos
Teclado	10 bytes/seg
Ratón	100 bytes/seg
Módem de 56K	7 KB/seg
Escáner	400 KB/seg
Cámara de video digital	3.5 MB/seg
802.11g inalámbrico	6.75 MB/seg
CD-ROM de 52X	7.8 MB/seg
Fast Ethernet	12.5 MB/seg
Tarjeta Compact Flash	40 MB/seg
FireWire (IEEE 1394)	50 MB/seg
USB 2.0	60 MB/seg
Red SONET OC-12	78 MB/seg
Disco SCSI Ultra 2	80 MB/seg
Gigabit Ethernet	125 MB/seg
Unidad de disco SATA	300 MB/seg
Cinta de Ultrium	320 MB/seg
Bus PCI	528 MB/seg

Metas, objetivos y Servicios

- Generalidad:
 - Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada
 - Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más "bajos" para que los procesos vean a los dispositivos, en

términos de operaciones comunes como: read, write, open, close, lock, unlock

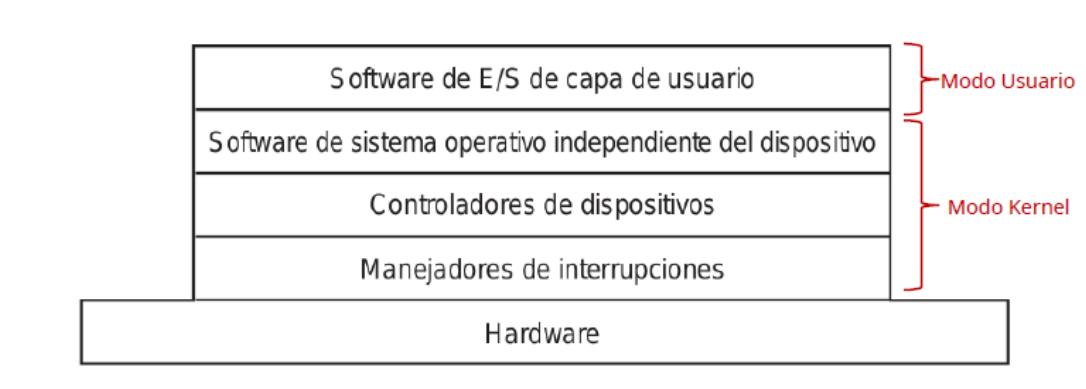
- Interfaz Uniforme



- Eficiencia
 - Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria y la CPU
 - El uso de multi-programación permite que un proceso espere por la finalización de su I/O mientras que otro proceso se ejecuta
- Planificación
 - Organización de los requerimientos a los dispositivos
 - Ej: Planificación de requerimientos a disco para minimizar tiempos
- Buffering - Almacenamiento de los datos en memoria mientras se transfieren
 - Solucionar problemas de velocidad entre los dispositivos
 - Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos
- Caching - Mantener en memoria copia de los datos de reciente acceso para mejorar performance
- Spooling - Administrar la cola de requerimientos de un dispositivo

- Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo: por ej. impresora
- Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo
- Reserva de dispositivos: Acceso exclusivo
- Manejo de Errores
 - El S.O. debe administrar errores ocurridos (lectura de un disco, dispositivo no disponible, errores de escritura)
 - La mayoría retorna un número de error o código cuando la I/O falla
 - Logs de errores
- Formas de realizar I/O
 - Bloqueante: El proceso se suspende hasta que el requerimiento de I/O se completa
 - Fácil de usar y entender
 - No es suficiente bajo algunas necesidades
 - No bloqueante: El requerimiento de I/O retorna cuando es posible
 - Ejemplo: Interfaz de usuario que recibe input desde el teclado/mouse y se muestra screen
 - Ejemplo: Aplicación de video que lee frames desde un archivo mientras va mostrandolo en pantalla

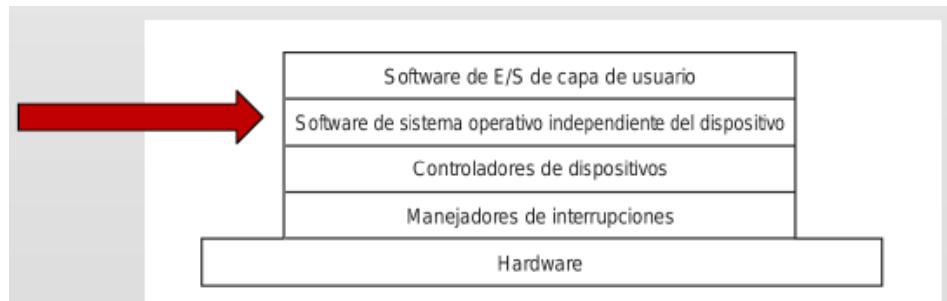
Diseño



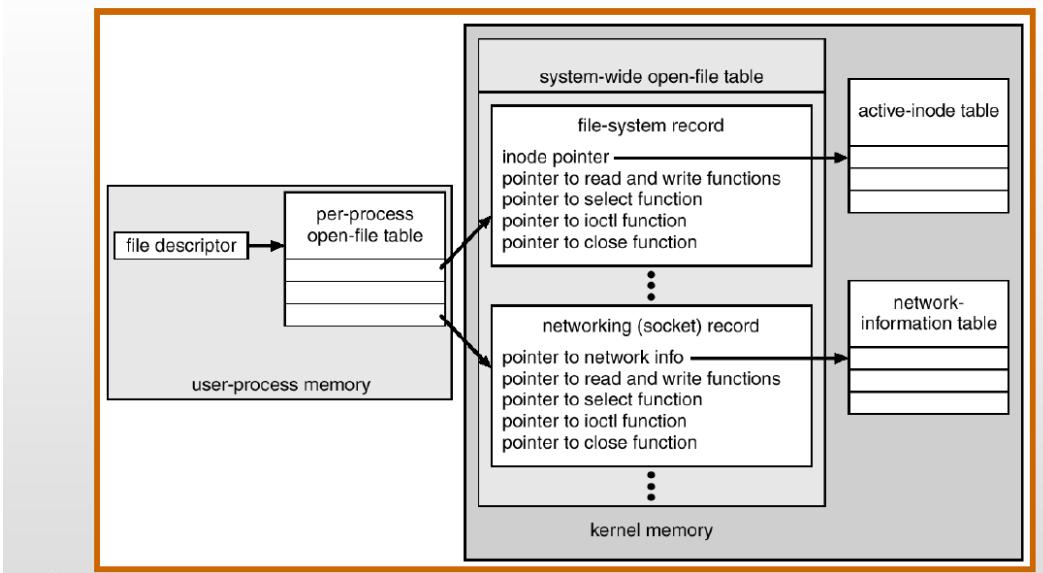
- Librerías de funciones
 - Permiten acceso a SysCalls
 - Implementan servicios que no dependen del kernel
- Procesos de apoyo
 - Demonio de Impresión (spooling)

Software Independiente SO

- Brinda los principales servicios de E/S antes vistos
 - Interfaz uniforme
 - Manejo de errores
 - Buffer
 - Asignación de Recursos
 - Planificación
- El Kernel mantiene la información de estado de cada dispositivo o componente
 - Archivos abiertos
 - Conexiones de red
 - Etc
- Hay varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc



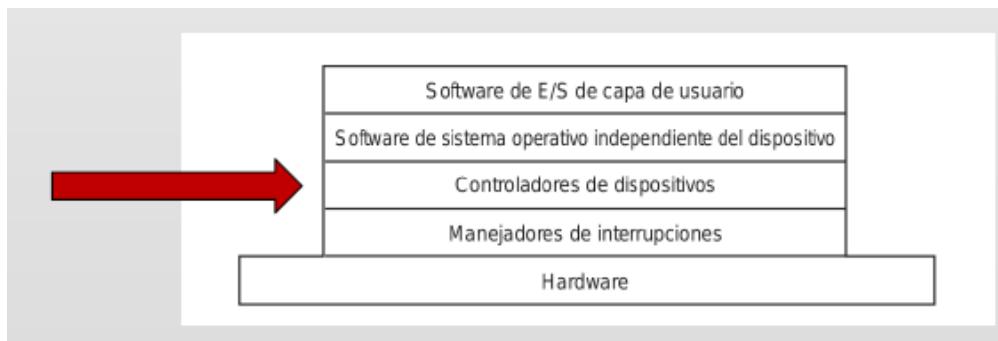
UNIX I/O Kernel Structure



Controladores (Drivers)

- Contienen el código dependiente del dispositivo
- Manejan un tipo de dispositivo
- Traducen los requerimientos abstractos en los comandos para el dispositivo
 - Escribe sobre los registros del controlador
 - Acceso a la memoria mapeada
 - Encola requerimientos
- Comúnmente las interrupciones generadas por los dispositivos son atendidas por funciones provistas por el driver
- Interfaz entre el SO y el Hard
- Forman parte del espacio de memoria del kernel
 - En general se cargan como modulos

- Los fabricantes de HW implementan el driver en función de una API especificada por el SO
 - open(), close(), read(), write(), etc
- Para agregar nuevo HW sólo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel



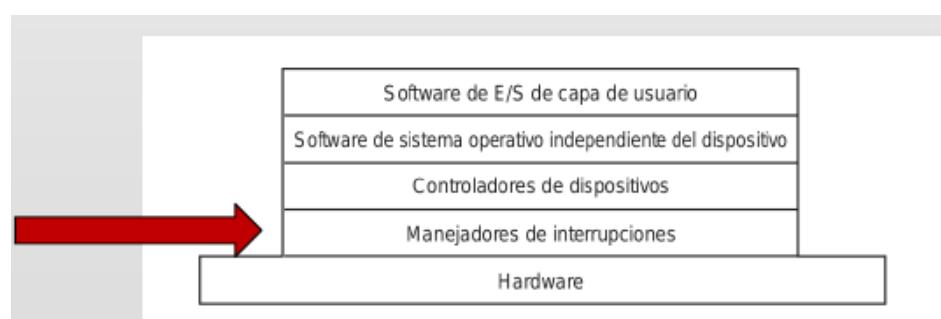
Ejemplo en Linux:

- Linux distingue 3 tipos de dispositivos
 - Carácter: I/O programada o por interrupciones
 - Bloque: DMA
 - Red: Ports de comunicaciones
- Los Drivers se implementan como módulos
 - Se cargan dinámicamente
- Debe tener al menos estas operaciones:
 - init_module: para instalarlo
 - cleanup_module: Para desinstalarlo
- Operaciones que debe contener para I/O
 - Open: abre el dispositivo
 - release: cerrar el dispositivo
 - read: leer bytes del dispositivo
 - write: escribir bytes en el dispositivo
 - ioctl: orden de control sobre el dispositivo
- Otras operaciones menos comunes

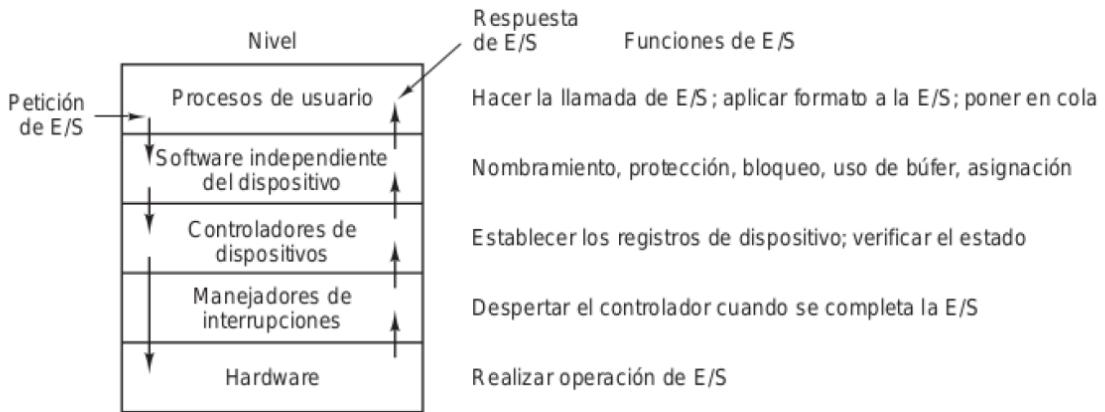
- lseek: posicionar el puntero de lectura/escritura
- flush: volcar los búferes al dispositivo}
- poll: preguntar si se puede leer o escribir
- mmap: mapear el dispositivo en memoria
- fsync: sincronizar el dispositivo
- fasync: notificación de operación asíncrona
- lock: reservar el dispositivo
-
- Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo
- Acceso al hardware
 - Funciones para acceso a los puertos I/O
- Leen o Escriben un byte en el puerto de E/S indicado

Gestor de interrupciones

- Atiende todas las interrupciones del HW
- Deriva al driver correspondiente según interrupción
- Resguarda información
- Independiente del driver



Ciclo de atención de un Requerimiento



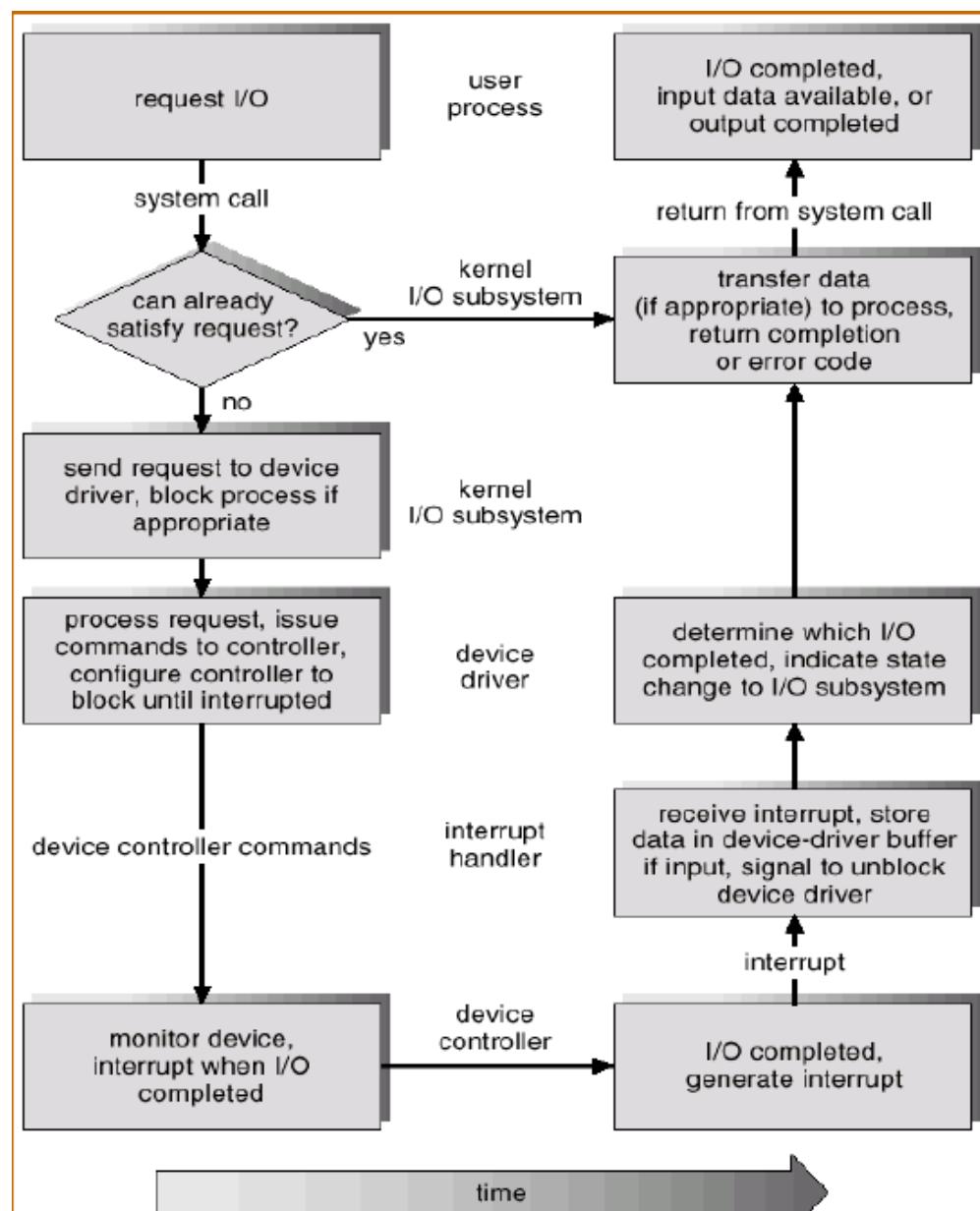
Desde el requerimiento de I/O hasta el hardware

- Consideraremos la lectura sobre un archivo en un disco:
 - Determinar el dispositivo que almacena los datos
 - Traducir el nombre del archivo en la representación del dispositivo
 - Traducir requerimiento abstracto en bloques de disco (Filesystem)
 - Realizar la lectura física de los datos (bloques) en la memoria
 - Marcar los datos como disponibles al proceso que realizó el requerimiento
 - Desbloquearlo
 - Retornar control al proceso

<pre>nico@yoko:~\$ ls -l /dev/sd* brw-rw---- 1 root disk 8, 0 oct 28 11:32 /dev/sda brw-rw---- 1 root disk 8, 1 oct 28 11:32 /dev/sda1 brw-rw---- 1 root disk 8, 2 oct 28 11:32 /dev/sda2 brw-rw---- 1 root disk 8, 5 oct 28 11:32 /dev/sda5 brw-rw---- 1 root disk 8, 16 oct 28 15:49 /dev/sdb brw-rw---- 1 root disk 8, 17 oct 28 15:49 /dev/sdb1 nico@yoko:~\$</pre>	<pre>nico@yoko:~\$ cat /proc/devices Character devices: 1 mem 4 /dev/vc/0 4 tty 4 ttys 5 /dev/tty 5 /dev/console 5 /dev/ptmx 5 ttyprintk</pre>	<p>THE I/O SUBSYSTEM</p> <table border="1"> <thead> <tr> <th colspan="4">block device switch table</th> </tr> <tr> <th>entry</th> <th>open</th> <th>close</th> <th>strategy</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>gdopen</td> <td>gdclose</td> <td>gdstrategy</td> </tr> <tr> <td>1</td> <td>gtopen</td> <td>gtclose</td> <td>gtstrategy</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="6">character device switch table</th> </tr> <tr> <th>entry</th> <th>open</th> <th>close</th> <th>read</th> <th>write</th> <th>ioctl</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>conopen</td> <td>conclose</td> <td>conread</td> <td>conwrite</td> <td>coniocctl</td> </tr> <tr> <td>1</td> <td>dzbopen</td> <td>dzbclose</td> <td>dzbread</td> <td>dzbwrite</td> <td>dzbioctl</td> </tr> <tr> <td>2</td> <td>syopen</td> <td>nulldev</td> <td>syread</td> <td>sywrite</td> <td>syiocctl</td> </tr> <tr> <td>3</td> <td>nulldev</td> <td>nulldev</td> <td>mmrread</td> <td>mmrwrite</td> <td>nodev</td> </tr> <tr> <td>4</td> <td>gdopen</td> <td>gdclose</td> <td>gdread</td> <td>gdwrite</td> <td>nodev</td> </tr> <tr> <td>5</td> <td>gtopen</td> <td>gtclose</td> <td>gtread</td> <td>gtwrite</td> <td>nodev</td> </tr> </tbody> </table>	block device switch table				entry	open	close	strategy	0	gdopen	gdclose	gdstrategy	1	gtopen	gtclose	gtstrategy	character device switch table						entry	open	close	read	write	ioctl	0	conopen	conclose	conread	conwrite	coniocctl	1	dzbopen	dzbclose	dzbread	dzbwrite	dzbioctl	2	syopen	nulldev	syread	sywrite	syiocctl	3	nulldev	nulldev	mmrread	mmrwrite	nodev	4	gdopen	gdclose	gdread	gdwrite	nodev	5	gtopen	gtclose	gtread	gtwrite	nodev
block device switch table																																																																		
entry	open	close	strategy																																																															
0	gdopen	gdclose	gdstrategy																																																															
1	gtopen	gtclose	gtstrategy																																																															
character device switch table																																																																		
entry	open	close	read	write	ioctl																																																													
0	conopen	conclose	conread	conwrite	coniocctl																																																													
1	dzbopen	dzbclose	dzbread	dzbwrite	dzbioctl																																																													
2	syopen	nulldev	syread	sywrite	syiocctl																																																													
3	nulldev	nulldev	mmrread	mmrwrite	nodev																																																													
4	gdopen	gdclose	gdread	gdwrite	nodev																																																													
5	gtopen	gtclose	gtread	gtwrite	nodev																																																													

Figure 10.2. Sample Block and Character Device Switch Tables

Ciclo de vida de un requerimiento de I/O



Performance

- I/O es uno de los factores que mas afectan a la performance del sistema:
 - Utiliza mucho la CPU para ejecutar los drivers y el codigo del subsistema de I/O

- Provoca Context switches ante las interrupciones y bloqueos de los procesos
- Utiliza el bus de mem. en copia de datos:
 - Aplicaciones (espacio usuario) – Kernel
 - Kernel (memoria física) - Controladora

Mejorar la Performance

- Reducir el número de context switches
- Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación
- Reducir la frecuencia de las interrupciones, utilizando:
 - Transferencias de gran cantidad de datos
 - Controladoras más inteligentes
 - Polling, si se minimiza la espera activa.
- Utilizar DMA

Variedad en los dispositivos de I/O

- Legible para el usuario
 - Usados para comunicarse con el usuario
 - Impresoras, terminales: Pantalla, teclado, Mouse
- Legible para la máquina
 - Utilizados para comunicarse con los componentes electrónicos
 - Discos, Cintas, Sensores, etc.
- Comunicación
 - Usados para comunicarse con dispositivos remotos
 - Líneas Digitales, Modems, Interfaces de red, etc.

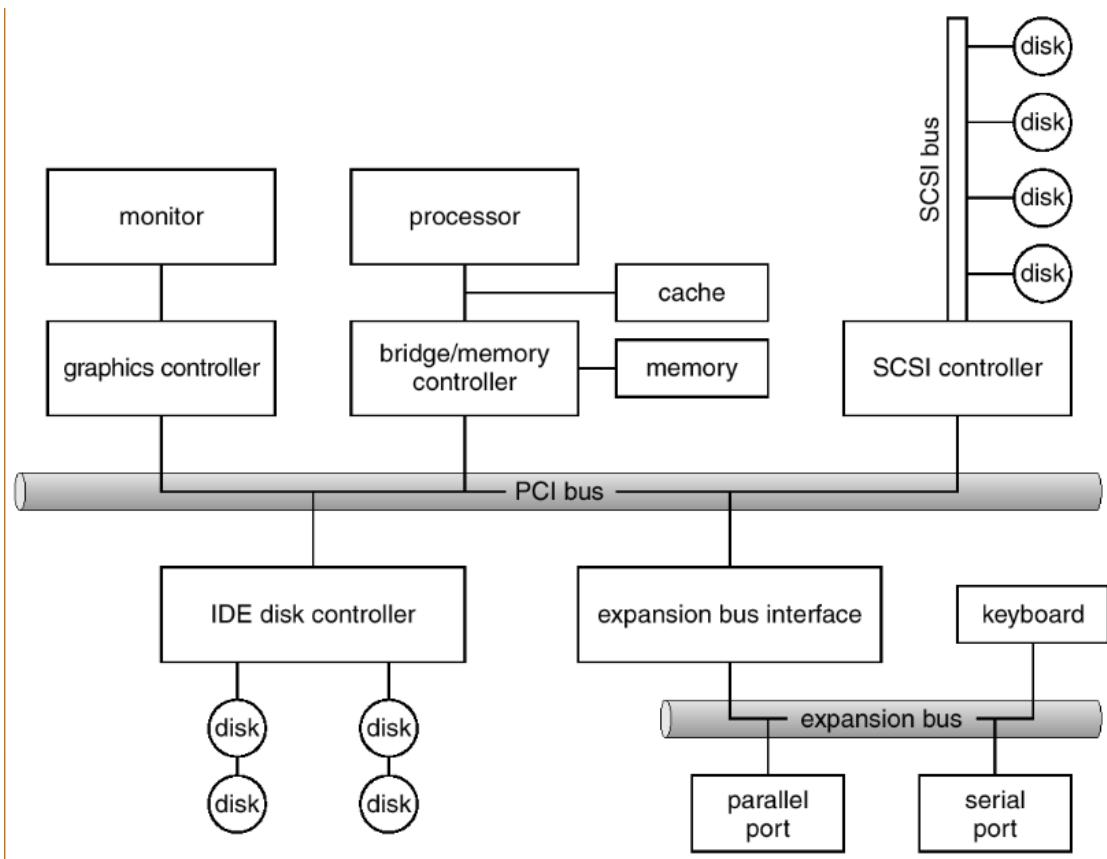
Problemas que surgen

- Amplia Variedad
 - Manejan diferentes cantidad de datos
 - En velocidades diferentes
 - En formatos diferentes
- La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM

Hardware y software involucrado

- Buses
- Controladores
- Dispositivos
- Puertos de E/S - Registros
- Drivers
- Comunicación con controlador del dispositivo: I/O Programada, Interrupciones, DMA

Estructura de BUS de una PC



Comunicación: CPU - Controladora

- ¿Cómo puede la CPU ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo?
 - La controladora tiene uno o más registros
 - Registros para señales de control
 - Registros de datos
- La CPU se comunica con la controladora escribiendo y leyendo en dichos registros

Comandos de I/O

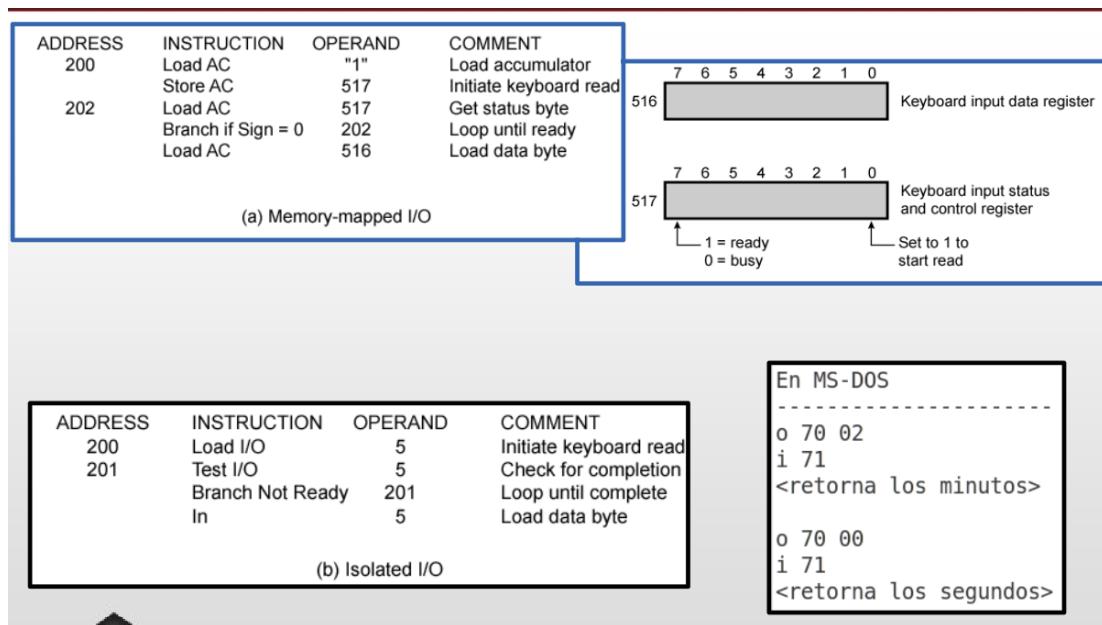
- CPU emite direcciones
 - Para identificar el dispositivo

- CPU emite comandos
 - Control - Que hacer?
 - Ej. Girar el disco
 - Test - controlar estado
 - Ej. Power? Error?
 - Read/Write
 - Trasferir información desde/hacia el dispositivo

Mapeo de E/S y E/S aislada

- Correspondencia en memoria (Memory mapped I/O)
 - Dispositivos y memoria comparten el espacio de direcciones
 - I/O es como escribir/leer en la memoria
 - No hay instrucciones especiales para I/O
 - Ya se dispone de muchas instrucciones para la memoria
- Isolated I/O (Aislada, uso de puertos de E/S)
 - Espacio separado de direcciones
 - Se necesitan líneas de I/O. Puertos E/S
 - Instrucciones especiales
 - Conjunto limitado

Memory mapped and Insolated I/O



Técnicas de I/O

Programada

- CPU tiene control directo sobre la I/O
 - Controla el stado
 - Comandos para leer y escribir
 - Transfiere datos
- CPU espera que el componente de I/O complete la operación
- Se desperdician ciclos de CPU

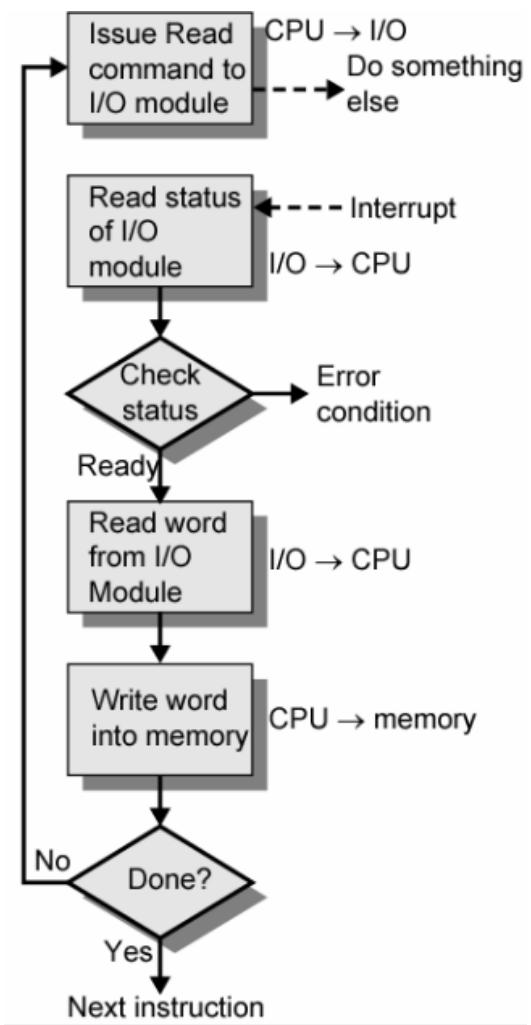
Polling

- En la I/O programada, es necesario hacer polling del dispositivo para determinar el estado del mismo
 - Listo para recibir comandos
 - Ocupado

- Error
- Ciclo de “Busy-wait” para realizar la I/O
- Puede ser muy costoso si la espera es muy larga

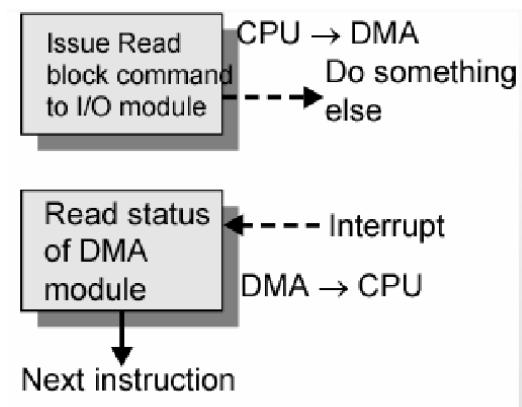
Manejada por interrupciones

- Soluciona el problema de la espera de la CPU
- La CPU no repite el chequeo sobre el dispositivo
- El procesador continúa la ejecución de instrucciones
- El componente de I/O envía una interrupción cuando termina



DMA (Direct Memory Access)

- Un componente de DMA controla el intercambio de datos entre la memoria principal y el dispositivo
- El procesador es interrumpido luego de que el bloque entero fue transferido



Pasos para una transferencia:

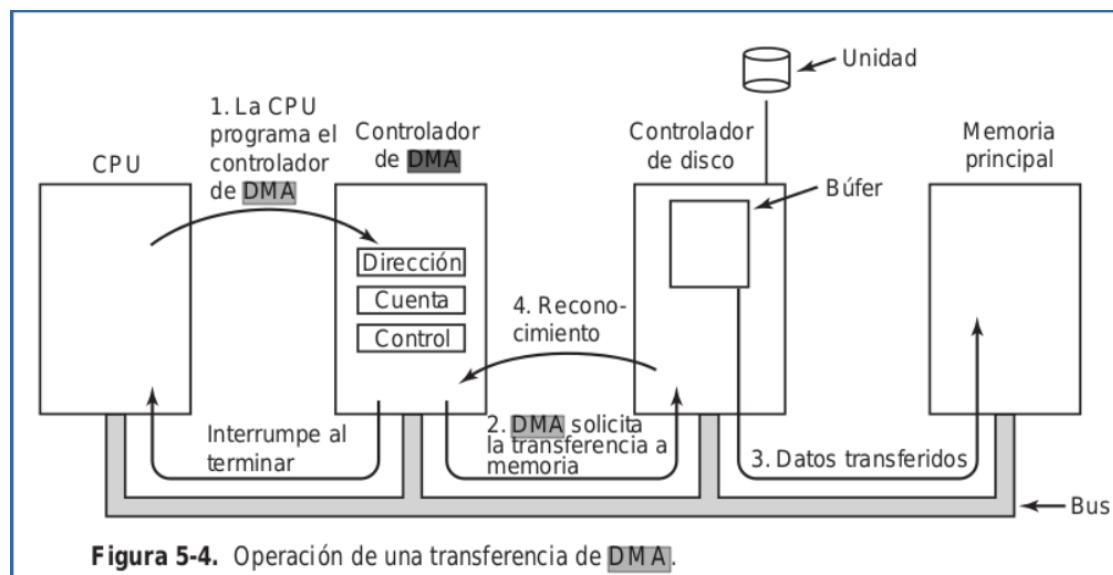


Figura 5-4. Operación de una transferencia de DMA.

Administración de Archivos

¿Por qué necesitamos archivos?

- Almacenar grandes cantidades de datos
- Tener almacenamiento a largo plazo
- Permitir a distintos procesos acceder al mismo conjunto de información

Archivo

- Entidad abstracta con nombre
- Espacio lógico continuo y direccionable
- Provee a los programas de datos (entrada)
- Permite a los programas guardar datos (salida)
- El programa mismo es información que debe guardarse

Archivos - punto de vista del usuario

- Que operaciones se pueden llevar a cabo
- Como nombrar a un archivo
- Como asegurar la protección
- Como compartir archivos
- No tratar con aspectos físicos
- Etc.

Archivos - Punto de vista de diseño

- Implementar archivos
- Implementar directorios
- Manejo del espacio en disco
- Manejo del espacio libre
- Eficiencia y mantenimiento

Sistemas de manejo de archivos

- Conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos
 - Crear
 - Borrar
 - Buscar
 - Copiar
 - Leer
 - Escribir
 - Etc.
- Facilita el acceso a los archivos por parte de las aplicaciones
- Permite la abstracción al programador, en cuanto al acceso de bajo nivel (el programador no desarrolla el soft de administración de archivos)

Objetivos del SO en cuanto a archivos

- Cumplir con la gestión de datos
- Cumplir con las solicitudes del usuario
- Minimizar / eliminar la posibilidades de perder o destruir datos
 - Garantizar la integridad del contenido de los archivos
- Dar soporte de E/S a distintos dispositivos
- Brindar un conjunto de interfaces de E/S para tratamiento de archivos

Tipos de Archivos

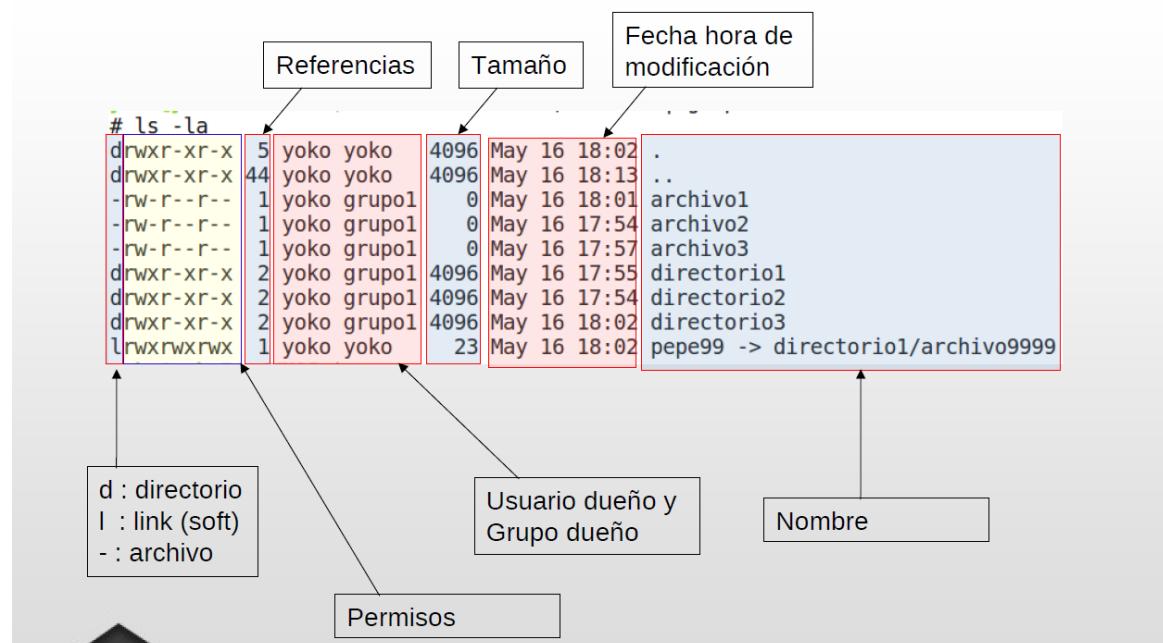
- Archivos Regulares
 - Texto plano
 - Source file
 - Binarios
 - Object File

- Executable File
- Directorios
 - Archivos que mantienen la estructura en el FileSystem

Atributos de un Archivo

- Nombre
- Identificador
- Tipo
- Localización
- Tamaño
- Protección, Seguridad y Monitoreo
 - Owner, Permisos, Password
 - Momento en que el usuario lo modifico, creo, accedio por ultima vez
 - ACLs

Ej: Tipos de archivos y atributos



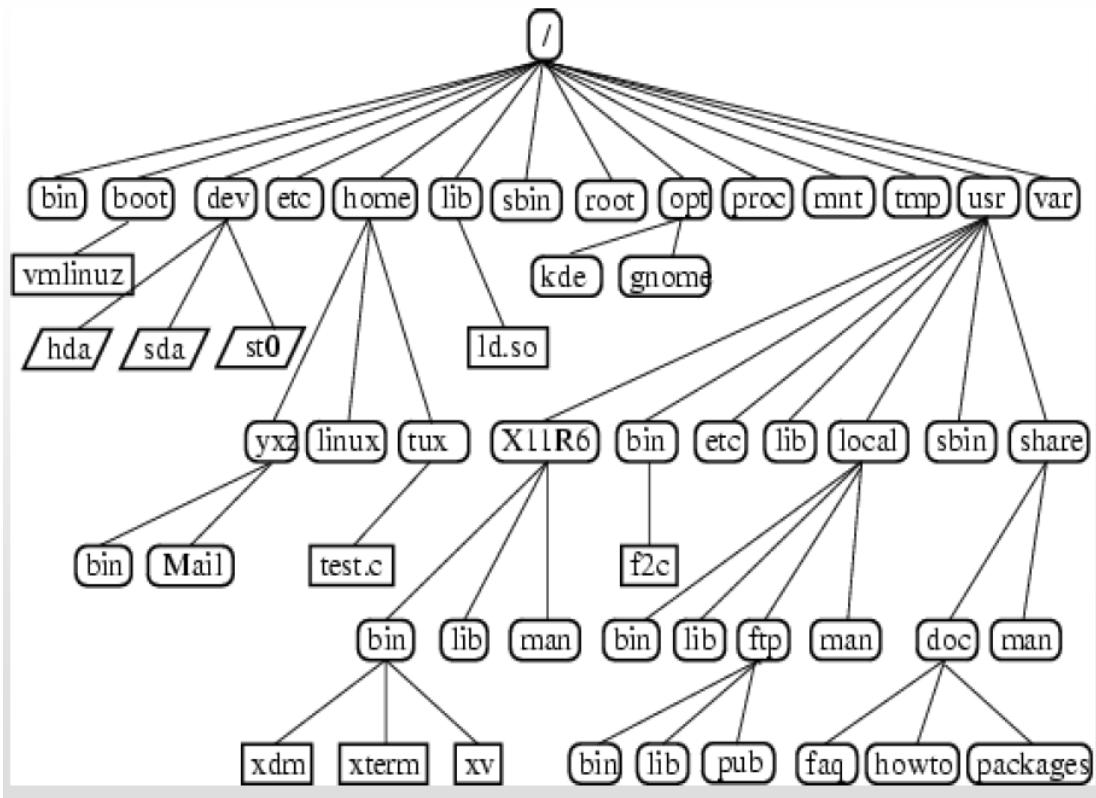
Directarios

- Contiene información acerca de archivos y directorios que están dentro de él
- El directorio es, en si mismo, un archivo
- Interviene en la resolución entre el nombre y el archivo mismo
- Operaciones en directorios:
 - Buscar un archivo
 - Crear un archivo (entrada de directorio)
 - Borrar un archivo
 - Listar el contenido
 - Renombrar archivos
 - Etc.

Directarios de Archivos

- El uso de los directorios ayuda con:
 - La eficiencia: Localización rápida de archivos
 - Uso del mismo Nombre de archivo:
 - Diferentes usuarios pueden tener el mismo nombre de archivo
 - Agrupación: Agrupación lógica de archivos por propiedades/funciones:
 - Ejemplo: Programas Java, Juegos, Librerías, etc.

Estructura de Dir. Jerárquica o Árbol



Estructura de Directorios

- Los archivos pueden ubicarse siguiendo un path desde el directorio raíz y sus sucesivas referencias (full pathname del archivo o PATH absoluto)
- Distintos archivos pueden tener el mismo nombre pero el fullpathname es único
- El directorio actual se lo llama “directorio de trabajo (working directory)”
- Dentro del directorio de trabajo, se pueden referenciar los archivos tanto por su PATH absoluto como por su PATH relativo indicando solamente la ruta al archivo desde directorio de trabajo

Identificación absoluta y relativa

Tanto archivos como directorios se pueden identificar de manera:

- Absoluta. El nombre incluye todo el camino del archivo:
 - /var/www/index.html

- C:/windows/winhelp.exe
- Relativa. El nombre se calcula relativamente al directorio en el que esté
 - Si estoy en el directorio /var/spool/mail/
 - Entonces es: ../../www/index.html

Compartir archivos

- En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos
- Debe ser realizado bajo un esquema de protección:
 - Derechos de acceso
 - Manejo de accesos simultáneos

Protección

- El propietario/administrador debe ser capaz de controlar:
 - Que se puede hacer
 - Derechos de acceso
 - Quien lo puede hacer

Derechos de acceso

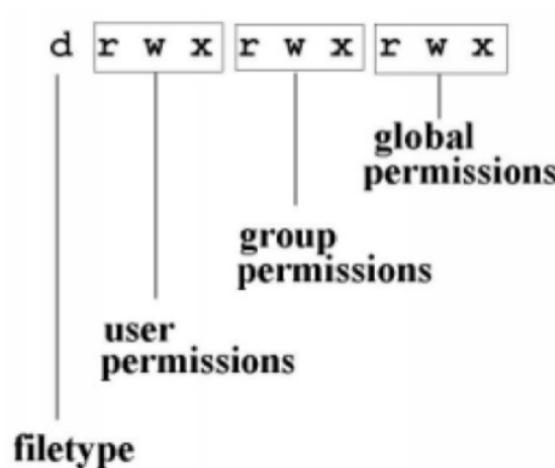
- Los directorios también tienen permisos, los cuales pueden permitir el acceso al mismo para que el usuario pueda usar el archivo siempre y cuando tenga permisos.
- Execution
 - El usuario puede ejecutar

- Reading
 - El usuario puede leer el archivo
- Appending
 - El usuario puede agregar datos pero no modificar o borrar el contenido del archivo
- Updating
 - El usuario puede modificar, borrar y agregar datos. Incluye la creación de archivos, sobreescribirlo y remover datos
- Changing protection
 - El usuario puede modificar los derechos de acceso
- Deletion
 - El usuario puede borrar el archivo
- Owners (propietarios)
 - Tiene todos los derechos
 - Puede dar derechos a otros usuarios. Se determinan clases:
 - Usuario específico
 - Grupos de usuarios
 - Todos (archivos públicos)

Ejemplo - Protección en UNIX

- Derechos de acceso son definidos independientemente para:
 - (u) user -Owner (creator) of a file
 - (g) group -Group
 - (o) other -all other users of the UNIX system
- Derechos de Acceso:
 - (r) Read access right; List right for directory
 - (w) Write access right; Includes delete/append rights
 - (x) Execute access right; Traverse right for directories

- Binary representation:
 - (x): Bit 0 (+1)
 - (w): Bit 1 (+2)
 - (r): Bit 2(+4)
- Rights can be combined
 - Read+Write access right: 6
 - Read+Execute access right: 3
 - Read-only: 2
- Los permisos que se pueden dar o quitar son:
 - r - de lectura
 - w - de escritura
 - x - de ejecución



```
$ ls -l
drwxrwxr-x 4 www     www     ..
-rw-rw-r-- 1 www     www     x_windows.tex
lrwxrwxrwx 1 lee     lee     img -> ../linux/img/
-rw-rw-r-- 1 lee     lee     test.log
```

Protección en Windows



Metas del sistema de Archivos

- Brindar espacio en disco a los archivos de usuario del sistema
- Mantener un registro del espacio libre. Cantidad y ubicación del mismo dentro del disco

Conceptos

- Sector
 - Unidad de almacenamiento utilizada en los discos rígidos
- Bloque/cluster
 - Conjunto de sectores consecutivos
- File System
 - Define la forma en que los datos son almacenados
- FAT: File allocation table

- Contiene información sobre en qué lugar están alojados los distintos archivos

Pre-asignación

- Se necesita saber cuánto espacio va a ocupar el archivo en el momento de su creación
- Se atiende a definir espacios mucho más grandes que lo necesario
- Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo
- ¿Qué pasa cuando el archivo supera el espacio asignado?
- Esta técnica suele usar la forma de asignación continua (podría usar otras también)

Asignación Dinámica

- El espacio se solicita a medida que se necesita
- Los bloques de datos pueden quedar de manera no contigua

Formas de asignación

Continua

- Conjunto continuo de bloques son utilizados
- Se requiere una pre-asignación
 - Se debe conocer el tamaño del archivo durante su creación
- File Allocation Table (FAT) es simple
 - Sólo una entrada que incluye Bloque de inicio y longitud
- El archivo puede ser leído con una única operación

- Puede existir fragmentación externa
 - Compactación

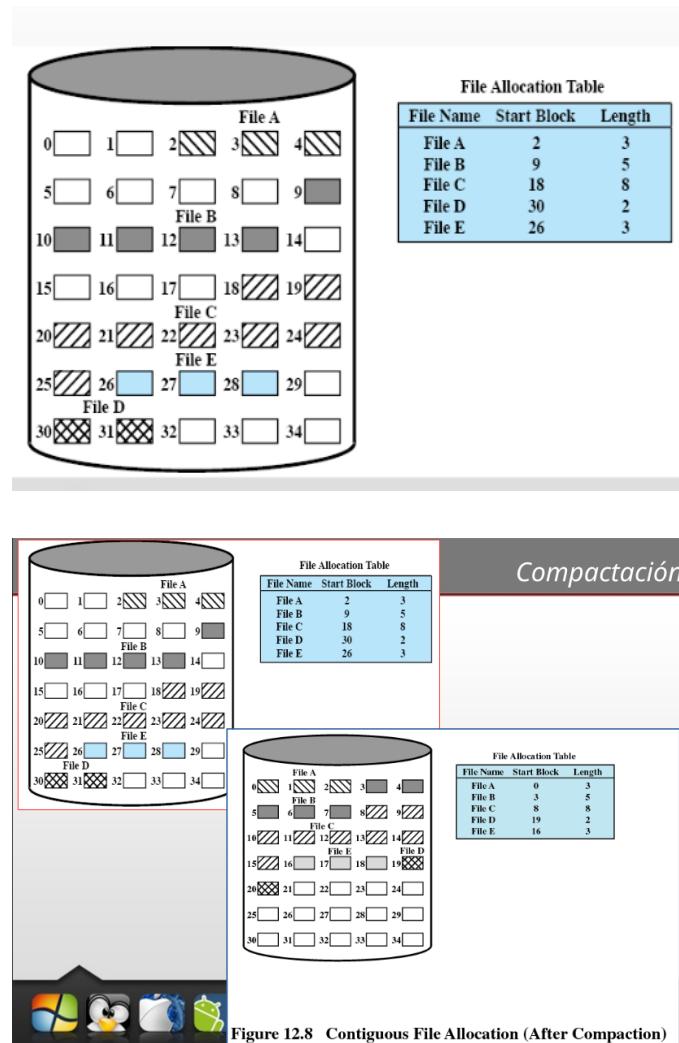


Figure 12.8 Contiguous File Allocation (After Compaction)

- Problemas de la técnica
 - Encontrar bloques libres contiguos en el disco
 - Incremento del tamaño de un archivo

Encadenada

- Asignación en base a bloques individuales
- Cada bloque tiene un puntero al próximo bloque del archivo
- File allocation table
 - Única entrada por archivo: Bloque de inicio y tamaño del archivo

- No hay fragmentación externa
- Útil para acceso secuencial (no random)
- Los archivos pueden crecer bajo demanda
- No se requieren bloques contiguos

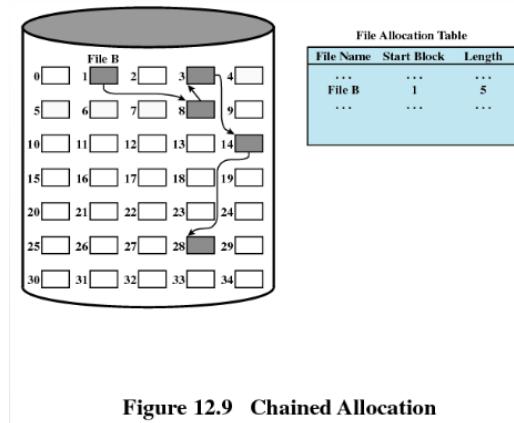
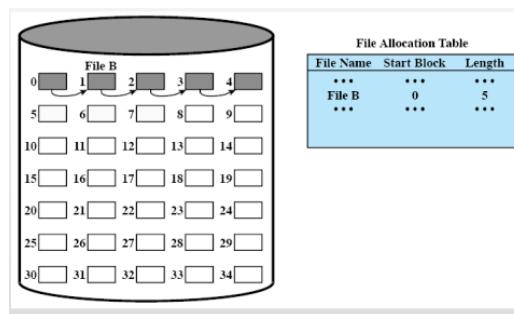


Figure 12.9 Chained Allocation

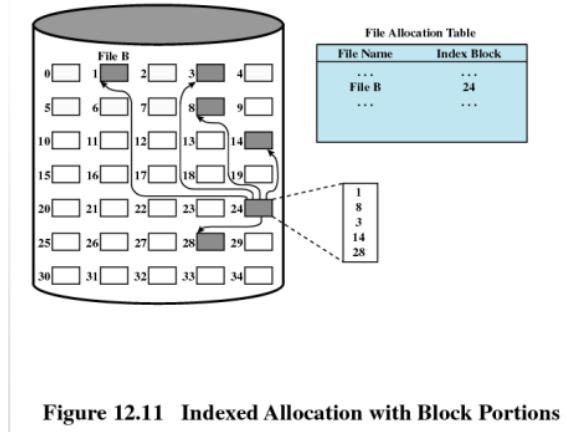
- Se pueden consolidar bloques de un mismo archivo para garantizar la cercanía de bloques de un mismo archivo



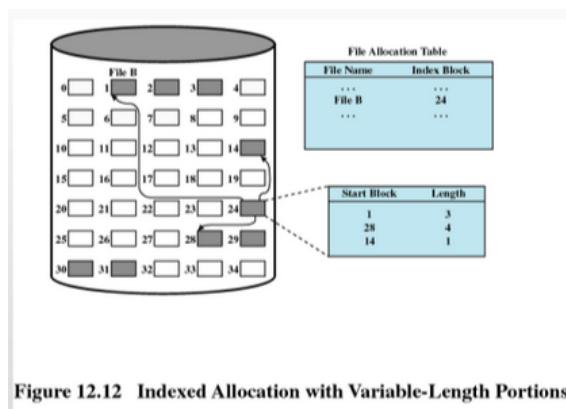
Indexada

- La FAT contiene un puntero al bloque índice
- El bloque índice no contiene datos propios del archivo, sino que contienen un índice a los bloques que lo componen
- Asignación en base a bloques individuales
- No se produce Fragmentación Externa
- El acceso "random" a un archivo es eficiente
- File Allocation Table

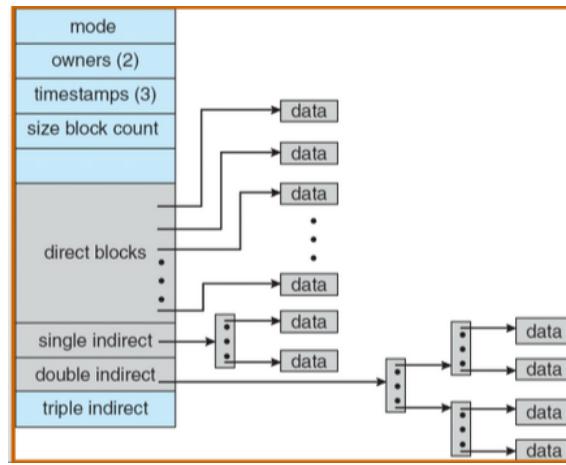
- Única entrada con la dirección del bloque de índices (index node / i-node)



- Variante: asignación por secciones
 - A cada entrada del bloque índice se agrega el campo longitud
 - El índice apunta al primer bloque de un conjunto almacenado de manera contigua



- Variante: niveles de indirección
 - Existen bloques directos de datos
 - Otros bloques son considerados como bloques índices (apuntan a varios bloques de datos)
 - Puede haber varios niveles de indirección



Gestión de Espacio Libre

- El control sobre cuáles de los bloques de disco están disponibles.
- Alternativas
 - Tabla de bits
 - Bloques libres encadenados
 - Indexación

Tabla de Bits

- Tabla (vector) con 1 bit por cada bloque de disco
- Cada entrada:
 - 0: bloque libre 1= bloque en uso
- Ventaja:
 - Fácil encontrar un bloque o grupo de bloques libres
- Desventaja
 - Tamaño del vector en memoria
 $\text{tamaño disco bytes} / \text{tamaño bloque en sistema de archivos}$
 ej: disco 16gb con bloques de 512 bytes → 32 mb

Ejemplo:

Ejemplo

00111

00001

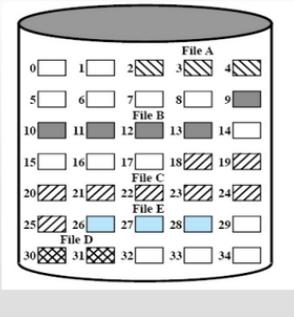
11110

00011

11111

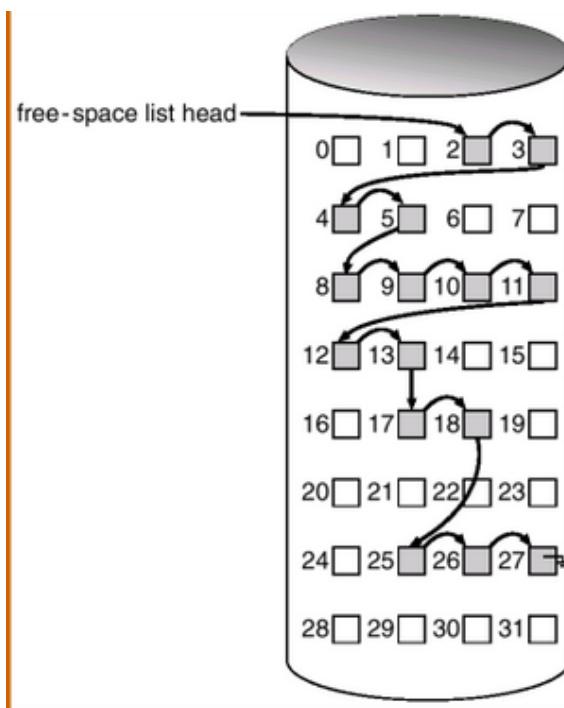
11110

11000



Bloques Encadenados

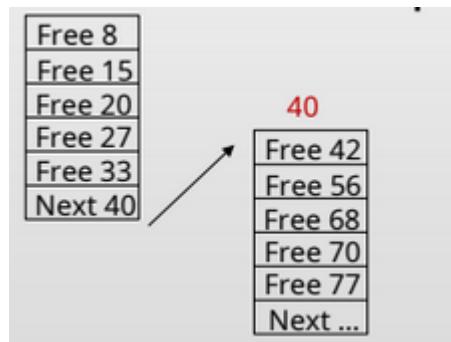
- Se tiene un puntero al primer bloque libre.
- Cada bloque libre tiene un puntero al siguiente bloque libre
- Ineficiente para la búsqueda de bloques libres → Hay que realizar varias operaciones de E/S para obtener un grupo libre
- Problemas con la pérdida de un enlace
- Difícil encontrar bloques libres consecutivos



Indexación (o agrupamiento)

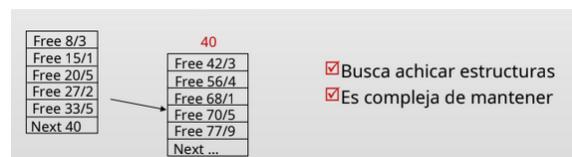
- Variante de "bloques libres encadenados"
- El primer bloque libre contiene las direcciones de N bloques libres

- Las N-1 primeras direcciones son bloques libres.
- La N-ésima dirección referencia otro bloque con N direcciones de bloques libres



Recuento (variante de indexación)

- Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación contigua)
- En lugar de tener N direcciones libres (índice) se tiene:
 - La dirección del primer bloque libre
 - Los N bloques libres contiguos que le siguen (#bloque, N siguientes bloques libres)



UNIX

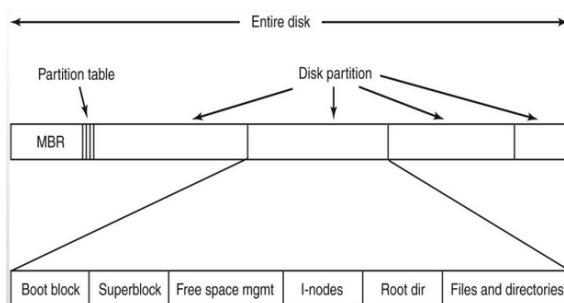
Manejo de archivos

- Tipos de archivos
 - Común
 - Directorio

- Archivos especiales (dispositivos /dev/sda)
- Names pipes (comunicación entre procesos)
- Links (comparten el i-nodo, solo dentro del mismo filesystem)
- Links simbólicos (tienen i-nodo propio, para filesystems diferentes)

Estructura del Volumen

- Cada disco físico puede ser dividido en uno o más volúmenes. Cada volumen o partición contiene un sistema de archivos. Cada sistema de archivos contiene:
 - Boot Block: Código para bootear el S.O.
 - Superblock: Atributos sobre el File System
 - Bloques/Clusters libres
 - I-NODE table: Tabla que contiene todos los I-NODOS
 - I-NODO: Estructura de control que contiene la información clave de un archivo
 - Data Blocks: Bloques de datos de los archivos

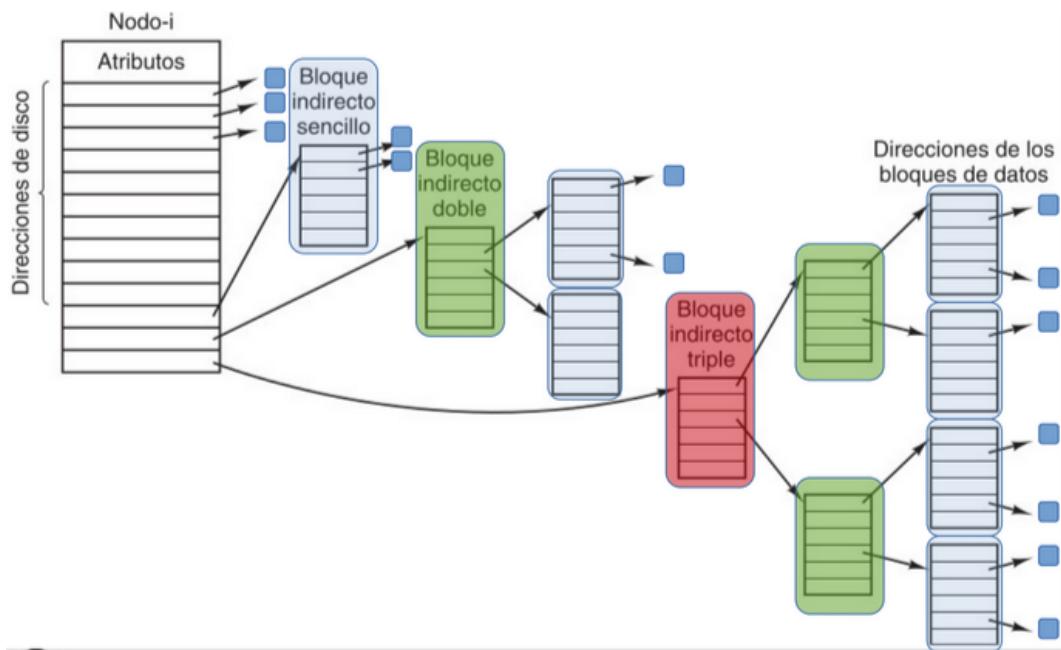


Información del i-nodo

- Un inodo es una estructura de datos del FileSystem que posee información sobre cada archivo, directorio u objeto que se almacene en el sistema de archivos
- Notar que el nombre del archivo no se almacena en esta estructura

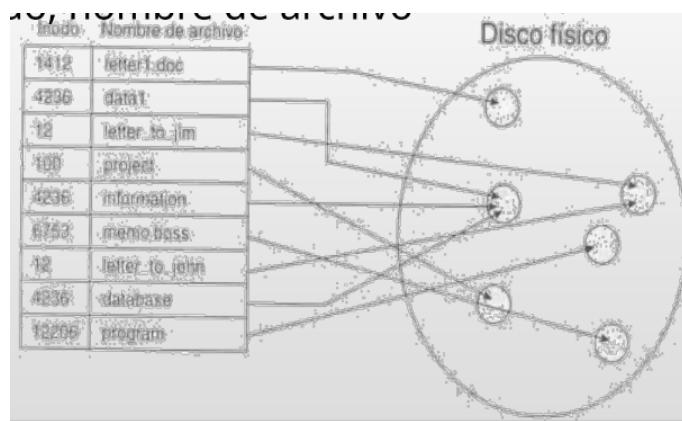
File Mode	16-bit flag that stores access and execution permissions associated with the file.
12-14	File type (regular, directory, character or block special, FIFO pipe)
9-11	Execution flags
8	Owner read permission
7	Owner write permission
6	Owner execute permission
5	Group read permission
4	Group write permission
3	Group execute permission
2	Other read permission
1	Other write permission
0	Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

I-NODO



Directarios

- Los nombres de archivos se almacenan en los directorios
- El directorio es una tabla de tuplas del tipo “Número de i-nodo, nombre del archivo”



Windows

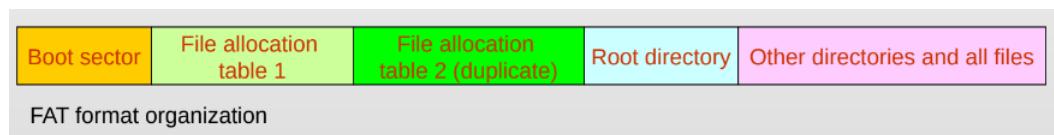
File Systems Soportados

- CD-ROM File System (CDFS) CD
- Universal Disk Format (UDF) DVD, Blu-Ray
- File Allocation Table
 - FAT12 MS-DOS v3.3 a 4.0 (año 1980), floppy
 - FAT16 MS-DOS 6.22, nombres cortos de archivo
 - FAT32 MS-DOS 7.10, nombres largos pero no soportados en MS-DOS
- New Technology File System (NTFS)

FAT

- FAT (File Allocation Table) es un sistema de archivos utilizado originalmente por DOS y Windows 9x
- ¿Por qué Windows aun soporta FAT file systems?:
 - Por compatibilidad con otro SO en sistemas multiboot
 - Para permitir upgrades desde versiones anteriores
 - Para formato de dispositivos como diskettes

- Las distintas versiones de FAT se diferencian por un número que indica la cantidad de bits que se usan para identificar diferentes bloques o clusters:
 - FAT12
 - FAT16
 - FAT32
- Se utiliza un mapa de bloques del sistema de archivos, llamado FAT
- La FAT tiene tantas entradas como bloques
- La FAT, su duplicado y el directorio raíz se almacenan en los primeros bloques de la partición



- Se utiliza un esquema de asignación encadenada
- La única diferencia es que el puntero al próximo bloque está en la FAT y no en los bloques
- Bloques libres y dañados tienen códigos especiales

Los directorios mantienen esta información

• Se utilizan estructuras de datos
es y dañados tienen
iales
• Los directorios mantienen esta información

DIRECTORIO		1er bloque	Tamaño
Nombre			
FICH_A		7	4
FICH_B		4	1
FICH_C		2	3

FAT	
Tamaño del disco	0
1	1
6	2
14	3
EOF	4
EOF	5
5	6
3	7
EOF	8
LIBRE	9
LIBRE	10
LIBRE	11
LIBRE	12
DAÑADO	13
8	14
LIBRE	15
...	...

FAT12

- En sistemas FAT12, al utilizarse 12 bits para la identificación del sector, la misma se limita a 2^{12} (4096) sectores
- Windows utiliza tamaños de sector desde los 512 bytes hasta 8KB (16 bloques consecutivos), lo que limita a un tamaño total de volumen de 32 MB, $2^{12} * 8$ KB
- Windows utiliza FAT12 como sistema de archivos de los disketts de 3,5 Y 12 pulgadas que pueden almacenar hasta 1,44 MB de datos

FAT16

- Al utilizar 16 bits para identificar cada sector puede tener hasta 2^{16} (65.536) sectores en un volumen
 - En windows el tamaño de sector en FAT16 varía desde los 512 bytes hasta los 64KB (128 sectores consecutivos), lo que limita a un tamaño máximo de volumen de 4 GB
 - El tamaño de sector dependía del tamaño del volumen al formatearlo

FAT32

- FAT32 fue el Filesystem mas reciente de la línea (posteriormente salió exFAT que algunos lo conocen como FAT64)
- FAT32 utiliza 32 bits para la identificación de sectores, pero reserva los 4 bits superiores, con lo cual efectivamente solo se utilizan 28 bits para la identificación:
 - El tamaño de sector en FAT 32 puede ser de hasta 32 KB, con lo cual tiene una capacidad teórica de direccionar volúmenes de hasta 8 TB
 - El modo de identificación y acceso de los sectores lo hace mas eficiente que FAT16. Con tamaño de sector de 512 bytes, puede direccionar volúmenes de hasta 128 GB.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

NTFS

- NTFS es el filesystem nativo de Windows desde Windows NT
- NTFS usa 64-bit para referenciar sectores
 - Teoricamente permite tener volumenes de hasta 16 Exabytes (16 billones de GB)
- ¿Porqué usar NTFS en lugar de FAT? FAT es simple, mas rápido para ciertas operaciones, pero NTFS soporta:
 - Tamaños de archivo y de discos mayores
 - Mejora performance en discos grandes
 - Nombres de archivos de hasta 255 caracteres
 - Atributos de seguridad
 - Transaccional

Cache de Disco

- Buffers en memoria principal para almacenamiento temporario de bloques de disco
- Objetivo: MINIMIZAR LA FRECUENCAI DE ACCESO AL DISCO

Algunas observaciones

- Cuando un proceso quiere acceder a un bloque de la cache hay dos alternativas
 - Se copia el bloque al espacio de direcciones del usuario → no permitiría compartir el bloque
 - Se trabaja como memoria compartida → permite acceso a varios procesos
 - Esta área de memoria debe ser limitada, con lo cual debe existir un algoritmo de reemplazo

Estrategia de reemplazo

- Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo no es referenciado
- Es una lista de bloques, donde el último es el más recientemente usado (LRU, Least Recently Used)
- Cuando un bloque se referencia o entra en la cache queda al final de la lista
- No se mueven los bloques en la memoria: se asocian punteros
- Otra alternativa: Least Frequently Used. Se reemplaza el que tenga menor número de referencias

Unix System V

Objetivo y estructura

- Minimizar la frecuencia de acceso a disco
- Es una estructura formada por buffers en Mem. principal
- El kernel asigna un espacio en la memoria principal durante la inicialización para esta estructura
- Un Buffer-Cache tiene dos partes

- Headers: Contienen información para modelar: la ubicación del bloque en RAM, número del bloque, estado, relaciones entre headers, etc.
- El buffer en sí: El lugar en RAM (referenciado por el header) donde se almacena realmente el bloque del dispositivo llevado a memoria

Buffer Cache en el Kernel

- El módulo de buffer cache es independiente del sistema de archivos y de los dispositivos de hardware
- Es un servicio del SO

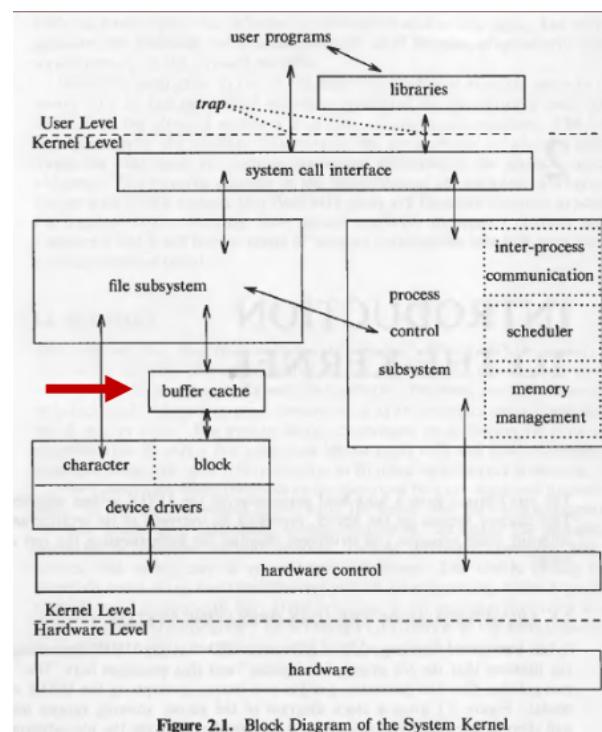


Figure 2.1. Block Diagram of the System Kernel

El Header

- Identifica el nro. de dispositivo y nro. de bloque
- Estado
- Punteros a:
 - 2 punteros para hash queue (más adelante vemos para que se usan)
 - 2 punteros para la free list (más adelante vemos para que se usan)
 - 1 puntero al bloque en memoria

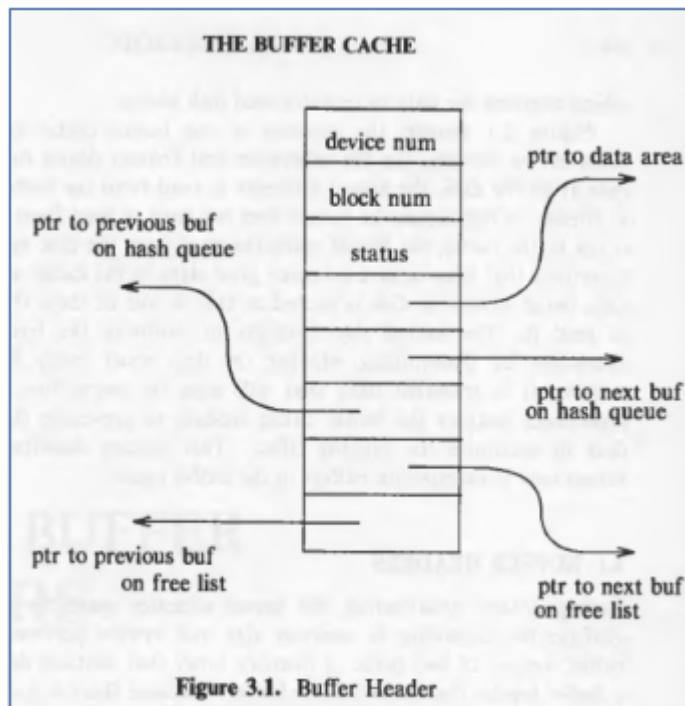


Figure 3.1. Buffer Header

Estado de los buffers

- Free o disponible
- Busy o no disponible (en uso por algún proceso)
- Se está escribiendo o leyendo del disco
- Delayed Write (DW): buffers modificados en memoria, pero los cambios no han sido reflejados en el bloque original en disco

Free List

- Organiza los buffers disponibles para ser utilizados para cargar nuevos bloques de disco
- No necesariamente los buffers están vacíos (el proceso puede haber terminado, liberado el bloque pero sigue en estado delayed write)
- Se ordena según LRU (Least recent used)

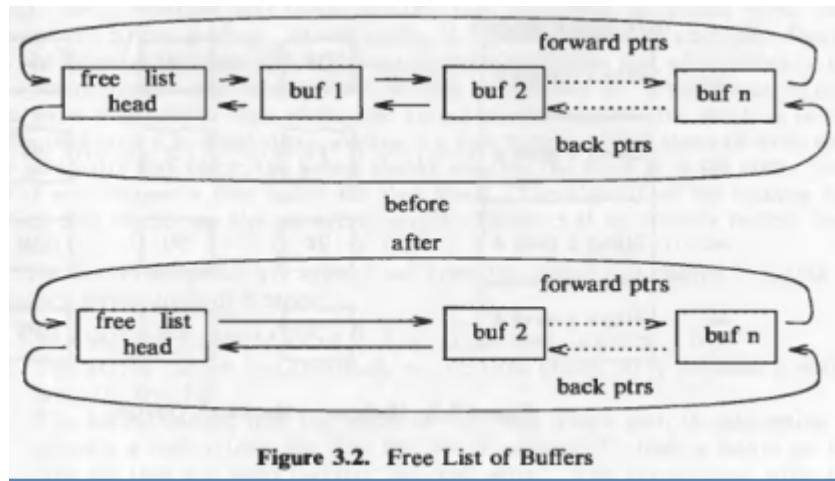


Figure 3.2. Free List of Buffers

Hash Queues

- Son colas para optimizar la búsqueda de un buffer particular
- Los headers de los buffers se organizan según una función de hash usando (dispositivo, #bloque)
- Al número de bloque (dispositivo/bloque) se le aplica una función de hash que permite agrupar los buffers cuyo resultado dio igual para hacer que las búsquedas sean más eficientes
- Se busca que la función de hash provea alta dispersión para lograr que las colas de bloques no sean tan extensas

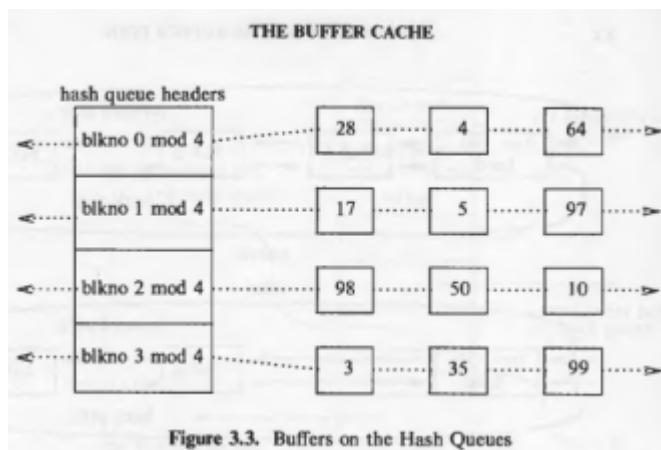


Figure 3.3. Buffers on the Hash Queues

- Para agrupar los bloques se utilizan los punteros que anteriormente habíamos visto que se almacenaban en el header

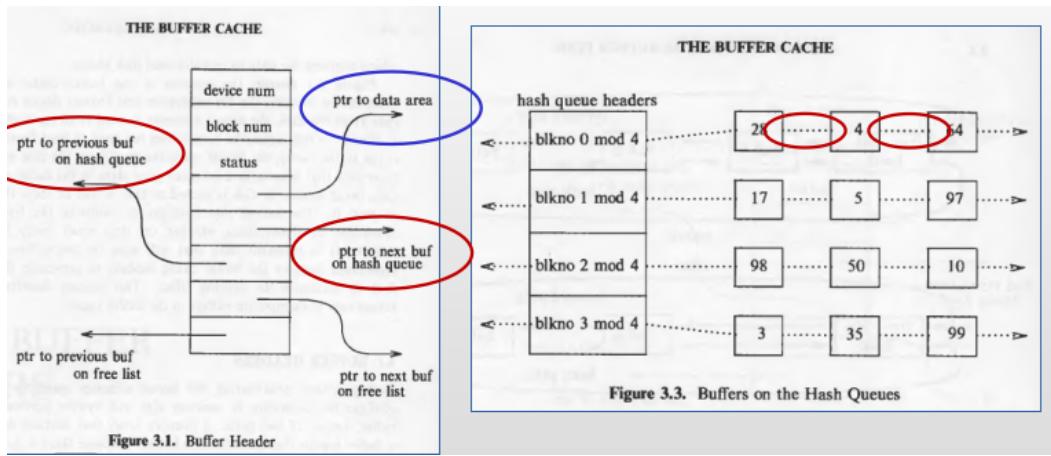


Figure 3.1. Buffer Header

Figure 3.3. Buffers on the Hash Queues

Free List

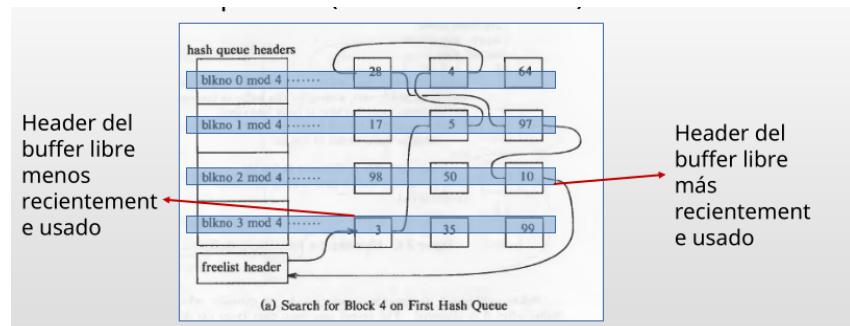
- Sigue el mismo esquema de la hash queue pero contiene los headers de los buffers de aquellos procesos que ya han terminado
- El header de un buffer siempre está en la Hash Queue
- Si el proceso que lo referenciaba terminó, va a estar en la Hash Queue y en la Free List

Funcionamiento del buffer Cache

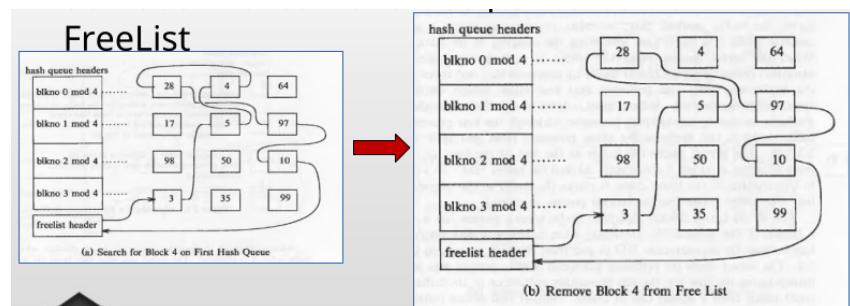
- Cuando un proceso quiere acceder a un archivo, se utiliza su i-nodo para localizar los bloques de datos donde se encuentra este
- El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar E/S
- Se pueden dar 5 escenarios:
 1. El kernel encuentra el bloque en la hash queue y el buffer está libre (en la free list)
 2. El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
 3. Idem 2, pero el bloque libre esta marcado como DW
 4. El kernel no encuentra el bloque en la hash queue y la free list está vacía
 5. El kernel encuentra el bloque en la hash queue pero está BUSY

Búsqueda/recuperación de un buffer (1er escenario):

- Ejemplo: busco el bloque 4:
 - El kernel encuentra el bloque en la hash queue
 - Está disponible (está en la free list)

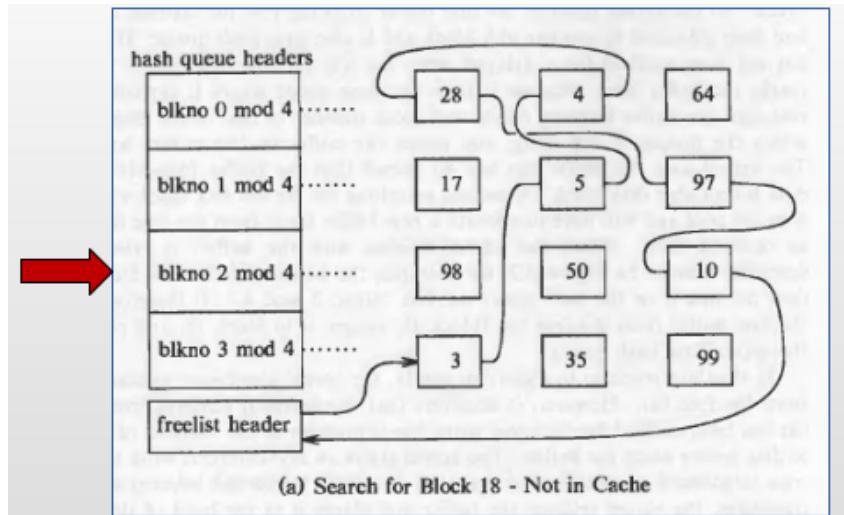


- Se remueve el buffer 4 de la free list
- Pasa el buffer 4 a estado BUSY (ocupado)
- El proceso usa el bloque 4
- Se deben reacomodar los punteros de la FreeList

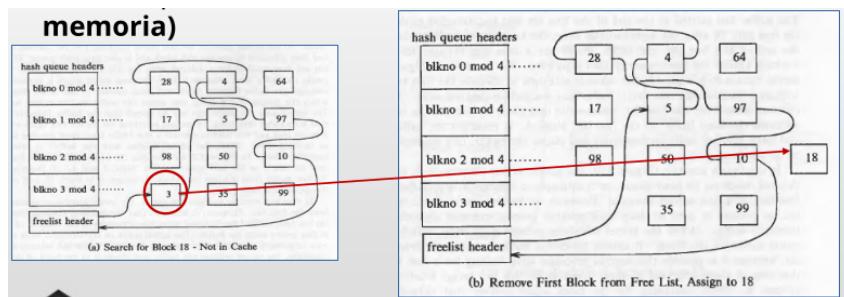


Búsqueda/recuperación de un buffer(2do escenario):

- Ejemplo: busco el bloque 18:
 - El bloque buscado no está en la hash queue
 - Se debe buscar un bloque libre

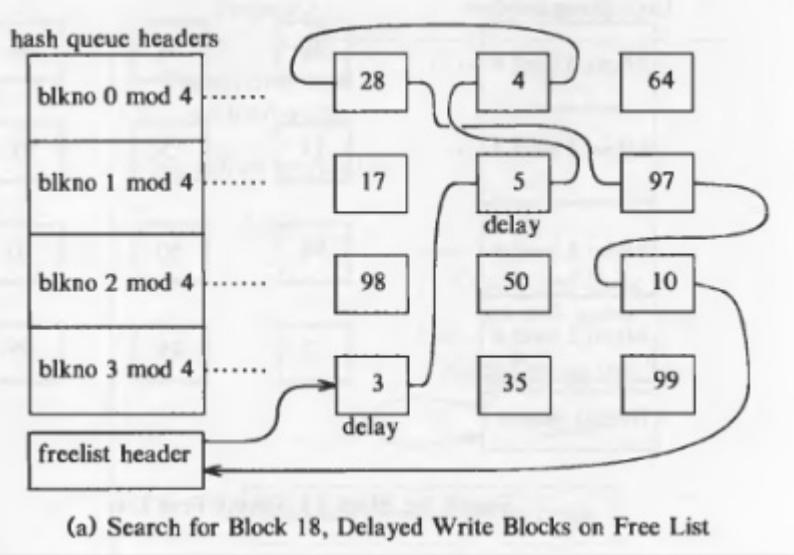


- Se toma un buffer de la free list (el 3)
- Siempre se usa el primero
- Se lee del disco el bloque deseado en el buffer obtenido
- Se ubica el header en la hash queue correspondiente (solo se cambian punteros, NO se intercambian ubicaciones de memoria)



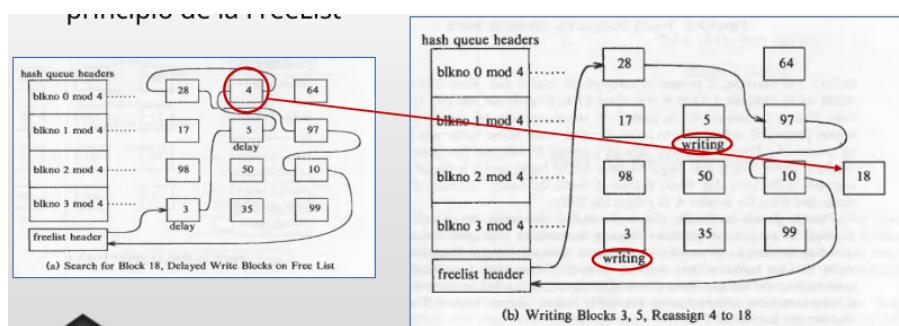
Búsqueda/recuperación de un buffer (3er escenario):

- Ejemplo: busca el bloque 18:
 - El kernel no encuentra el bloque buscado en la hash queue
 - Debe tomar el 1ero de la free list, pero está marcado DW
 - El kernel debe mandar a escribir a disco al bloque 3 y tomar el siguiente buffer de la free list



(a) Search for Block 18, Delayed Write Blocks on Free List

- Si también está DW, sigue con el mismo proceso hasta encontrar uno que no esté marcado como DW
- Mientras los DW se escriben en disco, se asigna el siguiente buffer free al proceso
- Una vez escritos a disco los bloques DW, estos son ubicados al principio de la FreeList



Búsqueda/recuperación de un buffer(4to escenario):

- El kernel no encuentra el bloque en la hash queue y la free list está vacía
- El proceso queda bloqueado en espera a que se "libere" algún buffer
- Cuando el proceso despierta se debe verificar nuevamente que el bloque no esté en la hash queue (algún proceso pudo haberlo pedido antes mientras éste dormía)

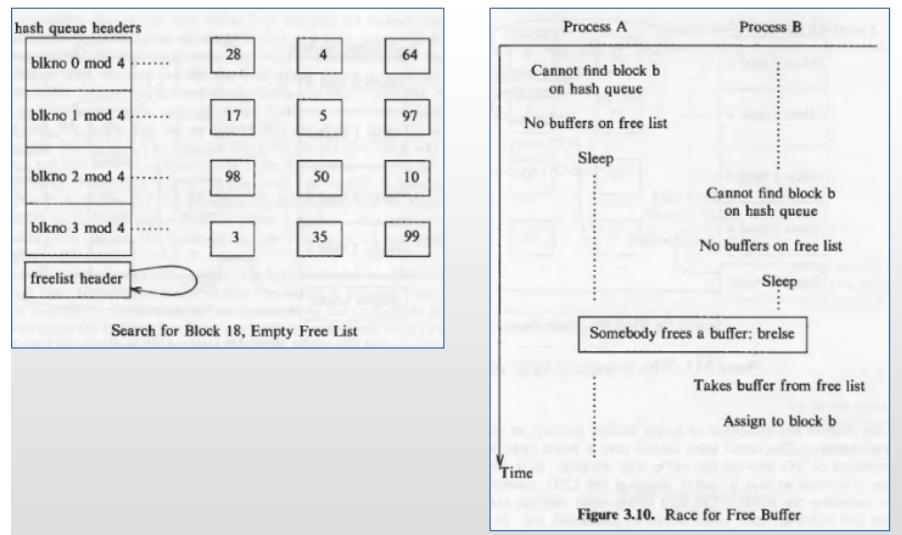
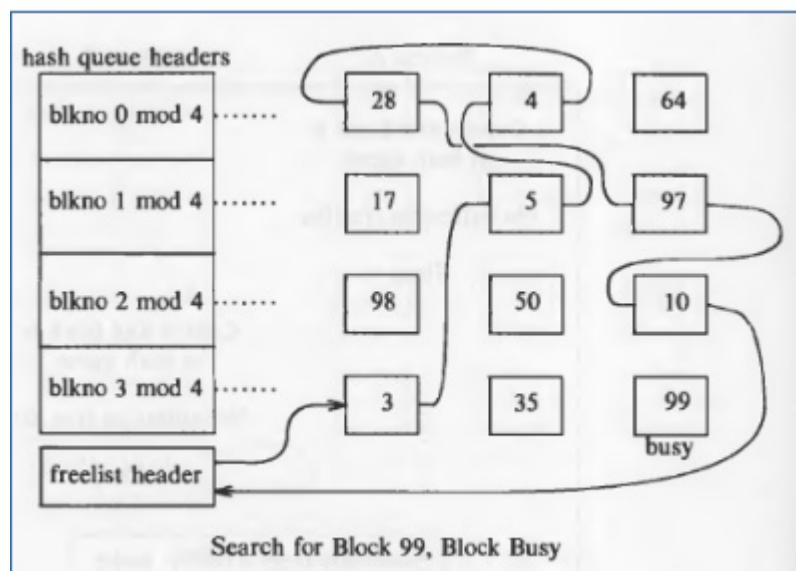


Figure 3.10. Race for Free Buffer

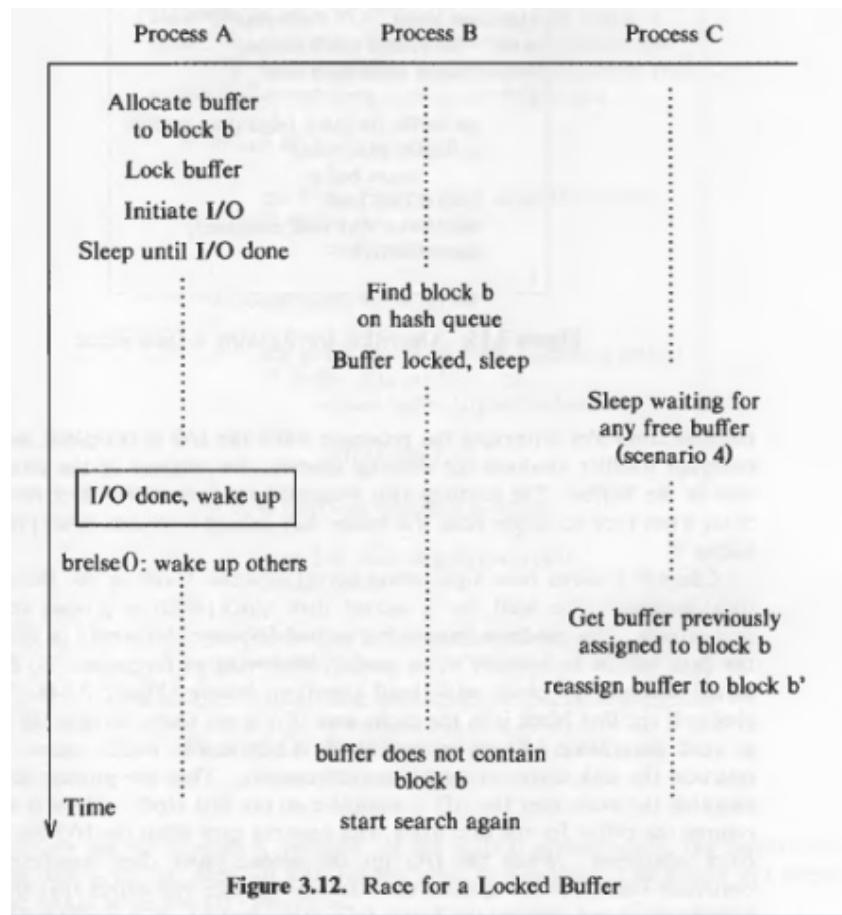
Búsqueda/recuperación de un buffer (5to escenario):

- Ejemplo: busca el bloque 99:
 - El kernel busca un bloque y el buffer que lo contiene está marcado como busy
 - El proceso se bloquea a la espera de que el buffer se desbloquee



- Eventualmente el proceso que tenía el buffer 99 lo libera
 - Se despiertan todos los proceso en espera de algún buffer

- El proceso que buscaba el buffer 99 debe buscarlo nuevamente en la hashqueue y en la freelist



Algoritmo de asignación

- Escenarios:
 1. El kernel encuentra el bloque en la hash queue y el buffer está libre.
 2. El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.
 3. Idem 2, pero el bloque libre esta marcado como DW.
 4. El kernel no encuentra el bloque en la hash queue y la free list está vacía.
 5. El kernel encuentra el bloque en la hash queue pero está BUSY.

```

algorithm getblk
input: file system number
       block number
output: locked buffer that can now be used for block
{
    while (buffer not found)
    {
        if (block in hash queue)
        {
            if (buffer busy)      /* scenario 5 */
            {
                sleep (event buffer becomes free);
                continue;        /* back to while loop */
            }
            mark buffer busy;    /* scenario 1 */
            remove buffer from free list;
            return buffer;
        }
        else      /* block not on hash queue */
        {
            if (there are no buffers on free list)      /* scenario 4 */
            {
                sleep (event any buffer becomes free);
                continue;        /* back to while loop */
            }
            remove buffer from free list;
            if (buffer marked for delayed write) {      /* scenario 3 */
                asynchronous write buffer to disk;
                continue;        /* back to while loop */
            }
            /* scenario 2 --- found a free buffer */
            remove buffer from old hash queue;
            put buffer onto new hash queue;
            return buffer;
        }
    }
}

```

Figure 3.4. Algorithm for Buffer Allocation