FINALES ISO

1)System calls:

a) DefiniciÛn. Explicar como pueden ser implementadas, para ello tenga en cuenta: participaciÛn o apoyo del harware, modos de ejecuciÛn, stacks (pilas).

b) Suponiendo que un proceso ejecuta la System Call "fork()", describa eventos y actividades que se suceden hasta que el nuevo proceso comienza su ejecuciÛn (tener en cuenta planificadores, dispatcher, etc)

1) Una SystemCall Es la forma en que los programas de usuario acceden a los servicios del SO, consiste de los siguientes pasos:

1. El proceso de Usuario llama a una Syscall (fork en este caso).
2. Se guarda en el stack del usuario los parámetros de la syscall y se lanza una interrupción.
3. Se cambia a modo Kernel.
4. Se guarda el PC (Program Counter: indica la dirección de la siguiente instrucción a ejecutar) y el PSW (Program Status Word: indica el estado del programa actual) en la pila del usuario.
5. Se carga en el PC la dirección de la subrutina que atiende la interrupción.
6. Se sacan los parámetros de la SysCall de la pila del usuario para ver que llamada tiene que atender.
7. Se guarda la dirección de la pila del usuario en el PCB del proceso y se pasa a utilizar la pila del kernel.
8. Se guardan los parámetros de la Syscall en la pila del kernel y se ejecuta la SysCall.
9. Si la SysCall bloquea el proceso se ejecuta el short term scheduler para que un nuevo proceso tome la cpu y esta no quede ociosa.
    a. Se da el context switch
    b. se guarda en la pila la dirección que el HW dejo previo a que el proceso seleccionado sea suspendido.
10. Más allá de que se haya bloqueado o no el proceso, quien ahora tengo el control de la CPU va a cambiar el modo a User Mode.
11. Ejecutar la sentencia RET para obtener de la pila la dirección de retorno a ejecutar.
12. Se obtienen de la pila el PSW y PC y se continúa la ejecución del proceso actual.

2) Memoria virtual con paginaciÛn por demanda:

a) Funcionamiento. Estructuras utilizadas, participaciÛn del hardware, actividades desarrolladas por el SO.

b) Fallos de p·ginas: Explicar como se resuelven y que estructuras de las descriptas en el punto anterior son utilizadas (tanto modificadas, creadas y/o consultadas)

2) La **paginación por demanda** es un sistema de paginación con el cual, además de las ventajas de la paginación convencional, se busca disminuir los tiempos de respuesta y aumentar la cantidad de programas en memoria. Para lograr estos objetivos se hace uso del paginador, el cual carga a memoria solo las páginas que serán utilizadas por el programa en ejecución, de esta manera se logra un menor tiempo de carga y un ahorro en cuanto a espacio utilizado por dicho programa, ya que, por un lado, no necesitamos que todo el programa este en memoria para comenzar su ejecución mientras que, por otra parte, al no estar el programa completo en memoria, disminuimos considerablemente el espacio que éste ocupa.

Ya que el paginador solo busca las páginas que se necesitan para ejecutar algún programa, debemos agregar un bit que nos diga si las referencias de memoria son válidas o no, de lo contrario, al no encontrar una página no podríamos diferenciar si el paginador aún no la carga o si esta es realmente una referencia inválida.

El proceso que se sigue es el siguiente:

1. Se intenta leer la página requerida
2. Si la página requerida ya esta en memoria, simplemente se lee.
3. Si no está en memoria, revisa si la referencia es válida (mediante el bit, el hardware).
4. Si la referencia es inválida, se aborta (se genera un trap al SO).
5. Si la referencia es válida, se intenta cargar la página.
6. Cuando la página sea cargada, se reintenta la instrucción.

Al buscar una página, si ésta no está en memoria, necesitará ser cargada. A este proceso se le llama **fallo de página**.

Al iniciar la ejecución de un programa, la tabla de páginas cuenta con todas sus entradas inválidas por lo cual el paginador fallará hasta tener lo necesario para iniciar el programa. Luego de esta carga inicial se comprobará si la siguiente página a utilizar ya está en memoria, en caso de que la página se encuentre, ésta es leída, pero cuando la página no es encontrada (y es una referencia válida) tenemos dos posibilidades:

- Si existe un frame libre, se carga y se lee.
- Si no tenemos frames libres, se intercambia la página de algún frame por la información a utilizar.

b) Como hemos visto anteriormente, cuando un proceso requiere una página que no está en memoria se genera un fallo de página. La gran mayoría de las dificultades de la paginación por demanda se deben a cómo los fallos de página son tratados.

En primer lugar, para que los fallos de página puedan ser tratados correctamente necesitamos un sistema que sea capaz de reiniciar una instrucción, de esta manera pasará lo siguiente:

- Una instrucción necesita una página que no está en memoria.
- Se genera fallo de página (No se puede satisfacer la instrucción).
- Se carga a memoria la página requerida.
- Se reinicia la instrucción

Tratamiento de fallos de páginas

Para lograr un rendimiento aceptable del sistema de paginación por demanda debemos mantener la tasa de fallos de página al mínimo. Existen diferentes enfoques para su disminución:

Algoritmos para el reemplazo de páginas

Algoritmo óptimo: Consiste en quitar de memoria la página que no será utilizada en más tiempo.
FIFO (First in first out): Como su nombre lo indica, la primera página que fue cargada a memoria es la primera en salir de esta.
LRU (Least recently used): Plantea quitar de memoria las páginas menos usadas recientemente, para ello, ordena las páginas poniendo arriba las que fueron usadas recientemente y va reemplazando por las páginas que se sitúan abajo.
Segunda oportunidad: Mejora del algoritmo FIFO. Se agrega un bit de referencia a todas las páginas y cuando éstas son utilizadas se fija en 1. El algoritmo consiste en buscar las páginas de la misma manera que lo hacemos en FIFO.
Entre otros...

SEGMENTACION POR DEMANDA::::::::(Tambien lo han tomado en el 2014)
3) Memoria virtual con segmentación por demanda:
http://es.calameo.com/read/003308231d6d4fbd671ee
SEGMENTACION POR DEMANDA::::::::

3) Sistemas de archivos.

a) Describa la estructura del sistema de archivos de UNIX System V.

b) Si quisiera crear un nuevo archivo, por ejemplo /home/usuario/final.txt, que estructuras de las descriptas en el

3) System V UNIX
UNIX - Estructura del Volumen

- Boot Block: Código para bootear el S.O.
- Superblock: Atributos sobre el File System
- 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

Primero tenes que describir la estructura de bloques del System v, osea boot block, superblock , tabla de inodos y bloques de datos. Despues te dice que detalles la creacion de un archivo, empezando por crear el i-nodo, almacenarlo en la tabla, modificacion de estructuras en su creacion como el superbloque, tabla de archivos y tabla de descriptores de archivos. Después tenés que saber la forma de acceder a ese path o de almacenar un archivo en un determinado directorio entrando con los i-nodos de los archivos.

ESTRUCTURA DEL VOLUMEN

Un sistema de ficheros UNIX reside en un único disco lógico o partición de disco y se compone de los siguientes elementos:

• Bloque de arranque. Contiene el código requerido para arrancar el sistema operativo.

• Superbloque. Contiene atributos e información sobre el sistema de ficheros, tal como el tamaño de la partición y el tamaño de la tabla de nodos-i.

• Tabla de nodos-i. La colección de nodos-i para cada fichero.

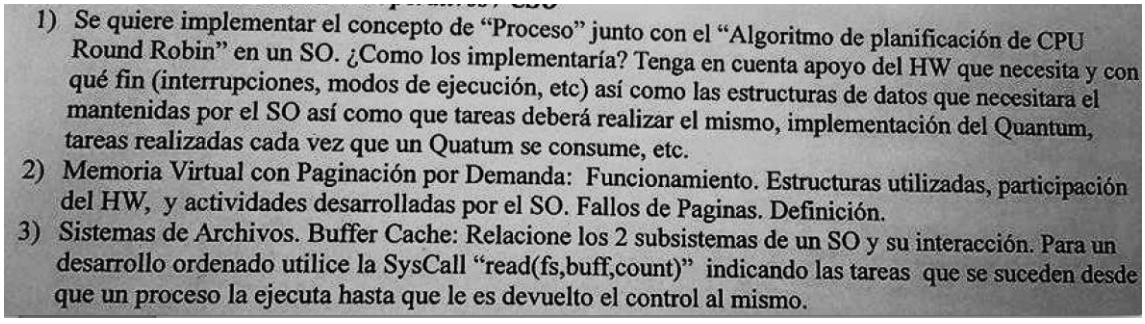
• Bloques de datos. El espacio de almacenamiento disponible para los ficheros de datos y subdirectorios.

Un sistema de archivos **UFS** se compone de las siguientes partes:

- unos pocos bloques al inicio de la partición reservados para bootstrap (el cual debe ser inicializado separadamente del sistema de archivos).
- un superbloque que contiene un número mágico (del inglés magic number) identificando esto como un UFS, y algunos otros números vitales describiendo la geometría y parámetros de puesta a punto del comportamiento.
- una colección de grupos de cilindros. Cada grupo de cilindros tiene estos componentes:
  - un respaldo del superbloque.
  - una cabecera de cilindro, con estadísticas, lista de espacio libre, etc. acerca de este bloque de cilindros, similar a los que se encuentran en el superbloque.
  - un número de inodos, cada cual conteniendo los atributos del archivo.
  - un número de bloques de datos.

Los inodos son numerados secuencialmente. Los primeros inodos están reservados por razones históricas, seguidos por los inodos del directorio raíz.

Los archivos de directorio contienen sólo la lista de archivos en el directorio y el inodo asociado para cada archivo. Todos los metadatos (metadata) son mantenidos en el inodo.


1) Se quiere implementar el concepto de "Proceso" junto con el "Algoritmo de planificación de CPU Round Robin" en un SO. ¿Como los implementaría? Tenga en cuenta apoyo del HW que necesita y con qué fin (interrupciones, modos de ejecución, etc) así como las estructuras de datos que necesitara el mantenidas por el SO así como que tareas deberá realizar el mismo, implementación del Quantum, tareas realizadas cada vez que un Quatum se consume, etc.
2) Memoria Virtual con Paginación por Demanda: Funcionamiento. Estructuras utilizadas, participación del HW, y actividades desarrolladas por el SO. Fallos de Paginas. Definición.
3) Sistemas de Archivos. Buffer Cache: Relacione los 2 subsistemas de un SO y su interacción. Para un desarrollo ordenado utilice la SysCall "read(fs,buff,count)" indicando las tareas que se suceden desde que un proceso la ejecuta hasta que le es devuelto el control al mismo.

1) Round-robin es un algoritmo de planificación de procesos simple de implementar. Dentro de un sistema operativo se asigna a cada proceso una porción de tiempo equitativa y ordenada, tratando a todos los procesos con la misma prioridad. En Sistemas operativos, la planificación Round-robin da un tiempo máximo de uso de CPU a cada proceso, pasado el cual es desalojado y retornado al estado de listo, la lista de procesos se planifica

por FIFO, del inglés "First In, First Out" (primero en entrar, primero en salir o primero llegado, primero atendido).
Este algoritmo de planificación, conocido por **Round robin**, está diseñado especialmente para sistemas de tiempo compartido. Se define un intervalo de tiempo denominado "Quantum", cuya duración varía según el sistema. La cola de procesos se estructura como una cola circular. El planificador la recorre asignando un cuanto de tiempo a cada proceso. La organización de la cola es FIFO. El Quantum se suele implantar mediante un temporizador que genera una interrupción cuando se agota el Quantum de tiempo. Si el proceso agota su ráfaga de CPU antes de finalizar el Quantum, el planificador asigna la CPU inmediatamente a otro proceso. Este algoritmo tiene un tiempo de espera relativamente grande.

::::::::PUEDEN TOMAR JUNTO A ALGUN ALGORITMO DE PLANIFICACION (NO SOLO ROUND ROBIN)

2) Hecho arriba.

3) http://www.ual.es/~acorral/DSO/Tema_4.pdf (IMPORTANTISIMO)::::::::: ver pasos de system call y cuando ejecuta la syscall (punto 8) agregar pasos de read (en este caso).

Final ISO mesa septiembre 2015

1. Procesos y syscall

a. Describa las estructuras de datos y actividades llevadas a cabo para implementar el manejo de procesos. Tener en cuenta: planificaciÛn, estados, mÛdulos, etc.
b. Syscall. DefiniciÛn. CÛmo trabaja fork() en Linux, indique el apoyo por parte de HW.
c. indicar que estructuras de a) son utilizadas y/o modificadas a causa de b)

2. Memoria virtual. PaginaciÛn con demanda
a. CÛmo y quiÈn realiza la traducciÛn lÛgica en fÌsica. Tener en cuenta HW, actividades del SO, estructuras de datos, etc.
b. HiperpaginaciÛn. DefiniciÛn. CÛmo puede ser tratada? RelaciÛn con working set.

3. Sistema de E/S. Describa el sistema de archivos en Unix System V. describa los pasos necesarios para abrir un archivo open('/home/iso/finales.doc').

1) a)Proceso: programa en ejecución, tiene 3 partes, código, datos y stack (datos temporales, parámetros, direcciones de retorno.)
Los procesos tienen dos pilas, una para el modo usuario y otra para el modo kernel
Atributos de un proceso: ID del proceso, ID del padre, ID de quien lo “disparó”, ID de grupo que lo “disparó”
PCB: hay uno por proceso, contiene estado, PC y registros de la CPU del proceso, es lo primero que se crea y lo último que se borra al iniciarse un proceso.
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.
Context Switch: se produce cuando la CPU cambia de un proceso a otro, se debe guardar la info del proceso saliente y cargar la del entrante, este cambio deja oscilando a la CPU.

Estados de un proceso:

- Nuevo (new)
    - Un usuario "dispara" el proceso, que es creado por su proceso padre, en este estado se crean las estructuras asociadas al proceso y este queda en la cola de procesos en espera de ser cargado en memoria.
- Listo para ejecutar (ready)
    - El proceso se encuentra en la cola de procesos listos, El scheduler de largo plazo elige el proceso para cargarlo en memoria, el proceso solo necesita que se le asigne CPU.
- Ejecutándose (running)
    - El short term sheduler lo eligió para asignarle CPU, y la tendrá hasta que se termine su tiempo asignado (quantum) o necesite que se produzca un evento como una I/O
- En espera (waiting)
    - El proceso necesita que se cumpla un evento para continuar, sigue en memoria pero no tiene la CPU, cuando se cumpla el evento pasará al estado listo.
- Terminado (terminated)

- 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.
- running-ready: Se termina el quantum del proceso y pasa al estado ready para competir por la CPU

Existen módulos que controlan la planificación, como por ejemplo el Scheduler de long term, short term y médium term. También existen el dispatcher (que hace el cambio de contexto, despacha el proceso elegido por el short term) y el loader (Carga en memoria el proceso elegido por el long term), que pueden no existir como módulos independientes pero su función debe ser cumplida.
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 el de short term.
Medium Term Scheduler (swapping): Si es necesario, reduce el grado de multiprogramación, Saca temporariamente 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.

b) Hecho arriba y junto con c).

2) Paginacion por demanda hecho arriba
a) El espacio de direcciones de un proceso varía dependiendo de la arquitectura del PC (32/64bits), este espacio es independiente de la ubicación real del proceso.
Direcciones lógicas: referencia a una localidad en memoria independiente de la asignación actual de los datos en la memoria, estas se traducen a direcciones físicas.
Direcciones físicas: Es la dirección absoluta en la memoria principal.
Para convertir una dirección lógica en física se utilizan dos registros auxiliares, Registro base que es la dirección de comienzo del proceso, y registro límite que es la dirección final del proceso. Su valor se fija cuando el proceso es cargado a memoria, con la dirección lógica y el registro base se genera una dirección física, esta se compara con el registro limite y si es invalida se genera una interrupción al SO.
Las direcciones en los programas fuentes son simbólicas, el compilador las convierte en direcciones reubicables, y el linkeditor en direcciones absolutas, este proceso se conoce como binding de direcciones.
Al hacer Swap out y Swap in de un proceso, si trabajamos con direcciones físicas se debe asignar al mismo espacio de memoria que ocupaba antes, en cambio con las lógicas se puede cargar en cualquier lado.
El mapeo de direcciones lógicas y físicas es realizado por el dispositivo de hardware MMU (Memory Managment Unit), se encuentra en el cpu y es reprogramado por el kernel.
El valor en el "registro de realocación" es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.

b) Thrashing: es cuando un Sistema pasa más tiempo paginando que ejecutando procesos reduciendo así la performance.

Working set: es la porción del espacio de direcciones del proceso que se encuentra en memoria. El HW se encarga de detectar una porción del proceso que no está en el working set.

Para evitar el thrashing se debe seleccionar un tamaño de working set optimo , si es muy chico no cubrirá la localidad, si es grande puede tomar varias localidades.
Para prevenir el thrashing el SO monitorea cada proceso asignándole tantos frames como la medida del working set del proceso requiera, si sobran frames entonces ejecuta otro proceso. Si la tasa de frames que necesita un proceso aumenta por encima de lo tolerable debe suspenderse otro proceso para cubrir las necesidades del proceso actual, de esta manera se aumenta el grado de multiprogramación optimizando el uso de la cpu.

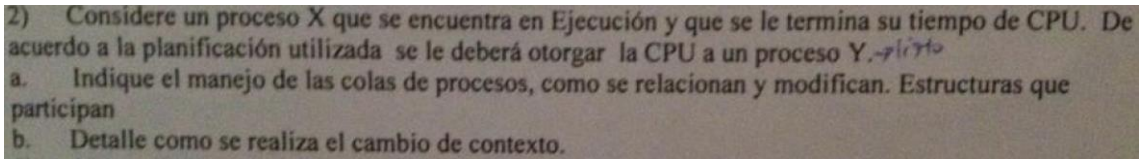
3) Sistemas de entrada salida: Existen varios dispositivos de I/O,

Legible para el usuario, como las impresoras, pantallas, teclados y mouse.
Legible para la máquina, como los discos, las cintas magnéticas, etc.
Y de Comunicación, como las líneas digitales, los módems, etc.
Como estos son muy variados y trabajan a velocidades diferentes con cantidades y formatos de datos distintos (y en gran mayoría más lentamente que la cpu y la RAM) se busca una manera de abstraerse de su funcionamiento específico. Para eso muchos dispositivos de HW y módulos de SW interactúan para resolver las operaciones de I/O. (Buses, Controladores, drivers, puertos, interrupciones, etc.) se sigue un estándar de resolución de I/o que permite que distintos dispositivos puedan comunicarse correctamente con todas las computadoras, el sistema operativo se comunica con el controlador y el controlador con el dispositivo.

E/s programada es sincrónica realiza polling, deja la cpu ociosa, ineficiente, la que es por interrupciones no es sincrónica, el cpu atiende otros procesos hasta que el controlador del dispositivo avisa con una interrupción que ya está lista la operación de I/O, en ese momento el proceso que se está atendiendo actualmente se bloquea y se restaura la ejecución del proceso que hizo la llamada de I/O.

La DMA es un mecanismo de I/O que lee una GRAN cantidad de información, el SO genera una interrupción de lectura al DMA, el DMA se lo manda al controlador, el controlador envía los bytes a la memoria directamente sin utilizar la CPU, cuando termina la transferencia se da una interrupción para avisar que terminó la operación.

Ver arriba...IMPORTANTISIMO::::::::::


2) Considere un proceso X que se encuentra en Ejecución y que se le termina su tiempo de CPU. De acuerdo a la planificación utilizada se le deberá otorgar la CPU a un proceso Y.
a. Indique el manejo de las colas de procesos, como se relacionan y modifican. Estructuras que participan
b. Detalle como se realiza el cambio de contexto.

2) a) Los procesos a medida que llegan se van encolando en la cola de procesos listos (luego de pasar del estado new a ready). Una vez cargados en memoria (por parte del modulo de long term scheduler y el Loader) es el modulo de short term scheduler el que se encarga de seleccionar cual de los procesos en la cola de listos se elige para que use la CPU, es decir, para que pase al estado de running y se apropie de la CPU. Una vez que el proceso X esta en ejecución y se le termina el tiempo de CPU, puede pasar que:
. Ocurra un evento o una acción de I/O, el proceso pasa a la cola de procesos en espera de dispositivos de I/O (y luego que se atendió vuelve al estado de ready para competir nuevamente por la CPU).
. Se le acabe su quantum (produciendo una interrupcion) y asi pase de running a ready.
.Termine su ejecucion pasando al estado de terminated.

b) Context Switch: Un **cambio de contexto** consiste en la ejecución de una rutina perteneciente al núcleo del sistema operativo multitarea de una computadora, cuyo propósito es parar la ejecución de un hilo (oproceso o Threads) para dar paso a la ejecución de otro distinto.

Context Switch: se produce cuando la CPU cambia de un proceso a otro, se debe guardar la info del proceso saliente y cargar la del entrante, este cambio deja oscilando a la CPU

Para hacer realidad la ejecución concurrente en primer lugar es necesario que el programa en ejecución se detenga voluntariamente. Puesto que esto no va a ocurrir nunca, es imprescindible la intervención del hardware. Gracias a las interrupciones generadas por el propio ordenador, es posible expulsar el programa en ejecución para dar paso al sistema operativo.

Cuando esto ocurre, el sistema operativo ejecuta inmediatamente la rutina de **cambio de contexto**. Esta rutina realiza las siguientes operaciones en el orden indicado:

1. **Salvar el estado del programa que se estaba ejecutando**. El estado, también denominado *contexto*, consiste en los valores de todos los registros del microprocesador. Se copian en la memoria principal.
2. **Seleccionar otro programa para ejecutar**. Entre todos los programas que estén preparados para ejecutarse, la rutina selecciona uno de ellos siguiendo algún algoritmo equitativo.
3. **Restaurar el estado del programa seleccionado**. Para ello, se toma el estado previamente copiado en la memoria principal y se vuelca en los registros del microprocesador.
4. **Ejecutar el programa seleccionado**. La rutina termina su ejecución saltando a la instrucción que estaba pendiente de ejecutar en el programa seleccionado.

Este ciclo se repite bien cada vez que ocurre un evento de entrada/salida, bien cuando vence un temporizador programado en el hardware.

3) Para poder utilizar Memoria Virtual el HW debe soportar paginación por demanda y/o segmentación., se necesita un dispositivo de memoria secundaria (disco) y el SO debe soportar el movimiento de páginas/segmentos entre MP y MS.

Memoria virtual con paginación:

Cada proceso tiene su tabla de páginas, donde cada entrada hace referencia al marco donde está la página en memoria principal. También tienen bits de control, el Bit V indica si la página esta en memoria, el Bit M indica si la página fue modificada (para saber que deben reflejarse los cambios en memoria secundaria.)
Page fault: ocurre cuando el proceso intenta usar una dirección que está en una página que no está en el wortking set (Bit V=0).
El HW lo detecta y genera un trap al SO.
El SO coloca el proceso en espera mientras gestiona la carga de la pagina necesaria, durante esta operación de E/S, otros procesos toma el control de la CPU, la carga de la página consiste en buscar un marco vacío y copiar en el marco la página necesaria, una vez realizado se avisa con una interrupción, entonces el SO actualiza la tabla de páginas en el proceso poniendo el bit V de la página en 1 y la dirección base del marco donde colocó la página, se pone en ready el proceso que generó el fallo de página.
Una tabla de páginas puede ser de un nivel, de múltiples niveles o tabla invertida (hashing), esta elección depende del HW. La tabla invertida se usa en arquitecturas donde el espacio de direcciones es muy grande, se tiene una entrada por cada marco y hay una sola tabla para todo el sistema, el número de página se transforma en un valor de HASH, y el HASH se usa como índice para encontrar el marco asociado
Tamaño de la Pagina ⌡ Pequeño ⌉ Menor Fragmentación Interna. ⌉ Más paginas requeridas por proceso ⌊ Tablas de páginas mas grandes. ⌉ Más paginas pueden residir en memoria ⌡ Grande ⌉ Mayor Fragmentación interna ⌉ La memoria secundaria esta diseñada para transferir grandes bloques de datos más eficientemente ⌊ Mas rápido mover páginas hacia la memoria principal.

Cuando se da un pagefault y no se tienen marcos vacíos, se busca una página víctima, lo ideal es que la victima no sea referenciada en un futuro cercano, esto se logra prediciendo el futuro a través de la lectura de las acciones pasadas.
Reemplazo Global ⎪El fallo de página de un proceso puede reemplazar la página de cualquier proceso. ⎪Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él. ⎪El SO no controla la tasa de page-faults de cada proceso. ⎪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 page-faults de cada proceso ⎪Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

4) Sistema de archivos: Brindar espacio en disco a los archivos de usuario y del sistema.
Mantener un registro de espacio libre.

Pre-asignacion:

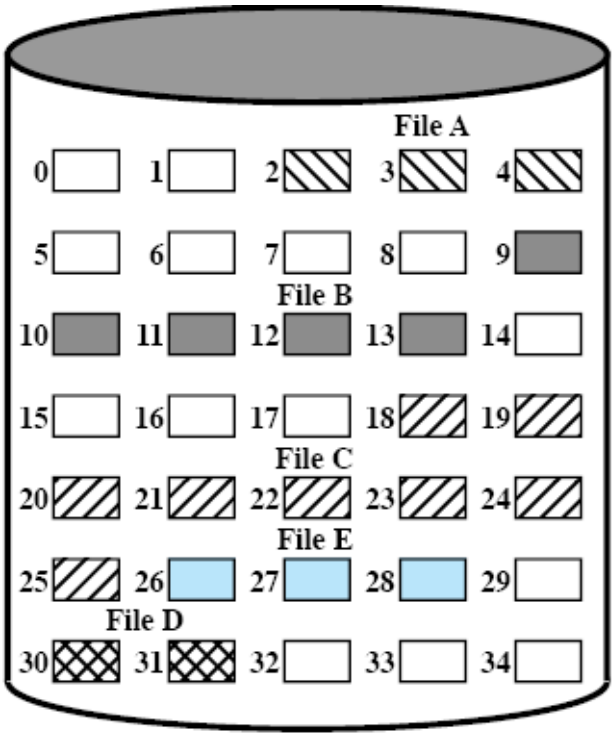
- Se necesita saber cuanto espacio va a ocupar el archivo en el momento de su creación

- Se tiende a definir espacios mucho más grandes que lo necesario

- Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo

Asignacion dinámica:
El espacio se solicita a medida que se necesita
Los bloques de datos pueden quedar de manera no contigua.

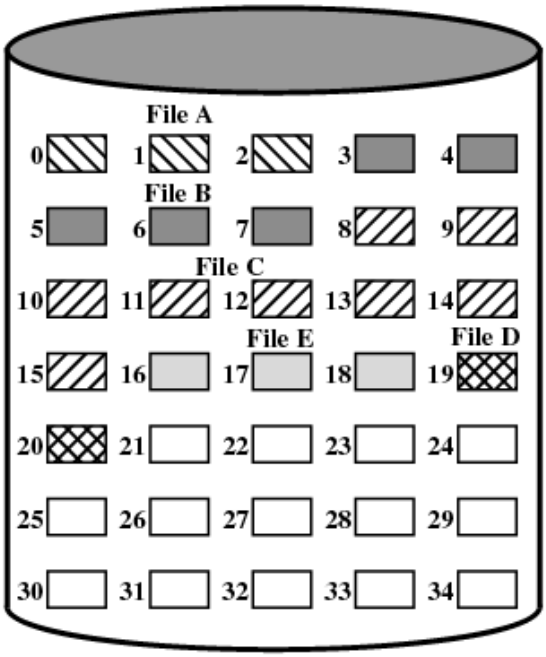
Continua:



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

- 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



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.8 Contiguous File Allocation (After Compaction)

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

Encadenada:

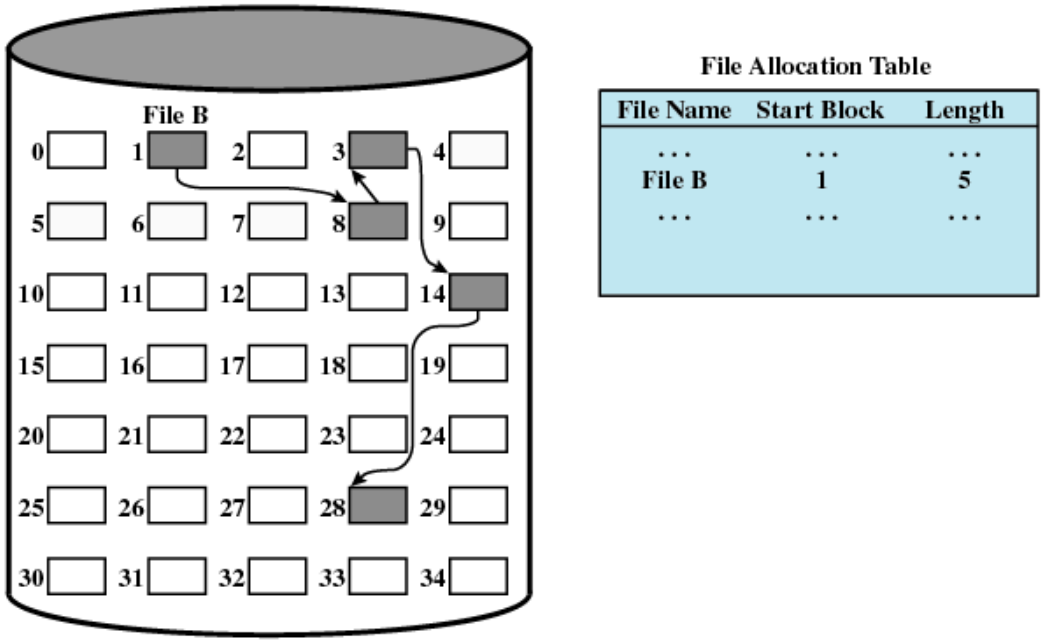
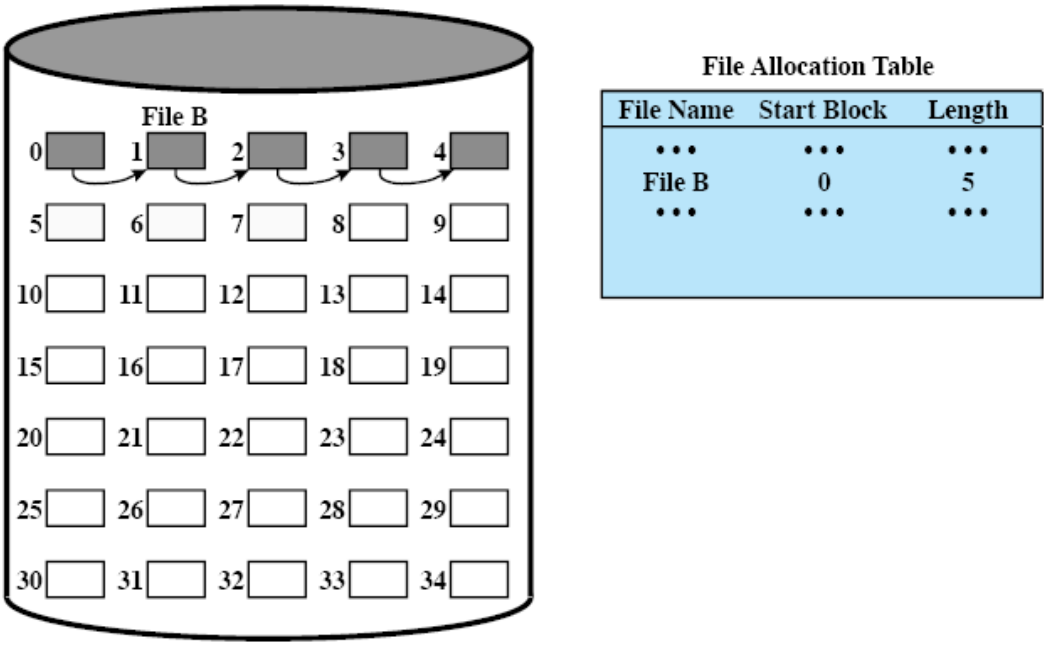


Figure 12.9 Chained Allocation

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



Indexada:

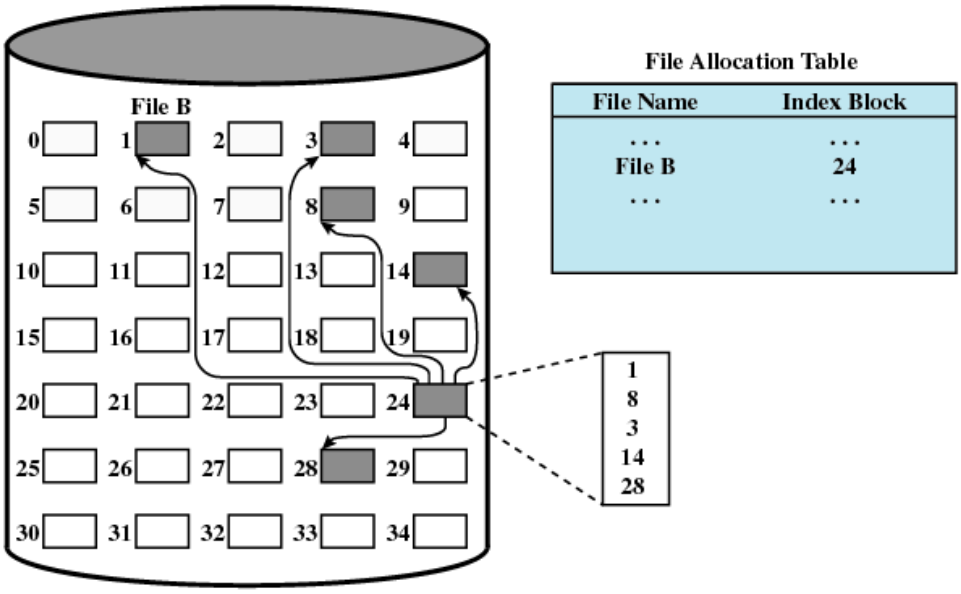


Figure 12.11 Indexed Allocation with Block Portions

- 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:

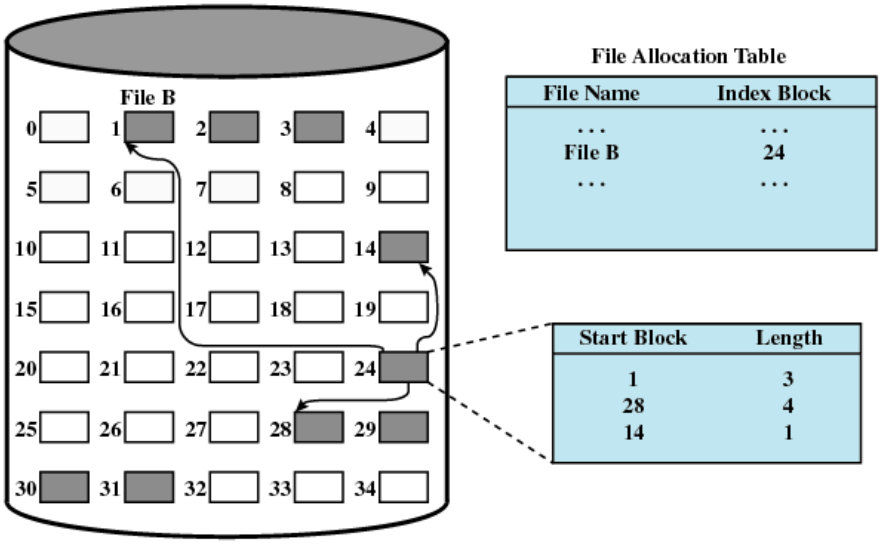


Figure 12.12 Indexed Allocation with Variable-Length Portions

4)