Introducción

Un SO es software que necesita procesador y memoria para ejecutarse, actúa como intermediario entre un usuario de una computadora y el HW de la misma.
Administra los recursos de la computadora, CPU, memoria y E/S. Hace más fácil el uso de la PC y un uso más eficiente de los recursos del sistema.
Se encarga de:

- Gestionar el HW
- Controla la planificación, administración y ejecución de los procesos
- Administración de almacenamiento y sistema de archivos
- Administración de dispositivos
- Interfaz entre aplicaciones y HW
- Permite incluir nuevas funciones al sistema, sin interferir en funciones anteriores
- Detecta errores de HW y SW y proporciona respuestas a los mismos

Los componente de un SO son: Kernel, shell (interacción con el usuario) y herramientas (editores, compiladores, librerías, etc).
El SO debe evitar que un proceso

- Se apropie de la CPU
- Intente ejecutar instrucciones de E/S
- Intente acceder a una posición de memoria fuera de su espacio declarado

Por este motivo el SO debe gestionar el uso de la CPU y detectar intentos de ejecución de instrucciones de E/S ilegales, accesos ilegales a memoria y proteger el vector de interrupciones, así como las rutinas de atención de las mismas.
Para esto existe cierto apoyo de HW

- Modos de Ejecución
- Protección de la memoria (lógica vs física)
- Interrupción de Clock (Protección de la CPU)

Modos de ejecución
El SO le ofrece al usuario un modo de “acceso seguro”, no solo seguro para terceros si no de si mismos. Este bit figura en el PSW (Program Status Word) que es un registro de control y de estado de la CPU. Este bit indica si se está ejecutando en modo Supervisor/Kernel o modo Usuario.

Las instrucciones privilegiadas deben ejecutarse en modo Supervisor/Kernel, ya que deben acceder a estructuras o ejecutar código que no son del propio proceso.
En modo Supervisor/Kernel hace manejo de

- CPU, memoria, operaciones de entrada/salida
- Administración multiprocesador, diagnósticos, testing
- Estructura del file system
- Comunicaciones: interfaz de red

En modo Usuario, el proceso puede acceder a espacio de direcciones “propias”.
El sistema arranca lo hace en modo Supervisor/Kernel, pero este bit debe modificarse a modo Usuario cada vez que comienza a ejecutarse un proceso del usuario, esto se realiza mediante una instrucción especial.
Cuando el usuario intenta ejecutar instrucciones privilegiadas que pueden causar problemas, el HW lo detecta como una operación ilegal y produce un trap al SO.
Cuando hay un trap o una interrupción, el modo se cambia a Supervisor/Kernel.
Las instrucciones de E/S son privilegiadas, deben ejecutarse en modo Supervisor/Kernel, deben ser gestionadas por el SO.

Protección de la memoria
El SO debe delimitar el espacio de direcciones de un proceso y poner límite a las direcciones que puede usar un proceso, para esto se utilizan, por ejemplo, un registro base y uno límite.
Debe proteger el espacio de direcciones de un proceso del acceso de otros procesos, si un proceso intenta acceder a un espacio de memoria que no le corresponde el HW genera un trap al SO.

Interrupción de Clock
La interrupción de Clock se utiliza para evitar que un proceso se apropie de la CPU, se implementa normalmente a través de un contador y un clock, el kernel le da valor al contador el cual se decrementa con cada click del reloj, y cuando llega a cero se expulsa el proceso de la CPU para así poder ejecutar otro proceso.

Procesos

Diferencia entre programa y proceso
Un programa es estático, no tiene PC y existe desde que se edita hasta que se borra.
Un proceso es un programa en ejecución, es dinámico, tiene PC y su ciclo de vida comprende desde que se "dispara" hasta que termina. Tiene una sección de código (texto), sección de datos (variables globales) y stacks, en general una modo usuario y otra en modo kernel (datos temporarios: parámetros, variables temporales y dirección de retorno).

Atributos de un proceso
- Pid y ppid (pid del padre)
- Id del usuario que lo "disparó"
- Si hay estructura de grupos, el id del grupo que lo "disparó"
- En ambientes multiusuarios, desde que terminal y quién lo ejecutó

PCB (Process Control Block)
Es una estructura de datos asociada al proceso, existe una por proceso, es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando se termina un proceso.
Contiene la siguiente información:
- Pid y ppid
- Valores de los registros de la CPU (PC, AC, etc)
- Planificación (estado, prioridad, tiempo consumido, etc)
- Ubicación (representación) en memoria
- Accounting
- E/S (estado, pendientes, etc)

Un proceso cuenta con:
- **Espacio de direcciones,** que es el conjunto de direcciones que ocupa para guardar: stacks, texto y datos, este espacio no incluye su PCB o tablas asociadas.
  Un proceso en modo usuario puede acceder sólo a su espacio de direcciones, pero en modo kernel puede acceder a estructuras internas o a espacios de otros procesos.
- **Contexto**, que incluye toda la información que necesita el SO para administrar el proceso y la CPU para ejecutarlo correctamente, son parte del contexto también, los registros de la CPU, PC, prioridad del proceso, si tiene E/S pendiente, etc.

Cambio de contexto
Se produce cuando la CPU cambia de un proceso a otro, esto lo realiza el Dispatcher, el cual debe resguardar los datos del proceso saliente, que pasa a espera, y debe cargar el contexto del nuevo proceso y continuar desde su última instrucción. Es tiempo no productivo de la CPU, y el tiempo que consume depende del soporte del HW.

Existe dos enfoques que influyen en el tiempo de cambio de contexto
- Enfoque 1, donde el kernel funciona como entidad independiente, se ejecuta fuera de todo proceso, el kernel tiene su propia región de memoria, stack, no es un proceso y se ejecuta como entidad privilegiada en modo supervisor.
- Enfoque 2, el kernel "dentro" del proceso, el kernel se encuentra dentro del espacio de direcciones de cada proceso, se ejecuta en el mismo contexto que algún proceso de usuario.

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 de E/S, finalización de lapso de tiempo, etc.

Su nombre proviene de la frecuencia de ejecución
Long term: controla el grado de multiprogramación, es decir la cantidad de procesos en memoria. Puede no existir y absorber la tarea el short term.
Medium term: si es necesario, reduce el grado de multiprogramación, saca temporalmente de memoria los procesos que sean necesarios para mantener el equilibrio del sistema (**swap in** y **swap out**).
Short term: decide a cuál de los procesos en la cola de listos se elige para que use la CPU.
Dispatcher y **Loader**, pueden no existir como módulos separados, pero su función debe cumplirse. El dispatcher hace cambio de contexto, cambio de modo de ejecución, despacha el proceso elegido por el short term y el loader, carga en memoria el proceso elegido por el long term.

Estados de un proceso
New: un usuario "dispara" un proceso, este 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 a ser cargado en memoria.
Ready: luego que el long term eligió al proceso para cargarlo en memoria, el proceso queda en estado ready, esperando por CPU, en la cola de procesos ready.
Running: el short term eligió el proceso para asignarle CPU, la cual tendrá hasta que termine el período de tiempo asignado, o hasta que necesite realizar alguna operación de E/S.
Waiting: el proceso necesita que se cumpla un evento para poder continuar, el evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso. El proceso sigue en memoria pero no tiene la CPU, y al cumplirse el evento pasará nuevamente al estado ready para competir nuevamente por la CPU.
Terminated: el proceso ha terminado.

Transiciones de estados de un procesos
New-Ready: por elección del long term, se carga en memoria.
Ready-Running: por elección del short term, se asigna 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: es un caso especial, cuando al proceso se le termina su quantum, sin haberse producido ningún evento.

Planificación de los procesos
Es necesario determinar cuál de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado, para esto se utilizan algoritmos de planificación, los cuales realizan la planificación del sistema, pueden ser:
Apropiativos: el proceso puede ser expulsado de la CPU. Utilizado por procesos interactivos, donde puede haber interacción con un usuario o un servidor que debe dar respuesta a diferentes requerimientos, de esta manera los procesos no acaparan la CPU.
Algoritmos: Round Robin, Prioridades, Colas multinivel y SRTF (al que le queda menos tiempo de procesamiento).
No apropiativos: el proceso se ejecuta hasta que el mismo decida abandonar la CPU. Mantienen la CPU ocupada la mayor cantidad del tiempo. Utilizado por procesos batch, donde no existen usuarios que esperen una respuesta en una terminal.
Algoritmos: FCFS (primero en llegar, primero en salir) y SJF (el proceso más pequeño primero).

Creación de procesos
Un proceso es creado por otro proceso, que su proceso padre y un proceso puede tener más de un hijo, por ende se forma un árbol de procesos.

1. Crear la PCB
2. Asignar pid

3. Asignar memoria para sus regiones: stack, texto y datos
4. Crear estructuras de datos asociadas: fork (copiar el contexto, regiones de datos, texto y stack)

Creación Unix, system call fork(), crea un nuevo proceso idéntico a su padre y luego la system call excve(), carga un nuevo programa en el espacio de direcciones.
En Windows, system call CreateProcess() que realiza los 2 pasos anteriores en uno.

Terminación de procesos
Ante un exit, se retorna el control al SO, el proceso padre puede terminar la ejecución de sus hijos (kill), cuando el padre termina sus hijos mueren en cascada o pueden seguir.

System Calls
Es una forma en que los programas de usuario acceden a los servicios del SO.
Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques o tablas de memoria o la pila. Las system calls son ejecutadas en modo Supervisor/Kernel.

Categorías de system calls
- Control de procesos: fork, waitpid, execve, exit.
- Manejo de archivos: open, close, read, write, iseek, stat.
- Manejo de dispositivos
- Mantenimiento de información del sistema,
- Comunicaciones

Para iniciar una system Call se indica el número de system call que se quiere ejecutar y los parámetros de esa system call, luego se emite una interrupción para pasar a modo kernel y así gestionar la system call, el manejador de interrupciones evalúa la system call deseada y la ejecuta.
Por ejemplo: una aplicación hace una llamada a la función printF(), esta función se convierte en llamada a write() de la librería de C (GLIBC), y esto a su vez en un write system call para el kernel linux.

Pasos que se suceden al llamar a una System Call
1. El proceso de usuario llama a una syscall por medio de la GlibC
2. La GlibC pone los parámetros para syscall en el stack y eleva una interrupción
3. Se cambia a modo kernel
4. Se coloca el PC y PSW en el stack del usuario (dependiendo de la arquitectura se pueden apilar otros registros)
5. Se coloca en el PC la dirección de la rutina de atención de interrupción que se extra de la IDT (Interrupts Description Table, es una tabla que se usa para manejar las interrupciones que se produzcan) y se continua la ejecución
6. Se sacan los parámetros de la syscall del stack del usuario
7. Se cambia a stack de kernel
8. Se colocan en el stack kernel los parámetros para la syscall y se ejecuta la syscall
   Si la syscall bloquea al proceso:
   - 8.1 Se ejecuta el short term para seleccionar un nuevo proceso
   - 8.2 Se realiza el cambio de contexto y se cargan los registros del nuevo proceso
   - 8.3 Se acomoda la dirección del stack, dejando apuntado a la dirección que el HW dejo previo a que el proceso seleccionado sea suspendido
9. Se cambia a modo usuario y pila de usuario
10. Se ejecuta RET
11. Se saca de la pila de usuario el PSW y PC
12. Continua la ejecución del proceso actual

Memoria

La administración de la memoria es una de las tareas más importantes del SO.
En los SO multiprogramados es necesario mantener varios programas en memoria al mismo tiempo. Existen varios esquemas para la administración de la memoria y requieren distintos soportes de HW. El SO es responsable de:

- Mantener qué partes de la memoria están siendo utilizadas y por quién
- Decidir cuáles procesos serán cargados a memoria cuando exista un espacio de memoria disponible
- Asignar y quitar espacio de memoria según sea necesario
- Protección de los datos para que los procesos no utilicen memoria que no les corresponden

Direcciones lógicas y direcciones físicas
Lógicas, referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.
Físicas, referencia a una localidad de memoria física (RAM). Dirección absoluta.
Una forma simple es utilizando registros auxiliares: Base, que tiene la dirección de comienzo del proceso en la RAM y Límite, dirección final o medida del proceso, ambos valores se fijan cuando es cargado en memoria, estos registros varían entre procesos. El problema que presenta es que hay que almacenar el espacio de direcciones en forma contigua en la memoria física, se produce fragmentación, la solución es: Paginación, segmentación.

Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal es necesario algún tipo de conversión a direcciones físicas:

- Resolución de direcciones (address-binding)
- Resolución en momento de compilación y en tiempo de carga, las direcciones físicas y lógicas son idénticas, para re ubicar un proceso es necesario recompilarlo y cargarlo.
- Resolución en tiempo de ejecución, las direcciones físicas y lógicas son diferentes, las direcciones lógicas son llamadas virtuales, el mapeo entre direcciones virtuales y físicas son realizadas por HW llamado MMU (Memory Management Unit)

MMU (Memory Management Unit)
Es un dispositivo de HW que mapea direcciones virtuales a físicas, es parte del procesador, reprogramarlo es una operación privilegiada, solo puede ser realizada en modo supervisor.
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, los procesos nunca usan direcciones físicas.

Paginación
La memoria física es dividida lógicamente en pequeños trozos de igual tamaño llamados “marcos”.
La memoria lógica (espacio de direcciones) es dividida en trozos de igual tamaño que los marcos llamadas “páginas”. De esta manera la cantidad de memoria desperdiciada por un proceso es el final de la última página, lo que minimiza la fragmentación interna y evita la externa.
Con esta técnica la memoria se encuentra ocupada con diferentes páginas de diferentes procesos.
El SO mantiene una lista con los marcos libres para su uso y una tabla de páginas por cada proceso, donde cada entrada contiene (entre otras) el “marco” en la que se coloca la página, de esta manera las páginas de un proceso pueden no estar contiguamente ubicadas en memoria, pueden intercalarse con páginas de otros procesos
La dirección lógica se interpreta como, un número de página y un desplazamiento dentro de la misma. El número de página es usado como índice dentro de una tabla de páginas y una vez obtenida la dirección del marco de memoria, se utiliza el desplazamiento para ubicar la dirección real en la memoria física. Este proceso lo realiza el MMU.

Paginación por demanda
El único inconveniente de la paginación pura es que todas las páginas de un proceso deben estar en memoria para que se pueda ejecutar. Esto hace que si los programas son de tamaños muy grandes, no puedan cargarse muchos a la vez, lo cual disminuye el grado de multipogramación del sistema. Para evitar esto y aprovechando el principio de cercanía de referencias, donde se puede esperar que un programa trabaje con un conjunto cercano de referencias a memoria (es decir con un working set más pequeño que el total de sus páginas), se permitirá que algunas páginas del proceso sean guardadas en un área de intercambio o memoria virtual, que es una zona del disco (memoria secundaria) reservada para alojar parte de los procesos que no caben en memoria principal mientras no se necesiten.
En este caso, el SO mantiene ademas el conocimiento sobre qué páginas están en memoria principal y cuáles no, usando una tabla de paginación. En cada entrada de la tabla de páginas (PTE o Page Table Entry), existe un bit de presencia (V) que estará en 1 si la página está en memoria principal o en 0 si no se encuentra, y otro bit de modificación (M) que advierte si la página fue modificada en memoria principal y debe descargarse a disco antes de abandonar la misma.
Cuando la CPU intenta acceder a una dirección de memoria lógica, el MMU, realiza una búsqueda en una memoria caché especial llamada TLB (Translation lookaside buffer) que mantiene las tablas de páginas usadas hace menos tiempo. En esta memoria están las PTE donde se pueden rescatas las direcciones físicas correspondientes a algunas direcciones lógicas de forma directa. Cuando la dirección requerida por la CPU se encuentra en la TLB se entrega y se lo conoce como un “TBL hit”, caso contrario es un fallo en TLB, lo cual es informado al SO mediante un “fallo de página”, de esta manera el SO busca en el área de páginas de intercambio y la carga en memoria física usando alguno de los algoritmos de reemplazo de páginas, y luego continua con la ejecución desde la instrucción que causó el fallo.

Ventajas de la paginación
- Transparente al programador
- Elimina fragmentación externa
- Posibilidad de cargar solo una parte del programa en memoria, el resto se cargará cuando se necesiten
- Facilidad de controlas las páginas ya que tienen todas el mismo tamaño
- No es necesario que las páginas estén contiguas en memoria

Desventajas de la paginación
- El costo de HW y SW se incrementa por la traducción de direcciones
- Problema de fragmentación interna
- Si ocurren muchas fallos de página la performance baja notablemente

Segmentación
Esquema que soporta el punto de vista del usuario. Un programa es una colección de segmentos. Un segmento en una unidad lógica como: programa principal, procedimientos y funciones, variables locales y globales, stacks, etc. Puede causar fragmentación.
Todos los segmentos de un programa pueden no tener el mismo tamaño (códigos, datos, rutinas).
Las direcciones lógicas consisten en dos partes: selector de segmento y desplazamiento dentro del segmento.
Tenemos una tabla de segmentos, que permite mapear la dirección lógica con la física. Cada entrada contiene: Base, dirección del comienzo del segmento y limite, longitud del segmento.

Segmentación por demanda
Igual que la paginación por demanda, un proceso no requiere que todos sus segmentos estén en memoria principal para poder ejecutarse. Por ello el selector de segmento posee un bit de validez para segmento, el cual indica si la página se encuentra actualmente en memoria.

Cuando un proceso direcciona un segmento, el MMU examina el bit de validez en la tabla de segmentos, si el segmento esta en memoria principal el acceso continuo sin problemas, en caso de que no esté se genera un trap al SO (falla de segmento) igual como sucede en la paginación por demanda.
Las rutinas de administración de memoria deberán ver si hay lugar para alojar al nuevo segmento, si no lo hay se efectúa una compactación. Si luego de la compactación sigue sin haber espacio, se descarga un segmento al área de intercambio y se incorpora el nuevo segmento a memoria principal.

Ventajas de segmentación

- Visible al programador
- Facilita la modularidad, estructuras de datos grandes y da soporte a la compartición y protección
- El programador puede conocer las unidades lógicas de su programa
- Fácil compartir segmentos

Desventajas de segmentación

- Problemas de fragmentación externa ya que los segmentos son de tamaño variables
- Se complica el manejo de memoria virtual, ya que los discos almacenan la información en bloques de tamaños fijos, mientras los segmentos son de tamaño variable
- Ahorra memoria, pero requiere de mecanismos adicionales de HW y SW
- Dificultad de traer un nuevo segmento a memoria, ya que será necesario encontrar un área de memoria libre ajustada a su tamaño (por su tamaño variable)

Segmentación Paginada

Para la segmentación se necesita que estén cargadas en memoria áreas de tamaños variables. Si se requiere cargar un segmento en memoria que antes estuvo en ella y fue removido a memoria secundaria, se necesita encontrar una región de la memoria lo suficientemente grande para contenerlo, lo cual no es siempre factible. En cambio recargar una página implica sólo encontrar un marco de página disponible.
A nivel de paginación, si se quieren referenciar en forma cíclica n páginas, estas deberán ser cargadas una a una, generándose varias interrupciones por fallos de páginas. Bajo segmentación sólo se cargará la pagina que contiene los ítems referenciados.
Puede hacerse una combinación de segmentación y paginación para obtener las ventajas de ambas. En lugar de tratar un segmento como una unidad contigua, éste puede dividirse en páginas. Cada segmento puede ser descrito por su propia tabla de páginas.
Las direcciones tienen tres componentes: s,p y d, donde la primer indica el número de segmento, la segunda el número de página dentro del segmento y la tercera el desplazamiento dentro de la página.

Ventajas de segmentación paginada

El esquema de segmentación paginada tiene todas las ventajas de la segmentación y la paginación:

- Debido a que los espacios de memorias son segmentados, se garantiza la facilidad de implantar la compartición y enlace.
- Como los espacios de memoria son paginados, se simplifican las estrategias de almacenamiento
- Se elimina el problema de fragmentación externa y la necesidad de compactación

Desventajas de la segmentación paginada

- Los tres componentes de la dirección y el proceso de formación de direcciones hace que se incremente el costo de su implantación. El costo es mayor que en el caso de segmentación pura o paginación pura

- Se hace necesario mantener un número mayor de tablas en memoria, lo que implica un mayor costo de almacenamiento
- Sigue existiendo el problema de fragmentación interna de todas, o casi todas las páginas finales de cada uno de los segmentos. Bajo paginación pura se desperdicia sólo la última página asignada, mientras que bajo segmentación paginada el desperdicio puede ocurrir en todos los segmentos asignados

Memoria Virtual
Para poder tener MV el HW 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 están 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.
Cada proceso tiene su tabla de páginas, y cada entrada en la tabla hace referencia al marco en el que se encuentra en memoria principal y tiene bits de control, bit V y M.

Fallo de página
Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en memoria principal, es decir la pagina no se encuentra en su conjunto residente, el bit V=0, este bit es controlado por HW y cuando este detecta la situación genera un trap al SO, entonces el SO podrá colocar este proceso en blocked o espera, mientras se gestiona el pedido de la pagina que necesita que se cargue.
El SO busca un marco libre en la memoria y genera una operación de E/S al disco para copiar en dicho marco la página del proceso que se necesita utilizar, mientras tanto el SO puede asignar la CPU a otro proceso mientras se completa la E/S. Una vez que la E/S finalice avisará mediante interrupción su finalización.
Una vez finalizada la E/S, se notifica al SO y este debe actualizar la tabla de páginas del proceso, coloca el bit V=1 y la dirección base del marco donde se colocó la página.
El proceso que generó el fallo de página vuelve a estado ready y cuando se vuelva a ejecutar lo hará desde la instrucción que antes generó el fallo de página.

Tabla de páginas
Cada proceso tiene su tabla de paginas, el tamaño de la misma depende del espacio de direcciones del proceso, puede alcanzar un tamaño considerable.
Formas de organizar las tablas:
- Tabla de un nivel, única lineal
- Tabla de dos niveles o más multinivel
- Tabla invertida, hashing

La forma de organizarla depende del HW.

Tamaño de la página
Pequeño: menor fragmentación interna, se requieren más páginas por proceso, lo que lleva a tablas de páginas muy 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 mover páginas a memoria principal.

Translation Lookaside Buffer (TLB)
Es una memoria caché de alta velocidad usada para almacenar entradas de páginas. 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 se encuentra se produce un hit, se obtiene el marco y es armada la dirección física, de lo contrario se produce un miss, se verifica si la página está en memoria, si no se genera un fallo de página, la TLB es actualizada para incluir la nueva entrada.

Asignación de marcos
Dinámica, el número de marcos para cada proceso varía.
Fija, número fijo de marcos para cada proceso.
Equitativa cantMarcos/cantProcesos.
Proporcional, se asigna acorde al tamaño del proceso.

Reemplazo de páginas
Cuando se genera un fallo de página y todos los marcos están ocupados, se debe seleccionar una página víctima.
Reemplazo global, el fallo de página de un proceso puede reemplazar la página de cualquier proceso. El SO no controla la tasa de fallos de página de cada proceso. De esta manera el proceso puede tomar marcos de otro proceso aumentando su cantidad de marcos asignados a él.
Reemplazo local, el fallo de página de un proceso solo puede reemplazar sus propias páginas, de su conjunto residente. No cambia la cantidad de marcos asignados. El SO puede determinar la cantidad de fallos de páginas de cada proceso. Un proceso puede tener marcos asignados que no usa y que no pueden ser utilizados por otros procesos.

Algoritmos de reemplazo
- Optimo, quitar de memoria la página que no será utilizada en más tiempo, algoritmo teórico.
- FIFO, primera página en entrar, primera en salir.
- LRU (Last recently used), quita las páginas menos usadas recientemente.
- 2da chance, mejora de FIFO pero utilizando el bit de presencia como respaldo.
- NRU (Non recently used): utiliza bits de R y M.

Hiperpaginación (Trashing)
El sistema está en trashing cuando pasa más tiempo paginando que ejecutando procesos, como consecuencia hay una baja en la performance del sistema.

Ciclo de Trashing
1. El kernel monitorea el uso de la CPU
2. Si hay baja utilización, aumenta el grado de multiprogramación
3. Si el algoritmo de reemplazo es global, pueden sacarse marcos a otros procesos
4. Un proceso necesita más marcos y comienzan los fallos de página y robo de marcos a otros procesos
5. Por swapping de páginas, y encolamiento en dispositivos, baja el uso de la CPU
6. Vuelve a 1

Se puede controlar el trashing utilizando algoritmos de reemplazo local, de esta manera no se roban los marcos a otros procesos, pero se sigue perjudicando la performance del sistema, aunque es controlable.

Estrategias para controlar Trashing
- **Working Set**, se apoya en el modelo de localidad, tiene una ventada de WS con las referencias de memoria más recientes. Si la ventana es chica no cubrirá la localidad, si es grande pude tomar varias localidades. El problema es que se deben mantener registros para poder lograr el control, la ventana es móvil.
- **Algoritmo PFF** (frecuencia de fallos de página), en base a la frecuencia con que se dan los fallos de página si es alta, se necesitan más marcos y si es baja los procesos tiene muchos marcos asignados.
- **Demonio de paginación**

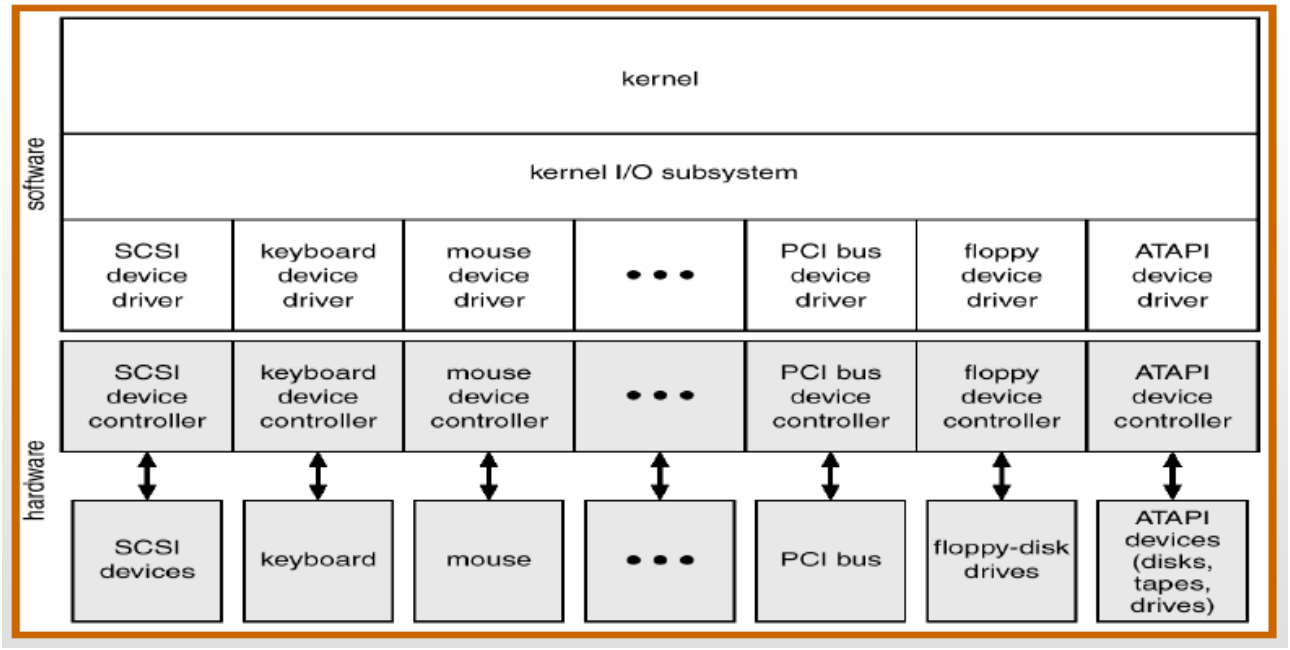
Subsistema de Entrada/Salida

El subsistema de E/S (SW) es el encargada de hacer la E/S, cuando se le pide algo se encarga de comunicarse con el driver del dispositivo asociado adecuado. El driver del dispositivo (SW), es quien se comunica directamente con el dispositivo en cuestión (HW).
Es deseable manejar todos los dispositivos de E/S de una manera uniforme, estandarizada.
Es más simple que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock, pero un proceso no puede realizar una E/S directa, lo hace con participación del SO.
Los dispositivos de E/S son más lentos respecto a la memoria y CPU, de esta manera el uso de la multiprogramación permite que mientras un proceso espera por la finalización de la E/S, otro proceso pueda tomar la CPU para ejecutarse.
El SO mantiene información de estado de cada dispositivo como: archivos abiertos, conexión a la red, etc.

Características de los dispositivos de E/S
Modo de transferencia

- Por bloques (discos): operaciones read, write , seek.
- Por caracter (teclado, mouse, puertos serie): operaciones get, put.

Tipo de acceso: Secuencial o aleatorio.
Tipo de transferencia: Sincrónico o asincrónico.
Si es un recurso compartido o no
Velocidad: Latencia, tiempo de posicionamiento, velocidad de transferencia, espera entre operaciones.
Dirección posible de la E/S: solo lectura, solo escritura, ambas.



Servicios del subsistema de E/S
Planificación: organización de los requerimientos a los dispositivos, por ejemplo los requerimientos al disco para minimizar tiempos.
Buffering: almacenamiento de los datos en memoria mientras se transfieren, esto soluciona los problemas de velocidad y tamaño 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, ya que algunos dispositivos no pueden atender distintos requerimientos al mismo tiempo como por ejemplo la impresora. Este mecanismo es utilizado para coordinar los accesos concurrentes al dispositivo.
Reserva de dispositivos: acceso exclusivo.
Manejo de errores: el SO debe administrar el manejo de errores por ejemplo cuando se produce un error en la lectura de un disco, en general se devuelve un código de error ante esta situación. Logs de errores.

Formas de realizar E/S
Bloqueante: el proceso queda suspendido hasta que el proceso de E/S se complete.
No bloqueante: el requerimiento de E/S retorna cuando es posible. Por ejemplo una aplicación de video que lee frames desde un archivo mientras va mostrándolos en pantalla.

Desde el requerimiento hasta el HW
Consideremos la lectura sobre un archivo en un disco:

1. Determinar el dispositivo que almacena los datos, traducir el nombre del archivo en la representación del dispositivo
2. Traducir el requerimiento abstracto en bloques de disco (filesystem)
3. Realizar la lectura física de los datos (bloques) en la memoria
4. Marcar los datos como disponibles al proceso que realizó el requerimiento, desbloquearlo
5. Retornar el control al proceso

Drivers
Contienen el código dependiente del dispositivo y manejan un tipo de dispositivo. Hacen la traducción de los requerimientos abstractos en los comandos para el dispositivo. Comúnmente las interrupciones de los dispositivos están asociadas a una función del driver.
Son una interfaz entre el SO y el HW, forman parte del espacio de memoria del kernel, en general se implementan como módulos y se cargan dinámicamente. Para agregar un nuevo HW, basta indicar el driver correspondiente sin necesidad de realizar cambios en el kernel.
El driver tiene registros de control y datos, la CPU escribe y lee estos registros para comunicarse.
En Linux debe contar con las operaciones init_module para instalarlo y cleanup_module para desinstalarlo.

Existe una variedad entre los dispositivos de E/S, entre ellos tenemos:

- **Legibles para el usuario**, los cuales se utilizan para comunicarse con el usuario como la impresora, pantalla, teclado, mouse.
- **Legibles para la máquina**, que son utilizados para comunicarse con los componentes electrónicos como los discos, sensores, etc.
- **Comunicación**, usados para comunicarse con dispositivos remotos como las lineas digitales, interfaces de redes, módems, etc.

Linux distingue tres tipos de dispositivos
Caracter: E/S programada o por interrupciones.
Operaciones más comunes:

- open, para abrir un dispositivo
- release, cerrar el dispositivo
- read, leer bytes del dispositivo
- write, escribir bytes en el dispositivo
- ioctl, orden de control sobre el dispositivo

Bloque: DMA
Red: puertos de comunicaciones

La E/S es uno de los factores que más afectan la performance del sistema, utilizan mucho la CPU para leer los drivers y el código de subsistema de E/S, provoca cambios de contexto ante las interrupciones y bloqueos de los procesos.
Para poder mejorar la performance se debe reducir el número de cambios de contexto, la cantidad de copia de los datos mientras se pasan del dispositivo a la aplicación y reducir la frecuencia de interrupciones utilizando: transferencias de gran cantidad de datos, controladoras más inteligentes y polling para reducir la espera activa. Utilización de DMA.

E/S Programada
La CPU tiene control directo sobre la E/S, controla el estado, comandos para leer y escribir, transfiere los datos. La CPU espera que el componente de E/S termine la operación, de esta manera se desperdician ciclos de CPU. **Polling**, en la entrada salida programada es necesario hacer polling del dispositivo para saber su estado, esto es muy costoso si es la espera es muy larga.

E/S Interrupciones
Soluciona el problema de la espera de la CPU, ya que la CPU no hace el checkeo sobre el dispositivo y continua con la ejecución de instrucciones, una vez que el componente de E/S termina, envía interrupción.

DMA (Direct memory access)
Evita polling y el consumo de CPU.
La CPU manda comandos al controlador de DMA para iniciar una transferencia DMA, luego de eso la CPU queda libre.
El controlador DMA es el que habla con el dispositivo y se genera la transferencia de datos con la Memoria, una vez que finalizada la transferencia se genera una interrupción a la CPU.
La CPU y el controlador de DMA comparten los buses de dirección y datos, la CPU puede acceder a la memoria caché.
Controlador de DMA es un soporte de HW.

Sistema de Archivos

Un archivo es la mínima unidad lógica de almacenamiento persistente de información, ya que se puede almacenar en memoria secundaria, puede ser recuperado para su actualización y modificación.

¿Cómo se identifica un archivo?

Los archivos se dividen en dos partes, los **datos** y los **metadatos**, los datos es la información que se quiere almacenar y los metadatos es la información de identificar el archivo.

Atributos de un archivo

Nombre: utilizado por los usuarios para poder identificar un archivo unívocamente dentro de un directorio.
Id: cada sistema de archivos debe tener un identificador que identifique de manera única a un archivo, usualmente es un número, esto es utilizado por el SO.
Opcionales:
Extensión del archivo
Ubicación, dónde están ubicados físicamente los datos de ese archivo.
Tamaño, es el tamaño actual de archivo, cuánto ocupa, dependiendo del sistema de archivos es la medida representada, generalmente se mide en bytes, pero puede ser palabras o bloques.
Bits de protección, qué y quiénes pueden hacer cosas con él, por ejemplo, que usuario puede acceder y en qué forma.
Loguin, para saber quién lo creo, cuándo se creó, se modificó, se leyó por última vez, sirve por ejemplo si un archivo no se usa hace mucho tiempo se puede eliminar.

Operaciones sobre los archivos

El SO brinda servicios para la manipulación de archivos:

- **Crear y abrir**
- **Escribir y leer**, en o de un archivo abierto previamente, habitualmente de manera secuencial
- **Reposicionar dentro de un archivo**, poder acceder a cualquier parte del archivo, el cual debe haber sido previamente abierto
- **Eliminar**
- **Truncar**, borrar todos los datos, pero dejar los metadatos, comenzar el archivo de cero conservando la información de los metadatos
- **Cerrar**, se quita la referencia del archivo en la tabla de archivos abiertos

Por lo general, los sistemas tienen una tabla de archivos abierto por proceso. Estos archivos se abren a través de un llamado al sistema y, de esa forma, se pueden operar con ellos (leer, escribir, etc.). Finalmente, el archivo es cerrado antes de que finalice la ejecución del proceso.
La tabla está a nivel de sistema y a nivel de proceso, por ejemplo, si se abre en un proceso un archivo que ya utilizó otro proceso, no se va a necesitar acceder a memoria.
Cuando se abre un archivo, se tiene un puntero al mismo (file pointer). Se tiene que conocer la ubicación en el dispositivo. Algunos sistemas proveen sistema de acceso único a un archivo (lock) por parte de los procesos, esto se refiere a que un archivo puede ser abierto en modo exclusivo, entonces no puede ser abierto por nadie más.

Métodos de acceso

Secuencial, cuando se abre un archivo se puede leer secuencialmente y no se puede ir a una posición más adelante del archivo, sin pasar por el medio, y lo mismo cuando se escribe, es algo que hoy en día no se fuerza ya que se usaba antes con las cintas.
Directo, alguien puede leer o escribir la posición que quiera dentro del archivo.

Directorios
El archivo se guarda en el disco, organizado en el sistema de archivos y particularmente lo pongo en un directorio, es toda la estructura donde se puede colocar el archivo.
Un sistema de archivos tiene un directorio, que tiene información de archivos que tiene ahí y el conjunto de archivos con los datos que hay en el directorio, esto puede estar almacenado en un disco o en varios discos.

Operaciones sobre los directorios
Búsqueda, es necesario poder buscar un archivo en un directorio
Crear un archivo, archivos nuevos deben ser creados e incorporados al directorio
Eliminar
Listar, visualizar los archivos que están en un directorio
Renombrar un archivo, cambiar el nombre de un archivo dentro de un directorio
Permitir la navegación, lograr acceder a todos los directorios de archivos y navegarlos

Estructuras de directorios
Nivel único, el esquema más sencillo es tener un único nivel de directorios en el sistema de archivos. Sirve para sistema monousuarios. No se pueden tener archivos con el mismo nombre.
Árbol, es ideal permitir varios niveles de directorios. Esto se logra permitiendo tener archivos de tipo directorio dentro de los directorios. Se genera una estructura jerárquica de directorios en forma de árbol. Ahora identifica a un archivo toda su ruta (path) absoluta, que es el camino desde la raíz hasta el archivo.
Si es un sistema multiusuario, cada usuario tiene su nivel de directorio y cada uno tiene sus archivos, en este caso pueden nombrar de la misma manera.
El proceso debe saber en qué directorio está parado actualmente, puede acceder de forma relativa utilizando los dos puntos para volver una carpeta hacia atrás.
El esquema de árbol es el que se utiliza hoy en día, es simple acceder desde la raíz hacia una hoja, pero desde una hoja hacia otra hoja es difícil.
Grafo, para potenciar la estructura anterior de árbol sería deseable tener caminos de acceso directo a otros directorios. Estos caminos se logran a través de archivos de tipo enlace simbólicos (soft links). A su vez, se permite que un archivo esté en más de un directorio (hard link).
Soft link, es un archivo que adentro dice a dónde realmente se quiere ir.
Hard link, los metadatos del archivo apuntan físicamente a los mismos datos que el otro, no hay una referencia a otro archivo si no que son dos metadatos de archivos que tienen un puntero que apuntan a los datos. El SO lleva una cuenta de los enlaces duros que se tiene, ya que al borrarse se pueden perder los datos, de esta manera no se puede diferencias el original y la copia.
Montaje de directorios, dada la estructura de grafo, los sistemas de archivos se pueden solapar en un único sistema de archivos.

Seguridad en archivos
Debido a que el sistema es multiusuario es necesario proteger la información de cada usuario.
En muchos casos los usuarios se agrupan según el uso que tienen sobre un sistema.
Se definen permisos sobre los archivos tanto a nivel de usuario como de grupo.
Los permisos más comunes son: Escritura, lectura, ejecución, eliminar y listar.

Implementación
Diagramar el sistema de archivos en capas, las cuales puedo resolver independientemente una de otra.
Dispositivos físicos, donde van a estar almacenas los datos
Capa de control de los dispositivos, como le hablo a los dispositivos físicos para que hagan cosas, por ejemplo, le dice al disco como leer o escribir un bloque (subsistema de E/S)

Sistema de archivos básicos, tiene información sobre que bloques tiene que leer, mantiene los buffers con la información
Módulo de organización de archivos, tiene el control sobre un archivo, sabe cómo está dividido dentro del disco
Sistema de archivos lógicos, tiene la información como atributos de protección, la estructura de directorio, como moverse para llegar a tal o cual archivo
Programas de aplicación que usan todos mis sistemas de archivos, la que utilizan los usuarios

Los dispositivos físicos (discos) contienen la siguiente estructura
Bloque de control para el boot, es necesario para lograr iniciar el SO
Bloque de control de partición, contiene la información de las particiones que existen en el disco, bloques utilizados y libres, cantidad de archivos, etc
Estructura de directorios, para la organización de los archivos
Bloque de control del archivo, los bloques de control de los archivos que están en el sistema de archivos, tiene la información que preciso para acceder al archivo

Sistemas de archivo virtual
Es común que un SO se acceda a más de una implementación de sistema de archivos, para esto se utilizan técnicas de orientación a objetos para lograr mantener una estructura independiente del sistema de archivos que se utilice.

Se genera una estructura en tres capas
Interfaz del sistema de archivo (open, read, etc.). Donde se brinda al usuario final poder realizar las operaciones relacionadas con los archivos.
Sistema de archivos virtual (Virtual File Server). Capa intermedia entre el usuario y el sistema de archivo. Conoce todos los sistemas que están conectados a él y sabe cómo manejarlos.
Implementación específica del sistema de archivo, se registra en la interface intermedia para que ella pueda manejarla.

Estructura de los directorios
Los directorios contienen la información de los archivos que pertenecen a él.
Para organizar la información existen varias alternativas:
Lista encadenada, los nombres de los archivos y un puntero al bloque de control son dispuestos en una lista encadenada. Es usual el uso de caches en memoria principal para acelerar el acceso. Se puede organizar como una lista ordenada, para poder ubicar más rápidamente, pero tiene un costo adicional.
Tabla de hash abierto, con el nombre del archivo se genera la clave utilizada.

Métodos de asignación
Para la disposición de los datos de los archivos en disco se tienen, en general, tres métodos:
Asignación contigua, los datos son dispuestos en forma contigua. Para mantener la información es necesario saber en qué bloque comienza y la cantidad de bloques que tiene el archivo. Se crea un archivo y se debe indicar el tamaño máximo que va a ocupar, para que se le pueda asignar un lugar en el disco con cierta cantidad contigua de bloques. Hay fragmentación externa. El archivo puede crecer más del tamaño máximo especificado y debe reubicarlo. Se pueden utilizar bloques denominados extents para extender el tamaño. Se accede rápidamente al archivo.
Asignación en forma de lista, los bloques de datos forman una lista encadenada. Es necesario una referencia al primer y último bloque de datos en el bloque de control de archivo. Se libera el problema de la fragmentación externa. Pero si se quiere acceder la forma es lineal, los punteros ocupan espacio en el bloque y la perdida de una referencia puede generar pérdida de información del archivo. El acceso directo es muy lento.

Ej.:FAT (file allocation table), una tabla indexada por cada sistema de archivo, que contiene las referencias a los bloques, es decir todos los punteros en una tabla todo junto. El acceso es más fácil. Debe existir un marcador que nos indique finaliza el archivo, también se necesita saber qué bloques están libre y qué ocupados.
Asignación indexada, se mantiene una tabla en donde cada entrada referencia a un bloque de datos. Los bloques son accedidos directamente a través del bloque de indexación, que es una tabla que cuenta con todos los punteros con sus correspondientes bloques de datos. El bloque de indexación ocupa lugar, se trata de que sea lo más pequeño posible, pero limita la cantidad de bloques. Una posible solución es indexar en varios niveles, algunos índices apuntan a bloques directos y otros a bloques de indexación.
Se mejora la eficiencia, respecto a la lista encadenada. Como desventaja es que si pierdo el bloque de indexación pierdo toda la información.
En Unix el file content block se llama i-nodo, el mismo es de hasta 3 niveles de indexación, lo cual permite representar chivos muy grandes.

Administración del espacio libre del sistema de archivos
En el sistema de archivos es necesario conocer qué bloques están ocupados y cuáles están libres para esto se pueden utilizar las siguientes alternativas:
Vector de bits, se dispone de un bit para cada bloque de datos del sistema, que representa si está ocupado o libre. Es eficiente, pero debe ser guardado en el disco y se lo trae a memoria cuando se lo requiere. Su tamaño puede volverse pesado. Es utilizado en Unix.
Lista de bloques libres, se mantiene una lista encadenada con los bloques libres a través de los bloques. Es necesario una referencia al primer bloque. Funciona bien si se pide de un bloque libre.
Agrupación, es una variación de la lista encadenada. En cada bloque de la lista se contiene un grupo de bloques libres. Si se necesitan más bloques se tienen que leer muchos punteros, la solución es que todos los punteros se pueden agruparse en un bloque, es decir un puntero al primer bloque libre que ya sabe todos los bloques libres.
Conteo, se mantiene una lista en donde cada bloque contiene información de cuantos bloques contiguos, a partir de él, están libres. Es más fácil para asignar varios bloques contiguos. Si hay demasiada fragmentación externa no funciona bien.

Buffer caché

El objetivo es en un marco de multiprogramación y multiusuario, tenemos que compartir información que esté en disco.
Cuando estábamos en E/S, el proceso que necesitaba hacer una E/S era llevado temporariamente a memoria secundaria, cuando tenia esa E/S pendiente, si mientras ese proceso estaba swapeado y se traía el requerimiento pero el proceso ya no estaba en memoria, entonces donde se ponía el requerimiento, ante esta alternativa había dos posibilidades, o no se permitía swapear un proceso con E/S pendiente o que lo que se trae sea almacenado en un espacio de direcciones que no es del proceso, un espacio del sistema, esto es buffer cache, es un área de memoria RAM que se destina a guardar bloques de disco que se hayan solicitado.
Cuando un proceso hace un requerimiento de un bloque de disco, ese bloque permanece en memoria más allá del tiempo que lo haya necesitado ese proceso especifico que generó la solicitud, el proceso lo lee e incluso puede modificarlo, pero no es llevado a disco automáticamente, permanece en memoria más tiempo. Los procesos comparten un bloque de disco en memoria, y se minimiza la frecuencia de acceso al disco, esa información es compartida y cada determinado tiempo es llevada a disco. Se tiene información en memoria que quizás fue actualizada, y no se actualizó en disco, quizás puede existir un corte de energía y se pierden esas modificaciones, esto sería un problema.
Se lee un bloque de disco y se carga en esta caché en memoria RAM, una vez que se copia se puede copiar al espacio de direcciones del proceso, entonces el proceso tiene una copia del bloque en su espacio de direcciones, pero esto no permitiría compartirlo. La otra posibilidad es que permanezca en el buffer cache y que de ahí sea compartido por los procesos que lo necesitan.
El área de buffer caché es limitada, entonces los buffers pueden no alcanzar para todas las solicitudes de disco, si no hay buffer libre se tiene que elegir un buffer víctima, el ideal sería el que hace más tiempo que no es utilizado, para respetar esto, se encolan los buffers de manera que el primero de la lista es el que hace menos tiempo se utilizó y el último de la lista es el que se utilizó recientemente, otra alternativa es eliminar el menos frecuentemente utilizado.

Buffer caché en Unix System V

Minimizar la frecuencia al disco, es una estructura formada por buffers los cuales están en área de la memoria RAM. Un buffer cuenta con dos partes: header que da información sobre el bloque que está dentro y su relación con otros buffers, y un lugar dónde se almacena el bloque de disco traído a memoria.
El header tiene 5 punteros, 2 punteros para el hash queue (relación con otros buffers), 2 punteros para la free list (relación con buffers libres) y 1 puntero al bloque de memoria.
Un buffer puede tener varios estados: libre (está en la free list), ocupado (siendo usado por uno más procesos, no está en la free list), que se está escribiendo o leyendo de disco en es momento, modificado (parecido al bit M en paginación) esto se llama delayed write.
La free list tiene los buffers que se encuentran libres, no necesariamente vacíos, pero esos bloques de disco ya no están siendo utilizados. El primer buffer de la free list, es el buffer libre que hace más tiempo no es usado.
Las hash queue son colas para buscar un buffer en particular, se organizan los headers de los buffers en colas, entonces son consultadas para saber si el bloque solicitado se encuentra ahí, se organizan utilizando una función de hash y relaciona aquellos headers que dan los mismos resultados.