CONCEPTOS DE SISTEMAS OPERATIVOS

Unidad 1- Introducción

Un sistema operativo es un programa que controla la ejecución de aplicaciones y procesos, que actúa como interfaz entre las aplicaciones y el hardware del computador, así como con los usuarios finales. El sistema operativo es software. Necesita de procesador y memoria para ejecutarse. Se puede considerar que los sistemas operativos tienen los siguientes objetivos:

- Facilidad de uso: simplifica la utilización del computador
- Eficiencia: gestiona los recursos de la computadora de manera eficiente.
- Evolución: Permite la implementación introducción y desarrollo de nuevos módulos sin interferir con sus servicios.

El kernel es el núcleo del sistema operativo. Se encuentra cargado en memoria principal y es el encargado de la administración de recursos. Sus servicios básicos son: El manejo de la memoria RAM en general, la administración de los procesos, la comunicación y la concurrencia, la gestión del hardware.

Los servicios básicos de todo sistema operativo son:

- Administración y planificación del procesador: imparcialidad en la ejecución de procesos, que no haya bloqueos y que se respeten las prioridades de los procesos.
- Administración de la memoria: Que se gestione la memoria física y virtual. Que se controle la jerarquía de memorias. Que haya multiprogramación.
- Administración del sistema de archivos: Acceso a medios de almacenamiento
- Administración de los dispositivos: Ocultamiento de las características del hardware, administración de procesos simultáneos.
- Detección de errores: errores de memoria y dispositivos, errores aritméticos y accesos no permitidos a memoria.
- Contabilidad: Recoger datos estadísticos del uso del CPU, monitorizar parámetros de rendimiento, anticipar mejoras futuras.

La Multiprogramación es el paradigma de gestión de memoria, en el cuál muchos procesos pueden convivir en memoria principal, compitiendo por el procesador y ahorrando tiempo productivo de reloj. Mediante la multiprogramación, la memoria está cargada con procesos listos, y en caso de que un proceso realice una operación de E/S o se bloquee en forma temporal, el procesador puede aprovechar ese tiempo atendiendo a otros procesos de la cola de listos.

Problemas que un S.O debe evitar: el sistema operativo debe proteger el vector de interrupciones, así como las rutinas de atención a interrupciones. Debe proteger la memoria de ser accedida en posiciones no permitidas o reservadas. Debe proteger los espacios de direcciones. Debe evitar que un proceso se apropie del CPU.

a) Protección de la memoria RAM: Delimitar el espacio de direcciones de cada proceso. Se deben poner límites a las direcciones utilizables por un proceso. Por ejemplo, un registro base y un registro límite. El S.O carga estos límites mediante instrucciones en modo privilegiado. Osea, en modo kernel.
b) Protección de la entrada/salida: Las instrucciones de E/S se definen como privilegiadas. Deben ser ejecutadas por el sistema operativo en modo kernel.
c) Protección de la CPU: El uso de interrupciones por clock para evitar que un proceso se apropie de la CPU. Se implementa mediante un clock y un contador. El S.O le da un valor al contador que se decrementa con cada ciclo de reloj. Al llegar a cero se produce una expulsión del proceso.

SYSTEMCALLS:

Perfeccionan una interfaz con la que los programas de usuario pueden invocar los servicios que el sistema operativo ofrece. Estas llamadas están escritas en C y C++, aunque determinadas tareas de las de nivel pueden necesitar escribirse con lenguaje ensamblador. Es como una llamada a procedimiento solo que entra al **kernel**.

Tipos de llamadas al sistema:

- Control de proceso: fin, ejecutar, cerrar, finalizar, organizar y liberar memoria.
- Manipulación de archivos: crear, eliminar, abrir, cerrar, leer, escribir.
- Manipulación de dispositivos: solicitar, liberar, leer, escribir.
- Mantenimiento de información: obtener y establecer las fechas, datos del sistema, atributos de un proceso.
- Comunicaciones: crear, enviar y recibir mensajes.

Funcionamiento general: Cómo pueden ser implemetadas?

Si un proceso está ejecutando un programa de usuario (en modo usuario) y necesita un servicio del SO (por ejemplo, leer datos de un archivo), tiene que ejecutar una instrucción de **trap** para transferir el control al sistema operativo. La interfaz de llamada al sistema intercepta la llamada a función dentro de la API e invoca la llamada al sistema necesario. Cada llamada al entrar tiene alocada un número y la interfaz de llamadas al sistema mantiene una tabla indexada según dichos números. Haciendo esta tabla, la interfaz de llamadas al sistema invoca la llamada necesaria del kernel del SO (que se ejecutará en modo kernel) y devuelve el estado de la aplicación de la llamada al sistema y los probables valores de retorno. Cuando se ejecuta una llamada al sistema, el HW la trata como una excepción, que efectúa una transferencia a una posición específica en el vector de interrupción.

Para pasar los parámetros adecuados a las llamadas, se pueden usar registros, bloques o talas en memoria o pila. Los dos últimos no limitan el número o la longitud de los parámetros que se quieren pasar.

Relación de sistema de archivos, entrada/salida y buffer caché.

Pasos para ejecutar la system call read: int cuenta = read (fd, bufer, nbytes);

1. El programa mete los parámetros en la stack (primero nbytes,
2. luego bufer,
3. luego fd).
4. Se llama al procedimiento de biblioteca read.
5. El procedimiento read coloca el número de la system call read en un registro, para que lo lea el sistema operativo.
6. El procedimiento read realiza un trap para cambiar al modo kernel, saltando a una dirección fija.
7. El despachador busca en una tabla de handlers al handler que maneja esta llamada al sistema.
8. El manejador de llamadas al sistema maneja la llamada.
9. El manejador de llamadas al sistema le regresa el control al procedimiento de biblioteca read. Vuelve a modo usuario del procesador, el sistema operativo sigue en modo kernel.
10. El procedimiento de biblioteca le regresa el control al programa. El sistema operativo sale de modo kernel.
11. El programa limpia la pila, incrementando el stack pointer

Unidad 2 – Procesos

PROCESOS - DEFINICIÓN

Un proceso es un programa en ejecución. Distinguimos procesos de programas, ya que estos últimos pueden existir en memoria secundaria, pero se transforman en procesos cuando se les asigna memoria RAM y están ejecutándose. Por eso, decimos que los programas son estáticos y pasivos, mientras que los procesos son dinámicos y activos.

Los procesos, a diferencia de los programas, tiene un *program counter* (PC) y existen desde que se los dispara hasta que finalizan.

Los procesos poseen tres elementos clave: en primer lugar, tiene CÓDIGO, que incluye toda la sección de código maquina ejecutable; poseen DATOS, que incluye las variables, el espacio de trabajo, etc. Por último, poseen STACKS que contiene datos tempranos como parámetros de subrutinas, variables temporales, y direcciones de retorno. Por lo general un proceso posee dos Stacks, una para el modo usuario y una para el modo kernel. Estos dos stacks son independientes y no comparten un mismo espacio en memoria

El elemento más importante de los procesos es el *Process Control Block*, o PCB. Si bien esta estructura asociada al proceso no forma parte del espacio de direcciones del mismo, contiene toda la información relativa a su proceso de modo que el sistema operativo pueda gestionarlo y el procesador pueda ejecutarlo correctamente. El PCB tiene la capacidad de salvar el contexto del proceso actual, de modo de interrumpir su ejecución para más tarde retomarla, como si nunca hubiera sido interrumpido. El PCB reside en la memoria RAM aun cuando el proceso no está listo para ser ejecutado en ella. El PCB es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina. Esta estructura permite al SO dar soporte a múltiples procesos e implementar un ambiente de multiprogramación.

La información contenida en el PCB es clasificable en 3 categorías:

a) IDENTIFICACION DEL PROCESO: contiene el ID del proceso (PID), identificación del proceso padre e identificación del usuario o grupo de usuario que lo disparó.
b) INFORMACION DEL ESTADO: contiene toda la información clave y necesaria para que el procesador pueda ejecutar el proceso. Posee el *program counter*, flags, información del estado del proceso, el valor de los registros de la CPU, y el PSW (*program status word*)
c) INFORMACIÓN DE CONTROL DEL PROCESO: posee la información de la planificación del CPU, prioridad de los procesos, información de la gestión de

memoria, información contable del uso del CPU, información de las E/S realizadas, lista de archivos abiertos y dispositivos, etc.

PROCESOS – MODO DE EJECUCIÓN

Los procesos, utilizan dos modos de ejecución: En primer lugar, el modo USUARIO. En este modo, no se permite la ejecución de instrucciones privilegiadas. El intento de ejecutar una instrucción privilegiada, produce una excepción del sistema operativo. En este modo, tampoco se puede acceder a determinadas posiciones de memoria RAM.
En segundo lugar, el modo KERNEL. Este modo permite la ejecución de instrucciones privilegiadas, como llamadas al sistema, interrupciones, asignación de espacio de memoria a los procesos, etc. En este modo se puede acceder al espacio de direcciones de otros procesos.

La razón por la que se usan estos dos modos, se debe a que es necesario proteger al sistema operativo y a las estructuras del sistema, como la PCB, el vector de interrupciones, la interferencia entre programas de usuario y administrador, etc.

El sistema operativo conoce el modo de ejecución del proceso, ya que esto se indica con un bit en el PSW, dentro del PCB.

PROCESOS- CAMBIOS DE CONTEXTO Y MODO

Cada vez que se produce una interrupción o una llamada al sistema, se debe guardar el estado del proceso que se estaba ejecutando para luego reanudar su ejecución. Se trata la interrupción o la llamada al sistema y luego se carga nuevamente el PCB del proceso despachado previamente. Este PCB puede también ser el de un nuevo proceso.

El proceso de context switching consta de los siguientes pasos:

1. Se almacena el contexto del procesador, incluyendo el PC y otros registros.
2. Se el PCB que está actualmente en el estado running. Esto implica cambiar el estado del proceso a alguno de los otros estados (ready, blocked, etc.).
3. Se mueve el PCB de este proceso a la cola apropiada (ready, blocked, etc.).
4. Se selecciona otro proceso para ejecución (esto implica llamar al scheduler).
5. Se actualiza el PCB del proceso seleccionado. Esto implica cambiar el estado de este proceso a running.
6. Se actualizan estructuras de datos para administración de la memoria.

7. Se restaura el contexto del procesador que existía en el momento en que el proceso seleccionado fue sacado del estado running por última vez. Esto implica cargar los valores del PC y otros registros.

Por lo tanto, un cambio de proceso implica inevitablemente un cambio de contexto. (para pasar de un proceso a otro, necesito guardar los datos de la PCB del primero). Por el contrario, un cambio de contexto no siempre implica un cambio de proceso, ya que puede volverse a cargar el proceso que se estaba ejecutando previamente.

Un cambio de contexto no siempre involucra un cambio de modo, ya que puedo alternar entre dos procesos de modo usuario. Además, un cambio de modo no siempre implica un cambio de contexto, este es el caso típico de un proceso que llama a una SystemCall dentro del mismo proceso, cambia el modo (de usuario a kernel) pero el contexto el siempre el del proceso original. Este enfoque es muy útil, ya que evita la penalización de 2 cambios de proceso. (Sacar el proceso A, cargar la rutina, sacar la rutina, reanudar el proceso).

PROCESOS - EVENTOS E INTERRUPCIONES

El sistema operativo puede intervenir en el funcionamiento normal de los procesos, cuando se genera algún evento de las siguientes categorías.

a) Interrupciones de HW externas: Son originadas por algún suceso externo al proceso que se estaba corriendo actualmente. Son independientes de este y pueden ser interrupciones de E/S, interrupciones de reloj, etc.

b) Interrupciones de HW internas: Son causadas por una condición de error generada dentro del proceso que se está ejecutando, como por ejemplo un desbordamiento de memoria, el acceso a una dirección ilegal de memoria, una división por cero, fallos de página, etc. Son detectadas como *traps* o excepciones con ayuda del hardware.

c)Interrupciones de Software: Son las llamadas al sistema, y son generadas por el proceso en ejecución. Se generan cuando un proceso necesita un servicio del S.O

PROCESOS – CREACIÓN DE PROCESOS

1) Asignar un único identificador al nuevo proceso. En ese momento se añade una nueva entrada a la tabla principal de procesos, que contiene una entrada por procesos. (PID).
2) Asignar espacio para el proceso. Esto incluye todos los elementos de la imagen del proceso. Así pues, el SO debe saber cuánto espacio se necesitará para el espacio privado de direcciones del usuario (programas y datos) y para la pila del usuario. Si un proceso es generado por otro, el proceso padre puede pasarle al SO los valores necesarios como parte de la solicitud de creación del proceso. Por último, se debe asignar espacio para el bloque de control del proceso.
3) Debe inicializarse el bloque de control del proceso. A parte de identificación del proceso contiene el ID de este proceso junto a otros ID apropiados, tales como el del proceso padre. La parte de información del estado del procesador normalmente se inicializa con la mayor parte de las entradas a cero, excepto para el contador de programa (que se prepara con el punto de entrada del programa) y los punteros a las pilas del sistema (que establecen los límites de la pila del proceso).
4) Se deben establecer los enlaces apropiados. Por ej., si el SO mantiene cada cola de planificación como una lista enlazada, entonces el proceso nuevo se debe poner en la cola de Listos o de Listos/suspendidos.
5) Puede haber otras estructuras de datos que crear o ampliar. Por ej., el SO puede mantener un archivo de compatibilidad para cada proceso que sea utilizado más tarde con propósitos de facturación y/o evaluación del rendimiento.

PROCESOS – PLANIFICACIÓN Y MÓDULOS

Casi todos los recursos de la CPU se planifican antes de utilizarlos. La CPU, como elemento principal de la PC, necesita ser planificada para un diseño optimo del sistema operativo.

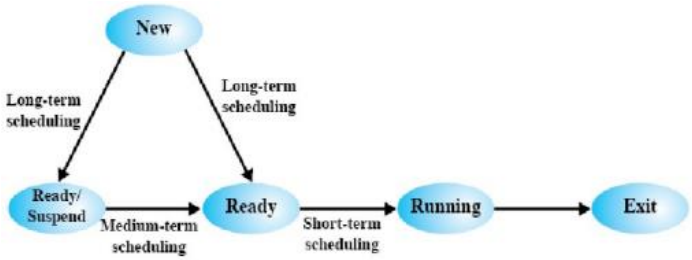
La clave para la multiprogramación está en la gestión de los procesos. El sistema debe tener un balance entre los procesos dedicados a las entradas/salidas (procesos I/O Bound) y los procesos dedicados a la utilización del CPU (CPU Bound). Para ello el sistema cuenta con planificadores (schedulers), que son pequeños módulos del sistema operativo y que se dividen en tres categorías.

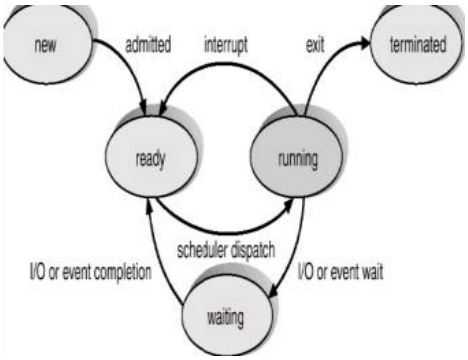
1) Planificador de largo plazo, o *long term scheduler*, es el encargado de determinar el grado de multiprogramación. Es quien regula la cantidad de procesos que entran a memoria principal. A su vez es el encargado de equilibrar los CPU Bound y los I/O bound.
2) Planificador de mediano plazo, o *medium term scheduler,* es el encargado de regular el grado de multiprogramación, reduciendo el mismo mediante el SWAP, o el desalojo de procesos a el área de intercambio de la memoria secundaria.
3) Planificador de corto plazo, o *short term scheduler*, es el encargado de controlar el grado de multiprocesamiento, osea la cantidad de procesos que se están ejecutando a la vez. También es el encargado de decidir qué proceso se va a ejecutar en un momento determinado.

Aparte de estos módulos, existen otras funciones que pueden o no ser llevadas a cabo por alguno de los módulos recién mencionados. Estas son el *Dispatcher* y el *loader*.

El Loader es el encargado de cargar en memoria el proceso elegido por el planificador a largo plazo y el Dispatcher es quien hace el cambio de contexto, cambia de modo de ejecución y despacha el proceso elegido por el planificador de corto plazo.

ESTADOS Y SCHEDULERS





Estados:

- NUEVO: 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
- LISTO: Luego que el scheduller 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.

- EJECUCIÓN: El scheduler de corto plazo lo eligió para asignarle CPU, tendrá la CPU hasta que se termine el período de tiempo asignado, termine o hasta que necesite realizar alguna operación de E/S.
- ESPERA: 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 de 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.
- Running-Ready: Cuando el proceso termina su quantum (franja de tiempo) sin haber necesitado ser interrumpirlo por un evento, pasa al estado de ready, para competir por CPU, pues no está esperando por ningún evento.

PROCESOS – TIPOS DE ALGORITMOS

Dentro de los procesos, nos encontramos con dos tipos de decisión. Por un lado, podemos tener algoritmos de tipo Apropiativo, en los cuales un proceso ejecutándose puede ser interrumpido y pasado a la cola de listos por el S.O. La decisión de expulsar puede ser tomada cuando llega un nuevo proceso, cuando un proceso pasa de estar bloqueado a listo, o basándose en una interrupción de reloj (finaliza el quantum). Este método evita que un proceso monopolice el uso del CPU.

Por otro lado, los algoritmos no Apropiativos, proponen que un proceso que está ejecutándose lo hace hasta que termine por si mismo o hasta que se bloquee por una entrada o salida.

Estos tipos de algoritmos tienen sentido en entornos determinados. Por ejemplo, si usamos procesos batch (no hay interacción con el usuario. Este último no espera una respuesta en un terminal) Se pueden utilizar algoritmos no apropiativos. En cambio, si utilizamos procesos interactivos, donde un usuario espera una respuesta, es conveniente usar algoritmos apropiativos, osea que el proceso pueda ser expulsado del CPU y no sea acaparado por el proceso.

PROCESOS – ALGORITMOS D E PLANIFICACION

1) PRIORIDADES: A cada proceso se le asigna una prioridad y el planificador siempre elegirá un proceso de prioridad mayor sobre un proceso de prioridad menor. Cuando se va a realizar una elección en la planificación, el planificador comenzará a en la cola de listas con la prioridad más alta
2) FCFS (First-Come-First-Served): Es el más simple y no es Apropiativo. La CPU se asigna a los procesos en el orden en que la solicitan. Hay una sola cola de procesos listos. Cuando el primero proceso entra al estado de inicio y se le permite ejecutarse todo el tiempo que desee. O se interrumpe debido a que se ha ejecutado demasiado tiempo. A medida que van entrando otros trabajos y si un proceso bloqueado pasa al estado listo, se colocan al final de la cola. No produce inanición.
3) ROUND ROBIN: Una forma de reducir el castigo que tienen los trabajos cortos con FCFS es la utilización de expulsión basándose en el reloj. Se genera una interrupción de reloj cada cierto intervalo de tiempo. Cuando sucede la interrupción, el proceso actual en ejecución se sitúa al final de la cola de listos y se selecciona el siguiente según la política de FCFS, produciéndose un cambio de contexto que no significa necesariamente un cambio de modo de ejecución. A cada proceso se le da un quantum de tiempo y luego es expulsado.
4) SJF (Shortest Job First): Es un algoritmo de planificación no Apropiativo, donde se requiere que los tiempos de ejecución de los trabajos se conocen de antemano. Cuando hay varios trabajos esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero. Un riesgo con este algoritmo es la posibilidad de inanición de los procesos más largos, si hay una llegada constante de procesos más cortos.
5) SRTF (Shortest Remaining Time First): Es una versión de SJF (es un algoritmo de planificación expulsivo), el planificador escoge el proceso que tiene el menos tiempo de proceso restante esperando. Cuando un nuevo proceso se una a la cola de listos, podría tener un tiempo restante menor que el proceso actualmente en ejecución, por lo tanto, el planificador podría expulsar al proceso actual cuando llega un nuevo proceso.

Unidad 3 – Memoria

GESTION DE MEMORIA – DEFINICION

En un sistema mono programado, la memoria principal se divide en 2 partes: una para el sistema operativo (normalmente en posiciones bajas de la memoria) y otra para el programa que se ejecuta en ese instante (normalmente en posiciones altas de la memoria).

En cambio, en un sistema multiprogramado, la parte de usuario de la memoria debe subdividirse aún más para hacer lugar a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el SO y se conoce como gestión de memoria.

En un sistema multiprogramado es vital repartir eficientemente la memoria para poder introducir tantos procesos como sea posible y que el procesador este la mayor parte del tiempo ocupado.

Se proponen 5 requisitos para la gestión de memoria:

a) REUBICACIÓN: El sistema busca cargar y descargar los procesos activos en la memoria principal para maximizar el uso del procesador, manteniendo una gran reserva de procesos listos para ejecutarse. Una vez que un programa se descargó al disco, cuando vuelve a ser cargado puede necesitar reubicarse en un área distinta de memoria. El S.O debe conocer la ubicación del PCB, de la pila, y la imagen del proceso. El procesador y el S.O deben ocuparse de las referencias a memoria dentro del programa, por ocuparse se entiende traducir las direcciones lógicas del programa a las direcciones físicas de la memoria donde están los datos.
b) PROTECCIÓN: El código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos, sin permiso. Al desconocerse la ubicación de un programa en memoria principal, es imposible comprobar las direcciones absolutas durante la compilación para asegurar la protección. Por lo tanto, todas las referencias a memoria generadas por un proceso deben comprobarse durante la ejecución para asegurar que las instrucciones solo hagan referencia al espacio de memoria destinado a dicho proceso. El procesador es el que debe satisfacer las exigencias de protección de memoria, ya que el SO no puede anticiparse a todas las referencias de memoria que hará el programa
c) COMPARTIMIENTO: La protección debe permitir el acceso de varios procesos a la misma zona de la memoria principal. Los mecanismos para respaldar reubicación forman parte básica de las capacidades de compartimiento.

d) ORGANIZACIÓN LOGICA: Si el SO y el hardware pueden tratar a los programas de usuario y los datos en forma de módulos, se conseguirá una serie de ventajas. La herramienta que satisface esta necesidad es la segmentación.
e) ORGANIZACIÓN FISICA: La memoria secundaria puede permitir un almacenamiento a largo plazo de programas y datos, al tiempo que una memoria principal mantiene los programas y datos de uso actual. En este esquema de 2 niveles, la organización del flujo de información entre la memoria principal y la memoria secundaria debe ser responsabilidad del sistema. La responsabilidad de este flujo NO puede ser asignada al programador.

MEMORIA – TECNICAS DE GESTION DE LA MEMORIA

a) Única partición: Para gestionar la memoria RAM mediante una partición, solo se necesita un registro límite y un registro de realocación
b) Particiones estáticas de igual tamaño: El esquema más simple para gestionar la memoria disponible es repartirla en regiones con límites fijos. Una posibilidad es tener particiones del mismo tamaño, en este caso cualquier proceso cuyo tamaño sea menor o igual que el tamaño de la partición puede cargarse en cualquier partición disponible. Si todas las particiones están llenas y no hay ningún proceso en estado libre o ejecutado, el sistema operativo puede mandar a swap a un proceso de cualquiera de las particiones y cargar otro proceso (división del planificador). Hay dos dificultades con particiones fijas del mismo tamaño: Un programa podría ser demasiado grande para entrar en una partición o bien un programa pequeño ocupa una partición entera desperdiciando espacio.
c) Particiones estáticas de distinto tamaño: Soluciona el tema del inciso anterior en cuanto a seleccionar una porción de memoria adecuada, pero supone algoritmos de búsqueda más complejos. El objetivo siempre es agregar cada proceso a la partición más pequeña dentro de la cual entra, se necesita una cola de planificación para cada partición. Se minimiza la fragmentación interna.
d) Particiones dinámicas: Las particiones se crean dinámicamente, de forma que cada proceso se carga en una partición de exactamente el mismo tamaño que el proceso. Es decir, se le asigna tanta memoria como necesita y no más. Este método comienza bien, pero finalmente, desemboca en una situación en la que hay un gran número de huecos pequeños de memoria. Conforme pasa el tiempo, la memoria comienza a estar más fragmentada y su rendimiento decae. Se produce fragmentación externa. Una técnica para vencer a la fragmentación externa, es la compactación. De vez en cuando, el sistema operativo desplaza los

procesos para que estén contiguos, de forma que toda la memoria libre quede juta en un bloque. El problema es que es un procedimiento que consume tiempo, por lo que desperdicia tiempo del procesador.

MEMORIA – ALGORITMOS DE UBICACIÓN

1)Mejor ajuste: toma el bloque más cercano en tamaño. Es el peor, la memoria principal se quedará rápidamente con bloques demasiados pequeños.

2) Primer ajuste: comienza a analizar la memoria desde el principio y toma el primer bloque disponible que sea suficientemente grande. Más sencillo, mejor y más rápido. Se requiere más frecuentemente la compactación.

3) Siguiente ajuste: comienza a analizar la memoria desde la última ubicación y elige el siguiente bloque que sea lo suficientemente grande.

4) Peor ajuste: toma siempre el hueco más grande disponible.

MEMORIA – REUBICACION

Un proceso puede ocupar diferentes particiones de memoria a lo largo de su vida. De esta manera las ubicaciones (de las instrucciones y datos) referenciadas por un proceso no son fijas, sino que cambian cada vez que se intercambia o desplaza un proceso. Para resolver este problema, se realiza una distinción entre varios tipos de direcciones.

Una dirección lógica es una referencia a una ubicación de memoria independiente de la organización actual de datos a la memoria. Se debe llevar a cabo una traducción a una dirección física antes de que se alcance el archivo a memoria.

Una dirección relativa es un ejemplo particular de dirección lógica, en el que la dirección se expresa como una ubicación relativa a algún punto conocido, normalmente un valor en un registro del procesador.

Una dirección física es una ubicación real de la memoria principal.

MEMORIA – PAGINACIÓN

Mediante la paginación simple, la memoria principal se encuentra dividida en trozos iguales de tamaño fijo, denominados marcos. A su vez, cada proceso está divido también en trozos de tamaño fijo y del mismo tamaño que los de memoria, denominados páginas.

El espacio de memoria malgastado por cada proceso debido a la fragmentación interna corresponde a una fracción de la última página. No hay fragmentación externa. El sistema operativo mantiene una lista de marcos libres.

Al usar el concepto de dirección lógica se permite que un proceso se cargue en marcos contiguos como no contiguos.

El sistema operativo mantiene una tabla de páginas por cada proceso. La tabla de páginas muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página.

La traducción de direcciones lógicas a físicas las realiza el HW. Dada una dirección lógica (número de página, desplazamiento), el procesador usa la tabla de páginas para producir una dirección física (número de marco, desplazamiento).

Una tabla de páginas contiene una entrada por cada página del proceso. Se indexa por número de página. Cada entrada en la tabla de páginas tiene el número del marco en la memoria principal, si existe, que contiene la página correspondiente.

MEMORIA – SEGMENTACIÓN

El programa y sus datos asociados se dividen en un número de segmentos.

NO se requiere que todos los programas sean de la misma longitud, pero hay una longitud máxima de segmento. Una dirección lógica está formada por un número de segmento y un desplazamiento. Se necesita que todos los segmentos de un programa se carguen en la memoria para su ejecución.

Un programa puede ocupar más de una partición y estas particiones no necesitan ser continuas. NO hay fragmentación interna, pero si externa. La segmentación es visible al programador y se proporciona como una utilidad para agregar programas y datos.

Se tiene una tabla de segmentos por cada proceso y una lista de bloques libres de memoria principal. Cada entrada de la tabla de segmentos proporciona la dirección inicial

de la memoria principal del correspondiente segmento. Además de la longitud del segmento.

Cuando un proceso pasa al estado ejecutando, la dirección de la tabla de segmentos se carga en un registro especial usado por el HW de gestión de memoria.

MEMORIA – MEMORIA VIRTUAL

No es necesario que todas las páginas o todos los segmentos de un proceso se encuentren en la memoria principal durante la ejecución.

Supongamos que se tiene que traer un nuevo proceso de memoria. El sistema operativo comienza trayendo únicamente una o dos porciones, que incluye la porción inicial del programa y la porción inicial de datos sobre la cual acceden las primeras instrucciones acceden. Esta parte del proceso que se encuentra realmente en la memoria principal para, cualquier instante de tiempo, se denomina conjunto residente del proceso. Cuando el proceso está ejecutándose, las cosas irán perfectamente mientras que todas las referencias a la memoria se encuentren dentro del conjunto residente. Usando una tabla de segmentos o páginas, el procesador siempre es capaz de determinar si esto es así o no. Si el procesador encuentra una dirección lógica que no se encuentra en la memoria principal, generará una interrupción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en un estado de bloqueado y toma el control.

Con esta estrategia de no cargar el proceso completo en memoria, se conduce a una mejora en la utilización del sistema ya que, pueden mantenerse una mayor cantidad de procesos en la memoria principal, y dado que un proceso puede ser incluso mayor que toda la memoria principal, este método ayuda a que pueda gestionarse mediante su conjunto residente.

Memoria virtual permite una multiprogramación muy efectiva que libera al usuario de las restricciones excesivamente fuertes de la memoria principal.

MEMORIA- FALLOS DE PÁGINA

Si el procesador encuentra una dirección lógica que no se encuentra en la memoria principal, generará una interrupción indicando un fallo de acceso a la memoria.

El HW detecta la ubicación y genera un trap en el sistema operativo. El sistema operativo coloca al proceso interrumpido en el estado bloqueado y toma el control. Para que la

ejecución de este proceso pueda realizarse más adelante, el sistema operativo necesita traer a la memoria principal la porción del proceso que contiene la dirección lógica que ha causado el fallo.

El SO realiza una petición de E/S, una lectura de disco. Después de realizar la petición de E/S, el SO puede activar otro proceso para que se ejecute mientras el disco realiza la operación de E/S. Una vez que la porción solicitada se ha traído a la memoria principal, una nueva interrupción ocurre, dando el control de nuevo al SO, que coloca al proceso afectado de nuevo en el estado listo, además de actualizar la tabla de páginas del proceso.

MEMORIA – PAGINACION CON MEMORIA VIRTUAL

En este caso la entrada en la tabla de páginas es más compleja debido a que solo algunas de las páginas del proceso se encuentran en la MP, se necesita que cada entrada de la tabla de páginas indique si la página está presente (P) en memoria, la entrada también debe indicar el número de marco de dicha página.

La entrada en la tabla de páginas incluye un bit de modificado (M), que indica si los contenidos de la página han sido alterados desde que la página se cargó por última vez en la MP. Si no hay algún cambio, no es necesario escribir la página cuando llegue el momento de reemplazarla por otra página en el marco de la página que ocupa.

La tabla de páginas debe encontrarse en la MP para poder ser accedida. Cuando un proceso se encuentra ejecutando, un registro tiene la dirección de comienzo de la tabla de páginas de dicho proceso. El número de página de la dirección virtual es una para indexar esa tabla y buscar el marco de la página correspondiente. Con el desplazamiento de la dirección virtual se genera la dirección real. La tabla de páginas está expuesta a paginación igual que cualquier otra página. Cuando un proceso está en ejecución su tabla de páginas debe encontrarse en MP.

MEMORIA – BUFFER DE TRADUCCIÓN ANTICIPADA (TLB)

Toda referencia a memoria virtual puede causar dos accesos a memoria física: una para buscar la entrada de la tabla de páginas y otra para buscar los datos. Esto duplica el tiempo de acceso a memoria. Se usa una caché especial para las entradas de la tabla de páginas, llamada buffer de traducción anticipada. Este caché tiene las entradas de la tabla de páginas que han sido usadas de forma más reciente.

Dada una dirección virtual, el procesador primero examina la TLB, si la entrada de la tabla de páginas está presente (TLB hit), entonces primero se recupera el número de marco y se construye la dirección real para acceder a la página. Si la entrada de la tabla de páginas no se encuentra (TLB miss), el procesador usa el número de página para indexar la tabla de páginas del proceso y examina la entrada. Si el bit de presente está puesto en 1, entonces la página se encuentra en MP, y el procesador puede recuperar el número de marco desde la entrada de la tabla de páginas para construir la dirección real. Si el bit de presente está en 0, entonces la página solicitada no se encuentra en la MP y se produce un fallo de acceso a memoria (fallo de página). El SO corrigió la página necesaria y actualizó la tabla de páginas.

MEMORIA – SEGMENTACIÓN CON MEMORIA VIRTUAL

El programador usa la memoria como diferentes espacios de direcciones o segmentos, los segmentos pueden ser de diferentes tamaños. Una referencia a la memoria es del tipo (memoria de segmento, desplazamiento)

Las ventajas para el programador son que se simplifica el tratamiento de estructuras de datos que pueden crecer, permite que programas se modifiquen o recopilen información de manera independiente. Da soporte a la compactación de procesos, y soporta mecanismos de protección. Hay una tabla de segmentos por cada uno de los procesos. En este caso las entradas en la tabla de segmento son más complejas.

Solo algunos de los segmentos del proceso pueden encontrarse en la MP, entonces se necesita un bit en cada entrada de la tabla de segmentación para indicar si el segmento se encuentra presente en la MP o no. Si el segmento está en memoria, la entrada debe indicar la dirección de comienzo y la longitud del segmento.

MEMORIA – PAGINACION Y SEGMENTACION COMBINADAS (segmentación paginada)

El espacio de direcciones se divide en un número de segmentos. Cada segmento se divide en un número de páginas de tamaño fijo, que son del tamaño de los marcos de la MP. Desde el punto de vista del programador, una dirección lógica contiene un número de segmento y un desplazamiento dentro del segmento. Desde el punto de vista del sistema, el desplazamiento dentro del segmento es visto como un número de página y el desplazamiento dentro de la página incluida en el segmento.

Para cada proceso existe una tabla de segmentos y varias tablas de páginas (una para cada segmento). Cuando un proceso está en ejecución, un registro mantiene la dirección de comienzo de la tabla de segmentos del proceso. El procesador usa la parte del número de segmento para indexar la tabla de segmentos para encontrar la tabla de páginas de dicho segmento. Después, el número de página de la dirección virtual se usa para indexar la tabla de páginas y buscar el número de marco, se combina con el desplazamiento de la dirección virtual para obtener la dirección real. El HW se encarga de la legalidad de una dirección.

Estructura:

- tabla de segmentos: una para cada proceso.
- tabla de páginas: una por segmento.
- tabla de bloques de memoria: para controlar asignación de páginas por parte del sistema operativo.

Ventajas:

- Facilidad de implantar la compartición y enlace.
- Se simplifican las estrategias de almacenamiento.
- Se elimina el problema de la fragmentación externa y la necesidad de compactación.

Desventajas:

- Los tres componentes de la dirección y el proceso de formación de direcciones hace que se incremente el costo de su implantación.
- 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.

MEMORIA – POLITICAS DE RECUPERACION

Paginación bajo demanda:

Una página se trae a memoria solo cuando se hace referencia a una porción en dicha página. Cuando un proceso se inicia va a haber una ráfaga de fallos de página. Ya que el paginador solo busca las páginas que se necesitan, debemos agregar un bit en la tabla de páginas 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 hardware para apoyar la paginación bajo demanda es el mismo que se usa para la paginación y segmentación, y los intercambios:

- Tabla de páginas: Esta tabla tiene la capacidad para marcar una entrada como inválida usando un bit válido-inválido o un valor especial de los bits de protección.
- Memoria secundaria: Esta memoria contiene las páginas que no se conservan en la memoria principal.

Estructuras:

- Tabla de páginas: una por cada proceso, una entrada por cada página.
- Descriptores de bloques de disco: Cada entrada está asociada a una página.
- Tabla de marcos: Describe cada marco de la memoria real y está indexada por número de marco.
- Tabla de uso de SWAP: una tabla por cada dispositivo de intercambio con una entrada por cada página.

Paginación adelantada: se traen a memoria también otras páginas, diferentes de las que ha causado el fallo de página. Es ineficiente si las páginas traídas no se usan luego.

MEMORIA – ASIGNACION DE MARCOS EN MEMORIA

El conjunto residente es la parte del proceso que se encuentra en la M para cualquier instante de tiempo. El tamaño del conjunto residente supone las siguientes consecuencias: Cuanto menor es la cantidad de memoria reservada para un proceso, mayor es el número de procesos que pueden residir en la MP. Si el conjunto de páginas de un proceso que está en memoria es pequeño, la probabilidad de un fallo de página es mayor. Más allá de un cierto tamaño no tendrá efecto sobre la tarea de fallos de página de un proceso.

La política de **asignación fija** proporciona un número fijo de marcos de MP disponibles para ejecución. Este número se decide cuando se usa el proceso y se determina en base al tipo de proceso. Cuando se produzca un fallo de página, la página que se necesita reemplazar será una de las páginas del mismo proceso.

La política de **asignación variable** permite que se reserven un número de marcos por proceso que puede variar a lo largo del tiempo de vida, en función de la tarea de fallos de página.

El reemplazo local supone que el número de marcos asociados a un proceso es fijo. Las páginas que se van a reemplazar se eligen entre los marcos asignados al proceso.

El reemplazo global, supone que las páginas que se van a reemplazar se eligen entre todos los marcos de la memoria principal. Esto hace que el tamaño del conjunto residente del proceso varíe.

MEMORIA – HIPERPAGINACIÓN

Decimos que el sistema está en trashing cuando pasa más tiempo paginando que ejecutando procesos. Como consecuencia, hay una importante baja de performance en el sistema.

Ciclo de thrashing

1) El SO 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: comienzan los fallos de página y robo de marcos a otros procesos.
5) Por swapping de páginas bajo el uso de la CPU.
6) Vuelvo al 1)

El scheduler de CPU y el thrashing

1) Cuando se disminuye el uso de la CPU, el scheduler de long term aumenta el grado de multiprogramación.
2) El nuevo proceso inicia nuevos fallos de páginas y por lo tanto más actividad de paginado.
3) Se disminuye el uso de la CPU.
4) Vuelvo a 1)

Se puede limitar el trashing usando algoritmos de reemplazo local. De esta forma, si un proceso entra en trashing no usa marcos a otros procesos (perjudica la performance del sistema). Si un proceso cuenta con todos los frames que necesita no habrá trashing.

TECNICAS PARA EVITAR THRASHING:

*MODELO DE LOCALIDAD: Se refiere a la cercanía de referencias. 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.

Un programa tiene varias localidades (ej cada rutina). Para prevenir la hiperactividad, el proceso debe tener en memoria sus páginas más activas.

*MODELO DEL WORKING SET: Este modelo está basado en el modelo de localidad. Existe una ventana del working set o conjunto residente, que son las referencias de memoria más recientes. El working set es el conjunto de paginas que tienen las más recientes referencias a páginas. Si la ventana del working set es muy chica, no cubrirá la localidad. Si es muy grande, puede tomar varias localidades. Prevención de thrashing por working set

- El SO monitorea cada proceso, dándole tantos frames hasta ser wssi.
- Si quedan frames, puede iniciar otro proceso.
- Si D excede m se elige un proceso para suspender.
- De esta forma se mantiene alto el grado de multiprogramación optimizando el uso de la CPU.

*FRECUENCIA DE FALLO DE PAGINA: Cuando la frecuencia de fallos de página es baja, los procesos tienen muchos marcos originados. Con este modelo se establecen límites superiores e inferiores para la tasa de fallos de páginas. Si la tasa de fallos de páginas excede el límite inferior, se le asigna otro marco a ese proceso. Si la tasa está por debajo del límite inferior, se le quita un marco al proceso. De esta forma se puede medir y controlar la memoria De fallos de páginas para evitar la hiperpaginación. Puede llegar a suspender un proceso su no hay más marcos. Los marcos se reasignan a procesos de alto PFF.

Unidad 4 – Entrada/Salida

Un SO controla todos los dispositivos de E/S de la computadora. Debe emitir comandos para los dispositivos, captar interrupciones y manejar errores. Deben dar una interfaz entre los dispositivos y el resto del sistema.

HW y SW involucrado: buses, controladores, dispositivos, puertos de E/S, registros, comunicación con controlador de dispositivo: E/S programada, interfaces.

Hay 3 técnicas para realizar la E/S:

1. E/S programada: el procesador emite una orden de E/S de parte de un proceso a un módulo de E/S; el proceso espera hasta que se complete la operación antes de continuar (espera activa, se queda preguntando al

dispositivo de E/S si ya está listo para realizar la E/S - Bit de ocupado/desocupado de la controladora).

2. E/S dirigida por interrupción: el procesador emite una orden de E/S de parte de un proceso a un módulo de E/S, el procesador continúa la ejecución de las instrucciones siguientes mientras el modulo E/S se prepara para la E/S. Cuando está listo para realizar la operación de E/S manda una interrupción y el procesador la capta (entre medio de cada instrucción verifica la línea de interrupciones). El procesador interrumpe lo que estaba haciendo y comienza la transferencia.
3. Acceso directo a memoria: un módulo de DMA, que es un procesador de propósito especial, controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador envía una petición de transferencia al DMA. El DMA interrumpe al procesador solo para indicarle que la transferencia ya termino.

Los controladores tienen uno o más registros: para señales de control y para datos. La CPU se comunica con los controladores escribiendo y leyendo registros. Al escribir en los registros, el SO puede hacer que el dispositivo envíe u ocupe datos, encienda o apague, u otras acciones. Al leer estos registros, el SO puede conocer el estado del dispositivo, si está preparado o no para ejecutar un comando.

<u>Mapeo de E/S</u>

- Correspondida en memoria (E/S mapeada en memoria)
    - Dispositivos y memoria comparten el espacio de direcciones.
    - La E/S es como escribir y leer en la memoria.
    - No hay instrucciones especiales para I/O (ya se dispone de muchas instrucciones para la memoria)
- E/S aislada (usa de puertos de E/S)
    - Espacios separados de direcciones.
    - Se necesitan líneas de E/S, puertos de E/S.
    - Instrucciones limitadas de E/S.

Planificar un conjunto de solicitudes de E/S significa determinar un orden adecuado en el que ejecutarlas. Mejora el rendimiento global del sistema EJ: planificación de requerimiento a disco para manejar movimientos. El SO mantiene una cola de espera de solicitudes para cada dispositivo. Cuando un proceso ejecuta una llamada al sistema de E/S bloqueante, la solicitud se coloca en la cola correspondiente a dicho dispositivo. El planificador de E/S reordena la cola para mejorar la eficiencia global del sistema.

E/S – BUFFER

Un buffer es una memoria intermedia que almacena datos mientras se están transmitiendo entre 2 dispositivos o entre un dispositivo y una aplicación. Es un amortiguador de velocidades.

Si un proceso realiza una E/S sin buffer, este debe quedar residente en la memoria principal, no pudiendo ser expulsado a disco. Esta condición reduce la oportunidad de usar el intercambio al fijar como residente parte de la memoria principal, lo que conlleva una disminución del rendimiento global del sistema. Asimismo, el dispositivo de E/S queda asociado al proceso durante la duración de la transferencia, no estando disponible mientras tanto para otros procesos.

Por lo tanto, en un entorno multiprogramado, donde hay múltiples operaciones de E/S, el almacenamiento intermedio es una herramienta que puede incrementar la eficiencia del SO y el rendimiento de los procesos individuales.

E/S - CACHE

Un cache es una región de memoria rápida que contiene capas de ciertos datos, entonces se mantiene en memoria cache una capa de los datos de reciente acceso para mejorar la performance.

Dicha memoria cache reduce el tiempo medio de acceso a memoria aprovechándose del principio de la proximidad. La cache de disco contiene una copia de algunos de los sectores del disco. Cuando se hace una petición de E/S solicitando un determinado sector, se comprueba si el sector está en la cache del disco. En caso afirmativo, se sirve la petición desde la cache. Debido al fenómeno de la proximidad de referencias, cuando se lee un bloque de datos en la cache para satisfacer una única petición de E/S, es probable que haya referencias a ese mismo bloque en el futuro.

La diferencia entre cache y buffer en que el buffer puede almacenar la única copia existen de un elemento de datos, mientras que la cache almacena una copia de un elemento que reside en otro lugar.

Protección de E/S

Un proceso de usuario puede intentar accidentalmente o deliberadamente interrumpir la operación normal del SO, tratando de ejecutar instrucciones de E/S ilegales. Para evitar esto se definen las instrucciones de E/S como instrucciones privilegiadas. Además, el SO debe proteger todas las ubicaciones de memoria mapeada y de los puertos de E/S frente a los accesos de los usuarios.

Unidad 5 – Archivos

ARCHIVO - DEFINICION

Un archivo es una colección de información relacionada, con un nombre, que se graba en almacenamiento relacionado. Es la unidad lógica más pequeña de almacenamiento secundario (no pueden escribirse datos en el almacenamiento secundario a menos que estos se encuentren dentro de un archivo).

Los archivos son unidades lógicas de información creados por los procesos. Los procesos pueden leer los archivos existentes y crear otros si es necesario. La información que se almacena en los archivos debe ser persistente, no debe ser afectada por la creación y terminación de los procesos. Un archivo debe desaparecer solo cuando su propietario lo elimina.

Los archivos generalmente poseen los siguientes atributos: nombre, identificador dentro del sistema de archivos, tipo, ubicación, tamaño, protección, fecha, hora, identificación del usuario.

DIRECTORIOS

La información de los archivos se almacena en la estructura de directorios, que también reside en el almacenamiento secundario.

Un sistema de gestión de archivos es aquel conjunto de software del sistema que proporciona servicios a los usuarios y aplicaciones para el uso de archivos. Los objetivos de un sistema de gestión de archivos son:

Almacenar datos y operar con ellos, Protección de los datos, Optimizar la performance del sistema, Minimizar la posibilidad de pérdida de datos, Brindar soporte para distintos

tipos de almacenamiento, Interface personalizada para aplicaciones de usuarios, Garantizar que sea posible la coherencia de los datos

En UNIX, todo es considerado y es tratado como si fuera un archivo. Inclusive los dispositivos y los directorios. El sistema de archivos de UNIX tiene una forma similar a la de una estructura de datos de tipo árbol.

Los directorios son una herramienta que nos ayuda a mantener archivos relacionados de manera conjunta, a su vez nos permite separar archivos no relacionados de otros grupos de archivos. Los subdirectorios del sistema de archivo, que también son directorios, poseen archivos de determinada naturaleza, por ejemplo, dev posee información de los dispositivos, bin posee archivos binarios, etc. Los directorios entonces, nos permiten tener una organización del espacio. Los directorios poseen información de los archivos, tal como ubicación, permisos, nombre, tamaño.

El directorio en el que uno se encuentra en un momento determina, se llama directorio de trabajo, y nos ayuda a nombrar a los archivos allí presentes a través de un path relativo (relativo al directorio actual). Otra forma de llamar a los achivos es mediante su path absoluto, el cual comienza desde el directorio superior root. Que existan estos paths, nos ayuda, entre otras cosas, con el nombramiento de archivos. Esta distinción de paths permite que dos archivos tengan el mismo nombre, si y solo si se encuentran en directorios distintos. El path absoluto SIEMPRE es único para un determinado archivo.

Los directorios implementan un sistema de permisos, mediante el cual determinado archivo solo será accesible para usuarios con permisos o no. Estos permisos pueden ser read, write, o execute, updating, deletion, etc.

Los owners, o propietarios de un determinado archivo, poseen todos los permisos sobre el archivo. A su vez, pueden dar permisos a otros usuarios.

Sistema de archivos UNIX SYSTEM V

El sistema de archivos de UNIX está formado por una tabla de i nodos, o nodos índices. Por cada archivo existente en el sistema, hay una entrada en la tabla de i nodos para ese archivo. Si hay n número de archivos, hay n número de entradas en la tabla.

Cada archivo es representado con un i nodo. La entrada en la tabla de i nodos no contiene el archivo en sí mismo ni su contenido, solo posee el metadato del archivo, osea la información sobre el archivo. El i nodo es una identidad de 64 bits. La información relativa al archivo es la siguiente: de que tipo es el archivo, que permisos de acceso tiene,

quien o quienes son los owners del archivo, la fecha de la última vez que fue accedido, la fecha en que fue creado, fecha de su última modificación, el tamaño del archivo, numero de bloques en memoria utilizados por el archivo, y las direcciones de los bloques de memoria donde el archivo está presente, entre otras.

Tipo de asignación indexada: Cada archivo de Unix tiene asociado un inodo. La lista de las direcciones de los bloques que forman un archivo se ubica en su inodo. Son 13 direcciones: 10 directas y 3 indirectas.

Estructura: El disco duro tiene en primer lugar el MBR o master boot record, luego la tabla de particiones. En una partición, posee el boot block (código para bootear el S.O) el superblock (contiene la información del sistema de archivos) tabla de i nodos: tabla que contiene todas las entradas i nodo. Y los data blocks: bloques de datos de los archivos.

Se especifica la medida de la lista de inodos en el momento de la configuración del filesystem. El kernel referencia inodos a través de un índice en la lista de inodos.

Las entradas de la tabla de i nodos poseen dos campos, numero de i nodo y nombre del archivo. Cuando se abre un archivo, el sistema de archivos debe tomar el nombre del archivo y localizar sus bloques de disco.

Cuando se quiere abrir un archivo, se debe ubicar primero cuál es su inodo. Esto se hace accediendo al directorio al cual pertenece el archivo y en la entrada correspondiente a su nombre, está el número de inodo. Se accede entonces a ese inodo, y de allí se tomarán las direcciones para llegar al contenido. Debemos considerar que el directorio es también un archivo, por lo tanto para acceder a él, deberemos buscar en el directorio padre cuál es su inodo, leerlo y recién ahí puedo ver el número de inodo del archivo que me interesa.

Unidad 6 – Buffer Caché

Una cache de disco es un buffer en MP para almacenar temporalmente bloques del disco. Contiene una copia de algunos sectores del disco. El objetivo es minimizar la frecuencia de acceso al disco.

Cuando se satisface una petición de E/S de la cache del disco, se deben entregar los datos de la cache al proceso solicitante, hay dos alternativas:

- Se copia el bloque de datos almacenada en la MP originada a la cache del disco a la memoria asignada al proceso de usuario (MP).

- Usando la técnica de memoria compartida, pasando el puntero correspondiente al bloque en la cache de disco.

Estrategia de reemplazo (algoritmo de reemplazo)

Cuando se trae un nuevo sector (bloque) a la cache de disco, se debe reemplazar uno de los bloques existentes. El algoritmo más utilizado es el de menos recientemente usado (LRU). Se reemplaza el bloque que ha estado en la cache más tiempo sin ser accedido. La cache consiste en una pila de bloques, estando el bloque más reciente accedido en lo más alto de la pila. Cuando se trae un bloque desde la memoria secundaria, se elimina el bloque que estaba al final de la pila, situándose el nuevo bloque en la cima de la fila. No se mueven los bloques en la memoria, se asignan punteros.