**Memoria**

Hiperpaginación (thrashing): Ocurre cuando el sistema invierte más tiempo (ciclos de reloj) implementando mecanismos de paginación que en la propia ejecución de un proceso.

Provoca grandes problemas de rendimiento e incrementa también el tiempo efectivo de acceso a memoria y no se llega a realizar ningún trabajo útil porque los recursos serán invertidos en mecanismos de paginación.

¿Cómo puede ser tratada?

La forma de evitar la hiperpaginación es mediante la utilización de algoritmos de reemplazo de páginas de manera local, es decir que si un proceso produce un fallo de página no robara frames a otro proceso, sino que se limitara a utilizar sus marcos; reemplazando sus propias páginas para evitar que otros procesos incurran en fallos de página.

Los algoritmos de reemplazo (tanto local como globalmente) compartiendo o no frames, están basados en los principios de localidad.

**Localidad temporal**: Si en un momento una posición de memoria es referenciada es muy probable que la misma posición sea referenciada nuevamente en un futuro cercano.

**Localidad espacial**: Si una posición de memoria es referenciada en un momento concreto es probable que las localidades cercanas a ella también serán referenciadas pronto.

Algoritmos de reemplazo:

FIFO (First In – First Out): Guarda en orden de carga las páginas de modo que reemplaza la que primero fue cargada. Se usa una cola.

Segunda chance: Es una pequeña modificación de FIFO. Cuando se saca una página (se toma la primera de la cola y en vez de sacarla, se consulta el valor de un bit de referencia). Si esta en 1, se pone en 0 y se vuelve a poner en la cola. Si estaba en 0 se saca.

Cada vez que la MMU (unidad de gestión de memoria) accede a una página fija su bit de referencia en 1. Para esto es necesario soporte para bit de referencia por hardware.

LRO: (Menos recientemente usada): La estrategia consiste en llevar al disco la página que ha permanecido por más tiempo sin ser accedida. Es costosa su implementación en cuanto a los recursos. Se trata de una lista enlazada con todas las paginas ordenadas, pero al momento de ser referenciada una página, esta tiene que cambiar de lugar, lo cual involucra un alto costo asociado.

NRU (No recientemente usada): Favorece a las páginas que fueron usadas recientemente. Funciona de la siguiente manera: Cuando una página es referenciada se fija el bit de referencia para esa página del mismo modo que ocurre cuando esta página es modificada (se fija el bit de modificada).

Usualmente estas acciones son realizadas por el hardware, aunque también pueden ser realizadas por software.

Cuando una página debe ser reemplazada, el sistema operativo divide las paginas en 4 categorías:

Categoría 0: No referenciada, no modificada.

Categoría 1: No referenciada, modificada.

Categoría 2: Referenciada, no modificada.

Categoría 3: Referenciada, modificada.

Las mejores páginas a cambiar son las de categoría 0. Las peores son las de categoría 3.

Se desaloja al azar una página de la categoría más baja que no esté vacía.

Optimo: Tiene como finalidad retirar la página que vaya a ser referenciada más tarde, para ello se necesita poder predecir la página lo cual es imposible. Se utiliza este modelo para compararlo con un algoritmo implementable. El que más se acerque al optimo será el mejor.

*Memoria virtual con paginación por demanda. Funcionamiento, estructuras utilizadas, participación del hardware. Fallos de página: explicar cómo se resuelven y que estructuras de las descriptas en el punto anterior son utilizadas (tanto modificadas, creadas, consultadas).*

Paginación por demanda: bajo esta técnica, la memoria es lógicamente dividida en partes de igual tamaño llamadas marcos. El espacio de direcciones del proceso es dividido en partes del mismo tamaño que los marcos llamadas páginas.

El sistema operativo mantiene una tabla de páginas por cada proceso (contiene el marco en el que se encuentra cada página).

Esta técnica utiliza un intercambiador perezoso que se encarga de nunca intercambiar una página a menos que vaya a ser necesaria.

Cuando hay que cargar un proceso, el paginador realiza una estimación de que paginas serán utilizadas antes de cargarlo. En lugar de cargar el proceso completo, solo carga las necesarias reduciendo así el tiempo de carga y la cantidad de memoria física ocupada.

Se necesita algún tipo de hardware para diferenciar las paginas necesarias de las que no lo son, para ello se utilizara un bit de validez.

La tabla de páginas puede marcar una entrada como invalida a través de dicho bit (valid – invalid) y el valor del bit de modificado.

Las tablas son mantenidas por el sistema operativo y utilizadas por el MMU (unidad de gestión de memoria) para realizar las traducciones.

El hardware para soportar paginación por demanda es el mismo que para paginación y swapping.

* Partición de swap (intercambio): Es el espacio que se utiliza para mover una página a memoria secundaria y de esa forma liberar la memoria principal para cargar otros procesos.

Mientras no haga falta, el proceso extraído de memoria puede continuar en disco. Cuando sea necesario, el sistema vuelve a hacer un intercambio, pasándolo de disco a RAM.

* TLB: Cache de ultimas paginas referenciadas. Administrada por la MMU (unidad de gestión de memoria) para reducir lectura en memoria principal.
* MMU (Unidad de gestión de memoria): Se encarga de traducir direcciones virtuales a direcciones reales. Realiza la protección de la memoria y genera una interrupción cuando se hace referencia a una página que no está en memoria.

Mientras el proceso se ejecute y acceda a páginas que se encuentren en memoria, la ejecución se llevara a cabo normalmente.

El intento de acceso a una página que no ha sido cargada en memoria (página marcada como invalida) provoca una interrupción de fallo de página.

El hardware de paginación al, traducir una dirección mediante la tabla de páginas, detectara que el bit de páginas invalida está activado, provocando una interrupción dirigida al sistema operativo.

Esta interrupción se produce como resultado de que el sistema operativo no ha cargado anteriormente en memoria la página deseada.

Procedimiento para tratar fallos de página

1. Se comprueba una tabla interna correspondiente al proceso (que usualmente se encuentra en la PCB), para determinar si la referencia era un acceso de memoria valido o invalido.
2. Si la referencia era invalida: Se termina el proceso. Si era válida: si todavía no había sido cargada en memoria, se carga.
3. Se busca un marco libre (por ejemplo, de la lista de marcos libres).
4. Se ordena una operación de disco para leer la página deseada en el marco recién asignado.
5. Una vez completada la lectura de disco, modificamos la tabla interna que se mantiene con los datos del proceso y la tabla de páginas para indicar que dicha página se encuentre ahora en memoria.
6. Se reinicia la instrucción que fue interrumpida. El proceso podrá acceder a la página como si siempre hubiese estado en memoria.

**Procesos**

*Describa los estados de un proceso. Planificadores, su relación con las transiciones entre los estados enumerados.*

Estados de un proceso:

* Nuevo: El usuario “dispara el proceso”. Un proceso es creado por otro proceso: su proceso padre.

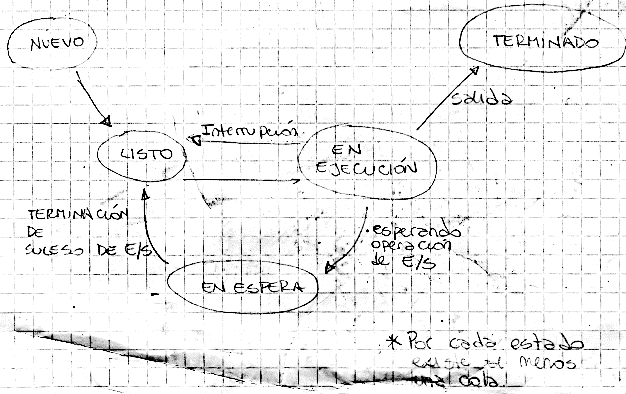
En este estado se crean las estructuras asociadas y el proceso queda en la cola de procesos nuevos, normalmente en espera de ser cargado en memoria.

* Listo o preparado: El scheduler de largo plazo elige el proceso para cargarlo en memoria. El proceso solo necesita que se le asigna CPU. Está en la cola de procesos listos. Es el loader quien lo carga.
* En ejecución: El scheduler de corto plazo lo eligió para asignarle CPU. Tendrá la CPU hasta que termina su quantum o hasta que necesite que se produzca un evento (por ejemplo, una E/S).

Es el dispatcher quien hizo que comience a ejecutarse luego de realizar el context switch.

* En espera: El proceso necesita que se cumpla el evento esperado. para continuar. El evento puede ser una E/S solicitada o una señal de otro proceso.

Sigue en memoria, pero no tiene la CPU. Al cumplirse el evento pasara al estado de listo.



Un proceso tiene como mínimo 3 partes: código, datos y stack (una en modo usuario y una en modo kernel).

La estructura más importante de un proceso es la PCB. Es una estructura que usa el SO para obtener información del proceso.

Es lo primero que se crea y lo último que se borra de un proceso. La PCB se encuentra siempre en memoria y contiene:

* Identificador del proceso (PID) también el PPID (id del padre).
* Estado del proceso.
* Contador del programa.
* Valores de registro de CPU (se utilizan en el context switch).
* Espacio de direcciones de memoria.
* Prioridad (en caso de utilizarse ese algoritmo de planificación de CPU que requiera prioridad).
* Lista de recursos asignados.
* Datos del propietario (usuario).
* Permisos asignados.
* Signals pendientes a ser servidos.
* Grupo al que pertenece (si así fuese) etc.

**Segmentación por demanda**

El termino memoria virtual se refiere a la abstracción de la memoria física en memoria lógica (como es vista por el proceso). La técnica de memoria virtual permite que se ejecuten procesos que no están completamente alojados en memoria.

La ventaja más notable es que los programas pueden ser más grandes que la memoria física.

Esto permite a los programadores despreocuparse de dicha limitación.

La segmentación es una técnica de gestión de memoria que pretende acercarse más al punto de vista del usuario.

Para implementar segmentación por demanda es necesario:

* Partición de intercambio (swap): es el espacio que se utiliza para mover un segmento a memoria secundaria y de esa forma liberar memoria principal para cargar otros procesos. Mientras no haga falta el proceso extraído de memoria puede quedarse en el disco ya que ahí no utiliza memoria física. Cuando sea necesario, el sistema vuelve a hacer un intercambio pasándolo a memoria RAM.
* Tabla de segmentos: se utiliza para traducir las direcciones virtuales en direcciones reales.

Para ello el procesador posee una tabla de segmentos con 4 filas.

Cada fila describe uno de los 4 segmentos del programa en ejecución.

* MMU (Unidad de gestión de memoria): Dispositivo que traduce direcciones virtuales a reales consultando la tabla de segmentos.
* Hardware de segmentación que indique la presencia de un segmento.
* Si se referencia a un segmento en memoria, se accede normalmente.
* Si se referencia a un segmento que no está en memoria, se produce una excepción.

En una organización segmentada, el espacio de direcciones de un proceso se descompone en 4 áreas llamadas segmentos:

* Segmento de código.
* Segmento de datos.
* Segmento de pila.
* Segmento de sistema, invisible para la aplicación, donde reside el núcleo.

El sistema mantiene una lista de segmentos en memoria periódicamente. El sistema coloca segmentos accedidos al final de la lista.

Limpia los bits de acceso y cuando hay que reemplazar un segmento, se reemplaza el o los primeros de la lista.

Se trata de una aproximación a la LRU (menos recientemente usado) porque la lista esta “ordenada” por tiempo de acceso.

El mecanismo de reemplazo es un poco distinto al de paginación por demanda porque los segmentos tienen distintos tamaños.

Al producirse un fallo de segmento:

1. El sistema operativo chequea si hay suficiente espacio en memoria principal, si no hay, se realiza una compactación.
2. Si aún no hay espacio:

Se elige el primer segmento de la lista y si es necesario se escribe en el intercambio.

Si ahora si hay espacio:

Se carga el segmento en memoria, se actualiza la tabla de segmentos y se pone al final de la lista.

Si no hay espacio suficiente: se vuelve al paso 1.

Para llevar un proceso a disco, el núcleo copia bit a bit cada segmento en el área de swap.

Luego libera la memoria que era ocupada por esos segmentos para que se pueda almacenar otros segmentos.

El descriptor del proceso sigue en memoria principal, pero el campo “offset” de la tabla de segmentos del proceso es reemplazado por una dirección que indica la dirección de cada segmento en el área de swap. Además, el proceso pasa a un estado especial que indica que se encuentra en disco y por lo tanto no puede ejecutarse.

Aunque la paginación por demanda es considerada el sistema de memoria más eficiente, se necesita una gran cantidad de hardware para implementarla.

Cuando no se dispone de ese hardware, una opción (aunque menos eficiente) es la segmentación por demanda.

**Sistema de archivos**

Un sistema de archivos permite realizar una abstracción de los dispositivos físicos de almacenamiento para que sean tratados a nivel lógico, como una estructura más sencilla que la estructura de su arquitectura de hardware.

Un sistema de archivos UNIX se caracteriza por:

* Poseer una estructura jerárquica.
* Realizar un tratamiento consistente de los datos de los archivos.
* Poder crear y borrar archivos.
* Permitir un crecimiento dinámico de los archivos.
* Proteger los datos de los archivos.
* Tratar a los dispositivos periféricos como archivos.

El Filesystem UNIX está compuesto por (en disco).

* Bloque de booteo para el sistema operativo.
* Superbloque: Información de cómo se compone el Filesystem (tamaño, espacio libre, etc.).
* Tabla de Inodos: tabla correspondiente a cada uno de los archivos que se encuentran en el Filesystem.
* Bloques de datos: contiene los bloques a los que hace referencia cada i-nodo.

Esta organizado a nivel lógico en forma de árbol invertido con un nodo principal conocido como el nodo raíz (“/”).

Estructura de datos en memoria

Tabla de descriptores: Puntero a tabla de archivos.

Tabla de archivos abiertos por el usuario y el Kernel:

* Es una estructura global del Kernel y en ella hay una entrada por cada archivo distinto que los procesos del Kernel o los procesos del usuario tienen abiertos.
* Punteros a la tabla de i-nodos.
* Cuenta de referencia.
* Modo de apertura.
* Desplazamiento (offset).

Tabla de i-nodos: Es una copia en memoria de la lista que hay en el disco, sumado a la información adicional.

*Analice en el sistema UNIX SystemV la creación de un nuevo archivo (por ejemplo, en el directorio /home/usuario).*

creat ()

1. – Obtener el inodo a partir del nombre (pathname).
2. Si el archivo no existía: Asignar el inodo libre y crear la entrada en el directorio padre (/home/usuario).

Si ya existía:

- Si el acceso no está permitido:

1. Truncar el archivo si queremos permisos de escritura.
2. Liberar todos sus bloques de datos.
3. Asignar una entrada en la tabla de archivos cuenta de referencia = 1.
4. Asignar una entrada en la tabla de descriptores de archivos.

open (apertura de un archivo)

1. Paso previo: Acceso al i-nodo => traer copia a memoria.

Descripción: open (nombre, flags, nodos).

1. El Kernel busca en el sistema de archivos el archivo pasado como parámetro: name i.
2. Si el archivo no existe o no se tienen permisos => Error.
3. Obtiene copia del inodo en memoria.
4. Asigna una entrada en la tabla de archivos:

- Puntero al i-nodo.

- Desplazamiento= “0” o bien, desplazamiento = tamaño del archivo.

1. Si el nodo es igual a O\_TRUNC, entonces liberar los bloques (disco) del archivo y establecer el tamaño de archivo (en i-nodo) a cero.
2. Asignar una entrada en la tabla de descriptores de archivos de usuario.

Sistema de E/S

UNIX Modela a los dispositivos de E/S como si fueran archivos, de esta forma, las operaciones sobre archivos (leer, escribir, buscar, posicionarse) se utilizan sobre los periféricos.

Entonces “imprimir” es escribir en la impresora y mostrar un carácter en pantalla es escribir en ella.

Para cada fichero especial (que representa un dispositivo) hay un driver.

Cada driver tiene un numero de dispositivo principal (mayor number) que sirve para identificarlo. Si el driver sirve a varios dispositivos tiene un numero de dispositivo (minor number) que sirve para identificarlo.

Los drivers pueden invocar procedimientos del Kernel para asignar memoria, administrar temporizadores, controlar DMA, etc.

Cuando se necesita un bloque de disco por cualquier motivo (inodo, directorio, datos) primero se verifica si está en la cache de buffers, se toma el bloque evitando el acceso a disco, si el bloque no se encuentra en cache, se lee de disco, se escribe en dicha cache y se lleva a donde se necesite.

**System Calls**

*System Calls. Descripción. Explicar su implementación. Para ello tenga en cuenta el apoyo del hardware. Modos de ejecución. Pilas y yodo aquello que involucre su ejecución.*

System Calls: Proporcionan los medios para que una aplicación de usuario le pida al sistema operativo que realice tareas reservadas del sistema operativo, en nombre del programa del usuario.

Cuando se ejecuta una llamada al sistema, el hardware la trata como una excepción que efectúa una transferencia a una posición especifica del vector de interrupción.

Dicha posición indica la dirección de la primera instrucción de la rutina de atención de la interrupción dada.

Existen 2 modos de ejecución:

* Modo Usuario: Cuando el sistema informático está ejecutando una aplicación de usuario.
* Modo Kernel: Cuando una aplicación de usuario solicita un servicio del sistema operativo a través de una System Call, debe pasar a modo Kernel para satisfacer dicha solicitud.

Para esto se necesita un bit extra en el hardware para indicar el modo actual.

Se utiliza una pila por cada modo de ejecución.

* Pila Modo Usuario: Contiene los argumentos, variables totales y otros datos relativos a funciones que se ejecutan en modo usuario. Es dinámica.
* Pila Modo Kernel: Contiene los frames de las funciones que se ejecutan en modo Kernel (estas funciones son las llamadas al sistema). Es estática.

Cuando se arranca el sistema, el hardware inicia en modo Kernel. El SO se carga, cambia a modo usuario y se cargan las aplicaciones de usuario.

Cuando se produce una excepción o una interrupción, es decir cuando el control pasa por el vector de interrupciones, el hardware cambia el bit a 0 (cero, Modo Kernel). El Kernel examina la instrucción para ver que system call se ha producido y el parámetro (para el tipo de servicio), verifica la corrección y legalidad de los parámetros, ejecuta la solicitud y devuelve el control a la siguiente instrucción a la de la llamada al servicio.

El modo dual nos proporciona los medios para proteger al sistema operativo de los usuarios que puedan causar errores, y también para proteger a los usuarios de los errores de otros usuarios.

Una vez que se dispone de protección de hardware, este detecta los errores de violación de modos y es el sistema operativo quien los maneja.

Si un programa de usuario falla de alguna forma, el hardware envía una excepción al sistema operativo que transfiere el control al propio sistema operativo a través de un vector de interrupciones, cuando se produce un error de programa, el sistema operativo debe terminar el programa de manera anormal.

Para asegurar que el sistema operativo mantiene el control sobre la CPU y una aplicación no monopolice la CPU, se puede utilizar un temporizador que puede configurarse para interrumpir la ejecución cada cierto periodo especifico.

Usualmente se utiliza un temporizador variable (reloj de frecuencia fija y contador).

El sistema operativo configura el contador, cada vez que el reloj avanza, el contador se decrementa.

Cuando el contador alcanza el valor cero, se produce una interrupción.

Antes de devolver el control al usuario, el sistema operativo se asegura de que el temporizador este configurado para realizar interrupciones.

**Working Set**

Working Set (Modelo del conjunto de trabajo).

Consiste en examinar las referencias de página más recientes dentro de esa ventana de trabajo (en el parámetro) las cuales conforman mi conjunto de trabajo.

Si una página está siendo usada en forma activa se va a encontrar en el conjunto. Si ya no está siendo utilizada será eliminada del conjunto en el tiempo dado por el parámetro en relación a su última referencia.