Introducción a los Sistemas Operativos

Resumen teorías

Memoria 2

Memoria virtual

Motivación

No todo el espacio de direcciones del proceso se necesitó en todo momento:

- Rutinas o Librerías que se ejecutan una única vez (o nunca)
- Partes del programa que no vuelven a ejecutarse
- Regiones de memoria alocadas dinámicamente y luego liberadas

No hay necesidad que la totalidad la imagen del proceso sea cargada en memoria

Conjunto residente

El SO puede traer a memoria las “piezas” de un proceso a medida que éste las necesita

Definiremos como “Conjunto Residente” o “Working Set” a la porción del espacio de direcciones del proceso que se encuentra en memoria

Con el apoyo del HW:

- Se detecta cuando se necesita una porción del proceso que no está en su conjunto residente
- Se debe cargar en memoria dicha porción para continuar con la ejecución.

Ventajas

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

Requerimientos

- El hardware debe soportar paginación por demanda (y/o segmentación por demanda)
- Un dispositivo de memoria secundaria (disco) que dé el apoyo para almacenar las secciones del proceso que no 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.

Con Paginación

- Cada proceso tiene su tabla de páginas
- Cada entrada en la tabla referencia al frame o marco en el que se encuentra la página en la memoria principal
- Cada entrada en la tabla de páginas tiene bits de control (entre otros):
    - Bit V: Indica si la página está en memoria (lo activa/desactiva el SO, lo consulta el HW)
    - Bit M: Indica si la página fue modificada. Si se modificó, en algún momento, se deben reflejar los cambios en Memoria Secundaria (lo activa/desactiva el HW, lo consulta el SO)

Fallo de página (Page Fault)

Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en la memoria principal. Bit V=0 (también marcado con i = inválido): la página no se encuentra en su conjunto residente

El HW detecta la situación y genera un trap al S.O

El S.O. Podrá colocar al proceso en estado de “Blocked” (espera) mientras gestiona que la página que se necesite se cargue

El S.O. busca un “Frame o Marco Libre” en la memoria y genera una operación de E/S al disco para copiar en dicho Frame la página del proceso que se necesita utilizar

El SO puede asignarle la CPU a otro proceso mientras se completa la E/S La E/S se realizará y avisará mediante interrupción su finalización

Cuando la operación de E/S finaliza, se notifica al SO y este actualiza la tabla de páginas del proceso:

- Coloca el Bit V en 1 en la página en cuestión

- Coloca la dirección base del Frame donde se colocó la página

El proceso que generó el Fallo de Página vuelve a estado de Ready (listo)
Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes generó el fallo de página

La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles

Cada vez que hay que alocar una página en un marco, se produce un fallo de página

¿Qué sucede si es necesario alocar una página y ya no hay marcos disponibles?: Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos (FIFO, Óptimo, LRU, etc.)

¿Cuál es el mejor algoritmo?: El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro

Performance

Si los page faults son excesivos, la performance del sistema decae

Tasa de Page Faults 0 <= p <= 1

- Si p = 0 no hay page faults
- Si p = 1, cada acceso a memoria genera un page fault

Effective Access Time (EAT)

EAT = (1 – p) * memory access + p * (page_fault_overhead + [swap_page_out] + swap_page_in + restart_overhead)

Tabla de páginas

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

Tabla de páginas de 2 niveles

El propósito de la tabla de páginas multinivel es dividir la tabla de páginas lineal en múltiples tablas de páginas

Cada tabla de páginas suele tener el mismo tamaño pero se busca que tengan un menor número de páginas por tabla

La idea general es que cada tabla sea más pequeña

Se busca que la tabla de páginas no ocupe demasiada memoria RAM

Además solo se carga una parcialidad de la tabla de páginas (solo lo que se necesite resolver)

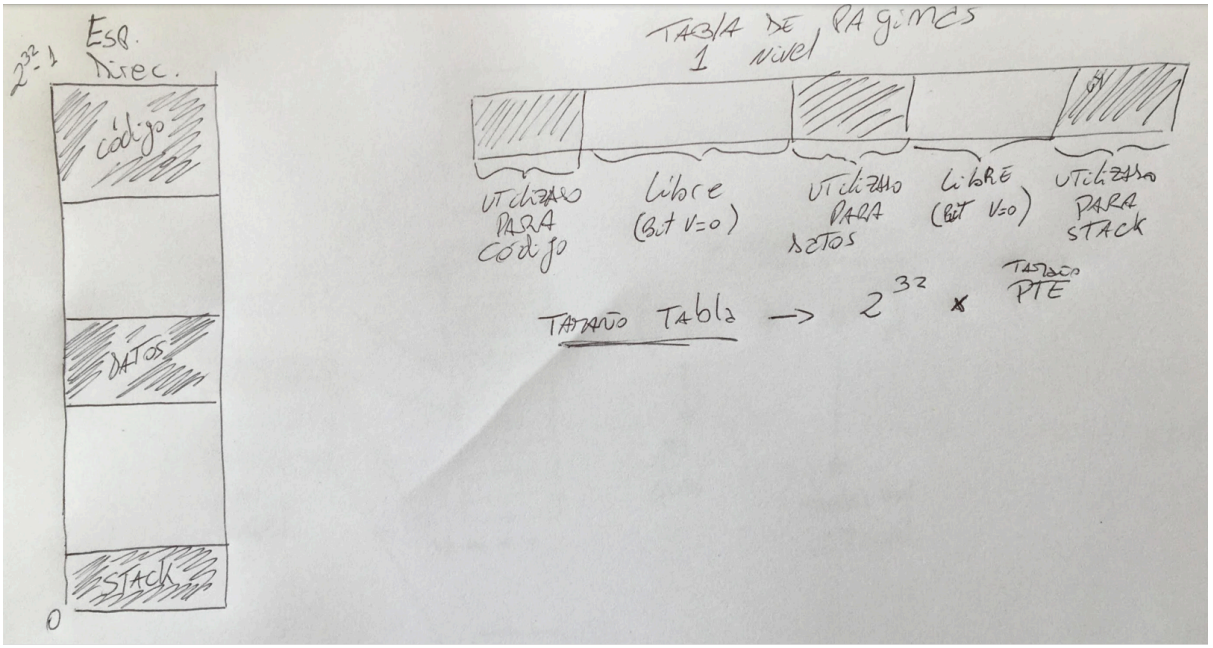
Existe un esquema de direccionamientos indirectos

Desventajas

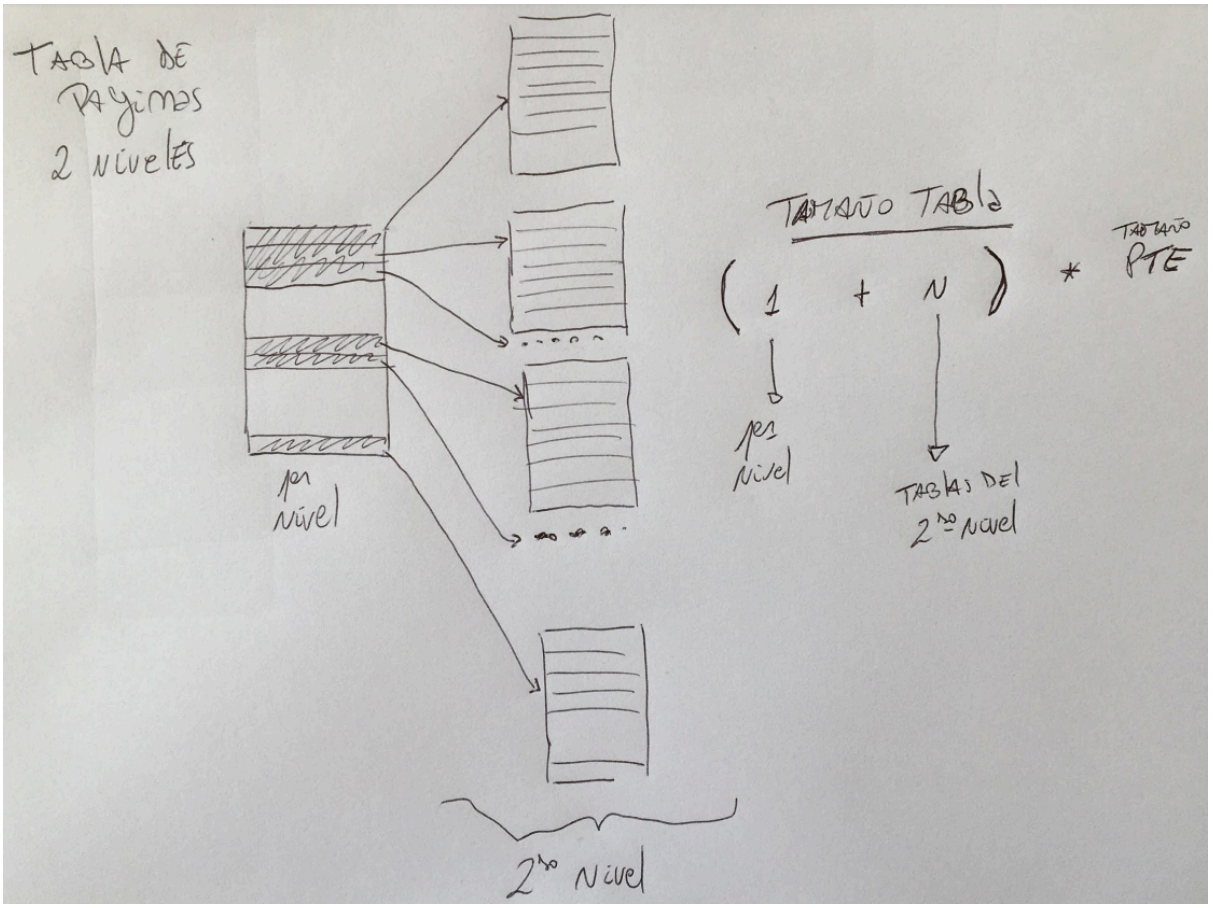
La cantidad de accesos a memoria para traducir una dirección aumenta en 1 por cada nivel de indirección

Ventajas

- Se puede paginar la tabla de páginas, ya que las tablas de páginas de segundo nivel se corresponden con el tamaño de una página, por lo que se podría sacar de memoria principal
- Se ahorra espacio en RAM:



En el caso de una tabla de páginas de 1 nivel, el tamaño de la misma va a ser siempre el máximo, independientemente de cuántas páginas estén siendo utilizadas.



En el caso de una tabla de páginas de 2 niveles, el tamaño de la misma va a variar, dependiendo de la cantidad de páginas en memoria, ya que la única tabla que sí o sí va a estar en memoria es la de 1er nivel, mientras que las tablas de páginas de 2do nivel, van a ser direccionadas solo en caso de existir esa PTE en la tabla de 1er nivel

Tabla invertida

- Utilizada en Arquitecturas donde el espacio de direcciones es muy grande
- Las tablas de páginas ocuparían muchos niveles y la traducción sería costosa. Por esta razón se adopta esta técnica
- Hay una entrada por cada marco de página en la memoria real
- Es la visión inversa a la que veníamos viendo
- Hay una sola tabla para todo el sistema
- El espacio de direcciones de la tabla se refiere al espacio físico de la RAM, en vez del espacio de direcciones virtuales de un proceso

- El número de página es transformado en un valor de HASH
- El HASH se usa como índice de la tabla invertida para encontrar el marco asociado
- Se define un mecanismo de encadenamiento para solucionar colisiones (cuando el hash da igual para 2 direcciones virtuales)
- Sólo se mantienen los PTEs de páginas presentes en memoria física
- La tabla invertida es organizada como tabla hash en memoria principal
- Se busca indexadamente por número de página virtual
- Si está presente en tabla, se extrae el marco de página y sus protecciones
- Si no está presente en tabla, corresponde a un fallo de página

Tamaño de página

- Pequeño
    - Menor Fragmentación Interna.
    - Más páginas requeridas por proceso
    - Tablas de páginas más grandes
    - Más páginas pueden residir en memoria
- Grande
    - Mayor Fragmentación interna
    - La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente: más rápido mover páginas hacia la memoria principal
    - Voy a tener páginas menos concisas (con información que puede no ser necesaria)

Translation Lookaside Buffer (TLB)

Cada referencia en el espacio virtual puede causar 2 (o más) accesos a la memoria física

Uno (o más) para obtener la entrada en tabla de páginas

Uno para obtener los datos

Para solucionar este problema, una memoria caché de alta velocidad es usada para almacenar entradas de páginas: el TLB

El buffer de traducción adelantada contiene las entradas de la tabla de páginas que fueron usadas más recientemente

1. Dada una dirección virtual, el procesador examina la TLB
2. Si la entrada de la tabla de páginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física
3. Si la entrada no es encontrada en la TLB (miss), el número de página es usado como índice en la tabla de páginas del proceso
4. Se controla si la página está en la memoria. Si no está, se genera un Page Fault
5. La TLB es actualizada para incluir la nueva entrada
6. El cambio de contexto genera la invalidación de las entradas de la TLB

Asignación de marcos

¿Cuántas páginas de un proceso se pueden encontrar en la memoria?

En la realidad todos los procesos comienzan con una cantidad fija de frames y, si un proceso necesita más, y hay frames libres, el SO le puede dar de más, o si tiene más de los que está usando el SO le va a sacar (mezcla entre dinámica y fija)

Asignación Dinámica

El número de marcos para cada proceso varía

Asignación fija

Número fijo de marcos para cada proceso

- Asignación equitativa: si tengo 100 frames y 5 procesos, 20 frames para cada proceso
- Asignación Proporcional: Se asigna acorde al tamaño del proceso. cantidad_frames_pr_i = (tmñ_pr_i /tmñ_total_procesos) * frames_totales

Reemplazo de páginas

Qué sucede si ocurre un fallo de página y todos los marcos están ocupados

“Se debe seleccionar una página víctima”

¿Cuál sería Reemplazo Óptimo?: que la página a ser removida no sea referenciada en un futuro próximo

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

Alcance del reemplazo

Reemplazo Global

El fallo de página de un proceso puede reemplazar la página de cualquier proceso

El SO no controla la tasa de page-faults de cada proceso

Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él

Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad

Reemplazo Local

El fallo de página de un proceso solo puede reemplazar sus propias páginas (de su Conjunto Residente)

No cambia la cantidad de frames asignados

El SO puede determinar cuál es la tasa de page-faults de cada proceso

Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos

Es la generalmente utilizada porque permite individualizar la tasa de PF de cada proceso (en el global no se puede porque puede ser otro el que le robe páginas, y por eso sucedan PFs), entonces el SO puede aumentar o reducir su conjunto residente para optimizar el uso de marcos

	Local Replacement	**Global Replacement**
Fixed Allocation	• Number of frames allocated to a process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Algoritmos de reemplazo

ÓPTIMO

Es solo teórico

FIFO

Es el más sencillo

LRU (Least Recently Used)

Requiere soporte del hardware para mantener timestamps de acceso a las páginas

Favorece a las páginas más recientemente accedidas

2da Chance

Un avance del FIFO tradicional que beneficia a las páginas más referenciadas

NRU (Non Recently Used)

Utiliza bits R y M Favorece a las páginas que fueron usadas recientemente

Orden de reemplazo:

Orden	Bit R	Bit M
1	0	0
2	0	1
3	1	0
4	1	1

Memoria 3

Thrashing (hiperpaginación)

Decimos que un sistema está en thrashing cuando pasa más tiempo paginando (tratando PF) que ejecutando procesos

Como consecuencia, hay una baja importante de performance en el sistema

Ciclo del thrashing

1. El kernel monitorea el uso de la CPU
2. Si hay baja utilización, aumenta el grado de multiprogramación
3. Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos
4. Un proceso necesita más frames. Comienzan los page-faults y robo de frames a otros procesos
5. Por swapping de páginas, y encolamiento en dispositivos, baja el uso de la CPU
6. Vuelve a 1)

El scheduler de CPU y el thrashing

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

Control del thrashing

Se puede limitar el thrashing usando algoritmos de reemplazo local

Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos

Si bien perjudica la performance del sistema, es controlable

Conclusión sobre thrashing

Si un proceso cuenta con todos los frames que necesita, no habría thrashing

Una manera de abordar esta problemática es utilizando la estrategia de Working Set, la cual se apoya en el modelo de localidad

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

Modelo de localidad

Cercanía de referencias o principio de cercanía

Las referencias a datos y programa dentro de un proceso tienden a agruparse

La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento

En cortos períodos de tiempo, el proceso necesitará pocas “piezas” del proceso (por ejemplo, una página de instrucciones y otra de datos…)

Un programa se compone de varias localidades

Ejemplo: Cada rutina será una nueva localidad: se referencian sus direcciones (cercanas) cuando se está ejecutando

Para prevenir la hiperactividad, un proceso debe tener en memoria sus páginas más activas (menos page faults)

Modelo de working set

Se basa en el modelo de localidad

Ventana del working set (Δ): las Δ referencias de memoria más recientes

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

Selección del Δ

Δ chico: no cubrirá la localidad

Δ grande: puede tomar varias localidades

Medida del working set

m = cantidad frames disponibles

WSSi= tamaño del working set del proceso pi

∑ WSSi=D

D= demanda total de frames

Si D>m, habrá thrashing

Prevención del thrashing

SO monitorea c/ proceso, dándole tantos frames hasta su WSSi (medida del working set del proceso pi)

Si quedan frames, puede iniciar otro proceso

Si D crece, excediendo m, se elige un proceso para suspender, reasignándose sus frames

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

Problema del modelo del WS

Mantener un registro de los WSSi (medida del working set del proceso pi)

La ventana es móvil

Prevención de thrashing por PFF

La técnica PFF (Page Fault Frequency o Frecuencia de Fallo de Páginas), en lugar de calcular el WS de los procesos, utiliza la tasa de fallos de página para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS

PFF alta Se necesitan más frames PFF baja

Los procesos tienen frames asignados que le sobran

Establecer tasa de PF aceptable:

- Si la tasa actual es baja, el proceso puede perder frames (si otro proceso los necesita)
- Si la tasa actual es alta, el proceso gana frames

Establecer límites superior e inferior de las PFF’s deseadas.

Si se Excede PFF máx.: se le asignan más frames al proceso, ya que el mismo genera muchos PF y probablemente los esté necesitando

Si la PFF está por debajo del mínimo: se le pueden quitar frames al proceso para ser utilizados en los que necesitan más

Se puede llegar a suspender un proceso si no hay más frames libres y todos los procesos están por arriba de PFF máx.

Demonio de Paginación

Proceso creado por el SO durante el arranque que apoya a la administración de la memoria

Se ejecuta cuando el sistema tiene una baja utilización o algún parámetro de la memoria lo indica (Poca memoria libre / Mucha memoria modificada)

Tareas:

- Limpiar páginas modificadas sincronizándolas con el swap:
    - Reducir el tiempo de swap posterior ya que las páginas están “limpias”
    - Reducir el tiempo de transferencia al sincronizar varias páginas contiguas
- Mantener un cierto número de marcos libres en el sistema
- Demorar la liberación de una página hasta que haga falta realmente
- Modifican working set (achican al haber pocos PF, agrandan al haber muchos)

Memoria compartida

Gracias al uso de la tabla de páginas varios procesos pueden compartir un marco de memoria. Para ello ese marco debe estar asociado a una página en la tabla de páginas de cada proceso

El número de página asociado al marco puede ser diferente en cada proceso

Código compartido

Los procesos comparten una copia de código (sólo lectura) por ej. editores de texto, compiladores, etc

Los datos son privados a cada proceso y se encuentran en páginas no compartidas

Mapeo de archivo en memoria

Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales

El contenido del archivo no se sube a memoria hasta que se generan Page Faults

El contenido de la página que genera el PF es obtenido desde el archivo asociado no del área de intercambio

Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente

Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el Sistema de Archivos

Es utilizado comúnmente para asociar librerías compartidas o DLLs

Copia en Escritura

La copia en escritura (Copy-on-Write, COW) permite a los procesos compartir inicialmente las mismas páginas de memoria

Si uno de ellos modifica una página compartida la página es copiada

En fork, permite inicialmente que padre e hijo utilicen las mismas páginas sin necesidad de duplicación.

COW permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas

Área de Intercambio

Área dedicada, separada del Sistema de Archivos (Por ejemplo, en Linux)

Un archivo dentro del Sistema de Archivos (Por ejemplo, Windows)

Técnicas para la Administración

A. Cada vez que se crea un proceso se reserva una zona del área de intercambio igual al tamaño de imagen del proceso. A cada proceso se le asigna la dirección en disco

de su área de intercambio. La lectura se realiza sumando el número de página virtual a la dirección de comienzo del área asignada al proceso

B. No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la página vuelve a memoria. Problema: se debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco

Cuando una página no está en memoria, sino en swap, como podríamos saber en qué parte del área de intercambio está?

Cada área de swap es dividida en un número fijo de slots según el tamaño de la página

Cuando una página es llevada a disco, Linux utiliza la PTE para almacenar 2 valores:

1. En número de área
2. El desplazamiento en el área (24 bits, lo que limita el tamaño máximo del área a 64 Gb)

Entrada/Salida

Responsabilidades del SO

Controlar dispositivos de E/S:

- Generar comandos
- Manejar interrupciones
- Manejar errores

Proporcionar una interfaz de utilización

Problemas

- Heterogeneidad de dispositivos
- Características de los dispositivos
- Velocidad
- Nuevos tipos de dispositivos
- Diferentes formas de realizar E/S (ver anexo)

Metas, objetivos y servicios

Generalidad

Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada

Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más “bajos” para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock

Eficiencia

Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria y la CPU

El uso de la multiprogramación permite que un procesos espere por la finalización de su I/O mientras que otro proceso se ejecuta

Planificación

Organización de los requerimientos a los dispositivos

Ej: Planificación de requerimientos a disco para minimizar tiempos

Buffering

Almacenamiento de los datos en memoria mientras se transfieren

Solucionar problemas de velocidad entre los dispositivos

Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos

Caching

Mantener en memoria copia de los datos de reciente acceso para mejorar performance

Spooling

Administrar la cola de requerimientos de un dispositivo

Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo: Por ej. Impresora

Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo

Reserva de Dispositivos

Acceso exclusivo

Manejo de Errores

El S.O. debe administrar errores ocurridos (lectura de un disco, dispositivo no disponible, errores de escritura) La mayoría retorna un número de error o código cuando la I/O falla. Logs de errores

Formas de realizar I/O

- Bloqueante: El proceso se suspende hasta que el requerimiento de I/O se completa:
    - Fácil de usar y entender
    - No es suficiente bajo algunas necesidades

- No Bloqueante: El requerimiento de I/O retorna en cuanto es posible:
    - Ejemplo: Interfaz de usuario que recibe input desde el teclado/mouse y se muestra en pantalla
    - Ejemplo: Aplicación de video que lee frames desde un archivo mientras va mostrándolo en pantalla

Diseño

En capas, donde se abstraen en cada una de las capas, objetivos específicos, y se oculta la implementación de esa capa a las demás, solo mostrando la interfaz necesaria al resto

Software capa de usuario

Librerías de funciones

- Permiten acceso a SysCalls
- Implementan servicios que no dependen del Kernel (no requieren de modo kernel)

Procesos de apoyo

- Demonio de Impresión (spooling)

Software del SO independiente al dispositivo

Brinda los principales servicios de E/S antes vistos:

- Interfaz uniforme
- Manejo de errores
- Buffer
- Asignación de Recursos

- Planificación

El Kernel mantiene la información de estado de cada dispositivo o componente:

- Archivos abiertos
- Conexiones de red
- Etc.

Hay varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc.

No se mete en el tipo de dispositivo de entrada salida en particular (por ej., se encarga de la planificación de discos, sin importar el tipo de disco)

Controladores (Drivers)

Contienen el código dependiente del dispositivo

Manejan un tipo dispositivo

Traducen los requerimientos abstractos (read/write/select) en los comandos para el dispositivo:

- Escribe sobre los registros del controlador
- Acceso a la memoria mapeada
- Encola requerimientos

Comúnmente las interrupciones generadas por los dispositivos son atendidas por funciones provistas por el driver

Interfaz entre el SO y el HARD

Forman parte del espacio de memoria del Kernel, que en general se cargan como módulos

Los fabricantes de HW implementan el driver en función de una API especificada por el SO: open(), close(), read(), write(), etc.

Para agregar nuevo HW sólo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel

Gestor de interrupciones

Atiende todas las interrupciones del HW

Deriva al driver correspondiente según interrupción

Resguarda información

Independiente del Driver

Performance

I/O es uno de los factores que más afectan a la performance del sistema:

- Utiliza mucho la CPU para ejecutar los drivers y el código del subsistema de I/O
- Provoca Context switches ante las interrupciones y bloqueos de los procesos
- Utiliza el bus de memoria en copia de datos:
    - Aplicaciones (espacio usuario) – Kernel
    - Kernel (memoria fisica) - Controladora

Mejorar la Performance

Reducir el número de context switches (Si el proceso 1 realiza una operación de I/O, y luego se pasa a ejecutar al proceso 2 y durante su ejecución termina la operación de I/O del proceso 1, que no se haga un context switch al proceso 1, sino que el kernel resuelva lo necesario, ya que está en modo kernel)

Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación

Reducir la frecuencia de las interrupciones, utilizando:

- Transferencias de gran cantidad de datos
- Controladoras más inteligentes
- Polling, si se minimiza la espera activa

Utilizar DMA

Archivos 1

Archivo

Entidad abstracta con nombre

Espacio lógico continuo (no necesariamente es físicamente continuo) y direccionable

Provee a los programas de datos (entrada)

Permite a los programas guardar datos (salida)

El programa mismo es información que debe guardarse

Punto de vista del Usuario

Qué operaciones se pueden llevar a cabo

Cómo nombrar a un archivo

Cómo asegurar la protección

Como compartir archivos

No tratar con aspectos físicos

Punto de vista del Diseño

Implementar archivos

Implementar directorios

Manejo del espacio en un medio de almacenamiento

Manejo del espacio libre

Eficiencia y mantenimiento

Sistema de Manejo de Archivos

Conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos:

- Crear
- Borrar
- Buscar
- Copiar
- Leer

- Escribir

Facilita el acceso a los archivos por parte de las aplicaciones

Permite la abstracción al programador, en cuanto al acceso de bajo nivel (el programador no desarrolla el soft de administración de archivos)

Objetivos de SO en cuanto a archivos

Cumplir con la gestión de datos

Cumplir con las solicitudes del usuario

Minimizar/eliminar la posibilidad de perder o destruir datos:

- Garantizar la integridad del contenido de los archivos

Dar soporte de E/S a distintos dispositivos

Brindar un conjunto de interfaces de E/S para tratamiento de archivos

Tipos de Archivos

- Archivos Regulares:
  - Texto Plano
    - Source File
  - Binarios
    - Object File
    - Executable File
- Directorios
  - Archivos que mantienen la estructura en el FileSystem

Atributos de un Archivo

- Nombre
- Identificador
- Tipo
- Localización
- Tamaño
- Protección, Seguridad y Monitoreo:

- Owner, permisos, password
- Momento en que el usuario lo modifico, creo, accedió por última vez
- ACLs

Directorios

Contiene información acerca de archivos y directorios que están dentro de él

El directorio es, en sí mismo, un archivo

Interviene en la resolución entre el nombre y el archivo mismo (el nombre completo de un archivo (path) está definido entre los directorios hasta llegar a él, y su nombre base (basename))

Operaciones en directorios:

- Buscar un archivo
- Crear un archivo (entrada de directorio)
- Borrar un archivo
- Listar el contenido
- Renombrar archivos

Estructura de Directorios

El directorio actual se lo llama “directorio de trabajo” (working directory), todo proceso tiene un working directory

Dentro del directorio de trabajo, se pueden referenciar los archivos tanto por su PATH absoluto como por su PATH relativo indicando solamente la ruta al archivo desde el directorio de trabajo.

Compartir archivos

En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos

Debe ser realizado bajo un esquema de protección:

- Derechos de acceso
- Manejo de accesos simultáneos

Protección

El propietario/administrador debe ser capaz de controlar:

- Qué se puede hacer: Derechos de acceso
- Quién lo puede hacer

Derechos de acceso

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

Permisos:

- Execution
- Reading
- Writing
- Appending
- Updating
- Changing
- Deletion

Owners (propietarios):

- Tiene todos los derechos
- Puede dar derechos a otros usuarios
- Se determinan clases:
    - Usuario específico
    - Grupos de usuarios
    - Todos (archivos públicos)

Archivos 2

Conceptos

Sector: Unidad de almacenamiento utilizada en los Discos Rígidos

Bloque/Cluster: Conjuntos de sectores consecutivos

File System: Define la forma en que los datos son almacenados

FAT (File Allocation Table): Contiene información sobre en qué lugar están alocados los distintos archivos

Pre-asignación

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

Se tiende a definir espacios mucho más grandes que lo necesario

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

¿Qué pasa cuando el archivo supera el espacio asignado?

Asignación Dinámica

El espacio se solicita a medida que se necesita

Los bloques de datos pueden quedar de manera no contigua

Formas de Asignación

Continua

Conjunto continuo de bloques son utilizados

Se requiere una pre-asignación: se debe conocer el tamaño del archivo durante su creación

FAT: única entrada que incluye bloque de inicio y longitud

El archivo puede ser leído con una única operación

Puede existir fragmentación externa

Problemas

Encontrar bloques libres continuos en el disco

Incremento del tamaño de un archivo

Encadenada

Asignación en base a bloques individuales

Cada bloque tiene un puntero al próximo bloque del archivo

FAT: única entrada por archivo: Bloque de inicio y tamaño del archivo

No hay fragmentación externa

Útil para acceso secuencial (no random)

Los archivos pueden crecer bajo demanda

No se requieren bloques contiguos

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

Indexada

La FAT contiene un puntero al bloque índice

El bloque índice no contiene datos propios del archivo, sino que contiene un índice a los bloques que lo componen

Asignación en base a bloques individuales

No se produce Fragmentación Externa

El acceso “random” a un archivo es eficiente

FAT: única entrada con la dirección del bloque de índices (index node / i-node)

Variantes

Asignación por secciones

A cada entrada del bloque índice se agrega el campo longitud

El índice apunta al primer bloque de un conjunto almacenado de manera contigua

Niveles de indirección

Existen bloques directos de datos

Otros bloques son considerados como bloque índices (apuntan a varios bloques de datos)

Puede haber varios niveles de indirección

Gestión de Espacio Libre

Control sobre cuáles de los bloques de disco están disponibles

Tablas de bits

Tabla (vector) con 1 bit por cada bloque de disco

Cada entrada: 0 = bloque libre 1 = bloque en uso

Ventaja

Fácil encontrar un bloque o grupo de bloques libres

Desventaja

Tamaño del vector en memoria tamaño disco bytes/tamaño bloque en sistema archivo

Ej.: Disco 16 Gb con bloques de 512 bytes 32 Mb

Bloques Encadenados

Se tiene un puntero al primer bloque libre

Cada bloque libre tiene un puntero al siguiente bloque libre

Ineficiente para la búsqueda de varios bloques libres: hay que realizar varias operaciones de E/S para obtener un grupo libre

Problemas con la pérdida de un enlace

Difícil encontrar bloques libres consecutivos

Indexación (o agrupamiento)

Variante de “bloques libres encadenados”

El primer bloque libre contiene las direcciones de N bloques libres

Las N-1 primeras direcciones son bloques libres

La N-ésima dirección referencia otro bloque con N direcciones de bloques libres

Archivos 3

UNIX

Tipos de Archivos

- Archivo común
- Directorio
- Archivos especiales (dispositivos /dev/sda)
- Named pipes (comunicación entre procesos)
- Links (comparten el i-nodo, solo dentro del mismo filesystem)
- Links simbólicos (tiene i-nodo propio, pueden usarse en filesystems diferentes)

Estructura del Volumen

Cada disco físico puede ser dividido en uno o más volúmenes

Cada volumen o partición contiene un sistema de archivos

Cada sistema de archivos contiene:

- Boot Block: Código para bootear el S.O.
- Superblock: Atributos sobre el File System: Bloques/Clusters libres
- I-NODE Table: Tabla que contiene todos los I-NODOS
- Data Blocks: Bloques de datos de los archivos

Windows

File Systems Soportados

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

- New Technology File System (NTFS)

FAT

FAT (File Allocation Table) es un sistema de archivos utilizado originalmente por DOS y Windows 9x

¿Porqué Windows aún soporta FAT file systems?:

- Por compatibilidad con otro SO en sistemas multiboot
- Para permitir upgrades desde versiones anteriores
- Para formato de dispositivos como diskettes

Las distintas versiones de FAT se diferencian por un número que indica la cantidad de bits que se usan para identificar diferentes bloques o clusters: FAT12 - FAT16 - FAT32

Se utiliza un mapa de bloques del sistema de archivos, llamado FAT

La FAT tiene tantas entradas como bloques

La FAT, su duplicado y el directorio raíz se almacenan en los primeros sectores de la partición

Se utiliza un esquema de asignación encadenada

La única diferencia es que el puntero al próximo bloque está en la FAT y no en los bloques

loques libres y dañados tienen códigos especiales

FAT12

En sistemas FAT12, al utilizarse 12 bits para la identificación del sector, la misma se limita a 4096 sectores

Windows utiliza tamaños de sector desde los 512 bytes hasta 8 KB, lo que limita a un tamaño total de volumen de 32 MB 4096 * 8 KB

Windows utiliza FAT12 como sistema de archivos de los diskettes de 3,5 y 12 pulgadas que pueden almacenar hasta 1,44 MB de datos

FAT16

FAT16 al utilizar 16 bits para identificar cada sector puede tener hasta 65.536 sectores en un volúmen

En windows el tamaño de sector en FAT16 varía desde los 512 bytes hasta los 64 KB, lo que limita a un tamaño máximo de volumen de 4 GB

El tamaño de sector dependía del tamaño del volumen al formatearlo

FAT32

FAT32 fue el Filesystem más reciente de la línea (posteriormente salió exFAT que algunos lo conocen como FAT64)

FAT32 utiliza 32 bits para la identificación de sectores, pero reserva los 4 bits superiores, con lo cual efectivamente solo se utilizan 28 bits para la identificación

El tamaño de sector en FAT 32 puede ser de hasta 32 KB, con lo cual tiene una capacidad teórica de direccionar volúmenes de hasta 8 TB

El modo de identificación y acceso de los sectores lo hace más eficiente que FAT16

Con tamaño de sector de 512 bytes, puede direccionar volúmenes de hasta 128 GB

NTFS

NTFS es el filesystem nativo de Windows desde Windows NT

NTFS usa 64-bit para referenciar sectores: teóricamente permite tener volúmenes de hasta 16 Exabytes (16 billones de GB)

¿Por qué usar NTFS en lugar de FAT? FAT es simple, más rápido para ciertas operaciones, pero NTFS soporta:

- Tamaños de archivo y de discos mayores
- Mejora performance en discos grandes
- Nombres de archivos de hasta 255 caracteres
- Atributos de seguridad
- Transacciones

Buffer-Caché

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

Cuando un proceso quiere acceder a un bloque de la caché hay dos alternativas:

1. Se copia el bloque al espacio de direcciones del usuario, esto no permitiría compartir el bloque
2. Se trabaja como memoria compartida, que permite acceso a varios procesos. Esta área de memoria debe ser limitada, con lo cual debe existir un algoritmo de reemplazo

Estrategia de reemplazo

Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado

Es una lista de bloques, donde el último es el más recientemente usado (LRU, Least Recently Used)

Cuando un bloque se referencia o entra en la caché queda al final de la lista

No se mueven los bloques en la memoria: se asocian punteros

Otra alternativa: Least Frequently Used: se reemplaza el que tenga menor número de referencias

En System V

Es una estructura formada por buffers

El kernel asigna un espacio en la memoria durante la inicialización para esta estructura

Un buffer tiene dos partes:

1. Header: Contiene información del bloque, número del bloque, estado, relación con otros buffers
2. El buffer en sí: el lugar donde se almacena el bloque de disco traído a memoria

El módulo de buffer cache es independiente del sistema de archivos y de los dispositivos de hardware

Es un servicio del SO

Header

- Identifica el número. de dispositivo y número de bloque
- Estado
- Punteros a:
    - 2 punteros para la hash queue
    - 2 punteros para la free list
    - 1 puntero al bloque en memoria

Estado de los buffers

Free o disponible

Busy o no disponible (en uso por algún proceso)

Se está escribiendo o leyendo del disco

Delayed Write (DW): buffers modificados en memoria, pero los cambios no han sido reflejados en el bloque original en disco

Free list

Organiza los buffers disponibles para ser utilizados para cargar nuevos bloque de disco

No necesariamente los buffers están vacíos (el proceso puede haber terminado, liberado el bloque pero sigue en estado delayed write)

Se ordena según LRU (least recently used)

Sigue el mismo esquema de la Hash queue pero contiene los headers de los buffers de aquellos procesos que ya han terminado

Hash Queues

Son colas para optimizar la búsqueda de un buffer en particular

Los headers de los buffers se organizan según una función de hash usando (dispositivo, #bloque)

Al número de bloque (dispositivo/bloque) se le aplica una función de hash que permite agrupar los buffers cuyo resultado dio igual para hacer que las búsquedas sean más eficientes

Se busca que la función de hash provea alta dispersión para lograr que las colas de bloques no sean tan extensas

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

El header de un buffer siempre está en la Hash Queue

Si el proceso que referenciaba al buffer terminó, va a estar en la Hash Queue y en la Free List

Funcionamiento del buffer cache

Cuando un proceso quiere acceder a un archivo, utiliza su i-nodo para localizar los bloques de datos donde se encuentra éste

El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S

Se pueden dar 5 escenarios:

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