***Administración de Memoria – I***

Memoria

La organización y administración de la “memoria principal” es uno de los factores más importantes en el diseño de los S. O.

Los programas y datos deben estar en el almacenamiento principal para:

\* Poderlos ejecutar.

\* Referenciarlos directamente.

El SO debe:

\* Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no.

\* Asignar espacio en memoria principal a los procesos cuando estos la necesitan.

\* Libera espacio de memoria asignada a procesos que han terminado.

\*Se espera de un S.O. un uso eficiente de la memoria con el fin de alojar el mayor número de procesos

\*Lograr que el programador se abstraiga de la alocación de los programas

\*Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros

\* Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria (librerías, código en común, etc.)

\* Garantizar la performance del sistema

Administración de Memoria

División Lógica de la Memoria Física para alojar múltiples procesos

– Garantizando protección

– Depende del mecanismo provisto por el HW

Asignación eficiente

– Contener el mayor numero de procesos para garantizar el mayor uso de la CPU por los mismos

Requisitos

[]Reubicación

\*El programador no debe ocuparse de conocer donde será colocado en la Memoria RAM

\*Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.

\*Las referencias a la memoria se deben “traducir” según ubicación actual del proceso

[]Protección

\*Los procesos NO deben referenciar – acceder - a direcciones de memoria de otros procesos, salvo que tengan permiso

\*El chequeo se debe realizar durante la ejecución: NO es posible anticipar todas las referencias a memoria que un proceso puede realizar.

[]Compartición

\*Permitir que varios procesos accedan a la misma porción de memoria. ωEj: Rutinas comunes, librerías, espacios explícitamente compartidos, etc.

\*Permite un mejor uso – aprovechamiento - de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones

Abstracción - Espacio de Direcciones

\*Rango de direcciones (a memoria) posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.

\*El tamaño depende de la Arquitectura del Procesador

\*32 bits: 0 .. 232 - 1

\*64 bits: 0 .. 264 – 1

\*Es independiente de la ubicación “real” del proceso en la Memoria RAM

Direcciones

Lógicas

\*Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.

\*Representa una dirección en el “Espacio de Direcciones del Proceso”

Físicas

\*Referencia una localidad en la Memoria Física (RAM) - Dirección absoluta

En caso de usar direcciones Lógicas, es necesaria algún tipo de conversión a direcciones Físicas.

Conversión de Direcciones

Una forma simple de hacer esto es utilizando registros auxiliares

\*Registro Base: Dirección de comienzo del Espacio de Direcciones del proceso en la RAM

\*Registro Limite: Dirección final del proceso o medida del proceso – Tamaño de su Espacio de Direcciones

\*Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria.

\*Varían entre procesos (Context Switch)

Dir. Lógicas vs. Físicas

Si la CPU trabaja con direcciones lógicas, para acceder a memoria principal, se deben transformar en direcciones físicas.

– Resolución de direcciones (address-binding): transformar la dirección lógica en la dirección física correspondiente

Resolución en momento de compilación (Archivos .com de DOS) y en tiempo de carga

- Direcciones Lógicas y Físicas son idénticas

-Para reubicar un proceso es necesario recompilarlo o recargarlo.

Resolución en tiempo de ejecución

-Direcciones Lógicas y Físicas son diferentes

-Direcciones Lógicas son llamadas “Direcciones Virtuales”

-La reubicación se puede realizar fácilmente

-El mapeo entre “Virtuales” y “Físicas” es realizado por hardware

• Memory Management Unit (MMU)

Memory Management Unit (MMU)

Dispositivo de Hardware que mapea direcciones virtuales a físicas

-Es parte del Procesador

-Re-programar el MMU es una operación privilegiada

• solo puede ser realizada en Kernel Mode

El valor en el “registro de realocación” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.

\*Los procesos nunca usan direcciones físicas

Mecanismos de asignacion de memoria

Particiones Fijas: El primer esquema implementado

\* La memoria se divide en particiones o regiones de tamaño Fijo (pueden ser todas del mismo tamaño o no)

\*⎫ Alojan un proceso cada una

\*Cada proceso se coloca de acuerdo a algún criterio (First Fit, Best Fit, Worst Fit, Next Fit) en alguna partición

Particiones dinámicas: La evolución del esquema anterior

\* Las particiones varían en tamaño y en número

\* Alojan un proceso cada una

\* Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso

Fragmentación

La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua

Fragmentación Interna:

\*Se produce en el esquema de particiones Fijas

\* Es la porción de la partición que queda sin utilizar

Fragmentación Externa:

\* Se produce en el esquema de particiones dinámicas

\* Son huecos que van quedando en la memoria a medida que los procesos finalizan

\* Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos utilizar

\* Para solucionar el problema se puede acudir a la compactación, pero es muy costosa

Problemas del esquema

El esquema de Registro Base + Limite presenta problemas:

– Necesidad de almacenar el Espacio de Direcciones de forma continua en la Memoria Física

– Los primeros SO definían particiones fijas de memoria, luego evolucionaron a particiones dinámicas

– Fragmentación

– Mantener “partes” del proceso que no son necesarias

– Los esquemas de particiones fijas y dinámicas no se usan hoy en día

Solución: Segmentación – Paginación

Segmentacion

Esquema que se asemeja a la “visión del usuario”. El programa se divide en partes/secciones

Un programa es una colección de segmentos. Un segmento es una unidad lógica como:

\*Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc.

\*Puede causar Fragmentación

Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas).

Las direcciones Lógicas consisten en 2 partes: Selector de Segmento y Desplazamiento dentro del segmento

Tabla de Segmentos

Permite mapear la dirección lógica en física.

Cada entrada contiene:

ω Base: Dirección física de comienzo del segmento

ω Limit: Longitud del Segmento

Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos.

Segment-table length register (STLR) : cantidad de segmentos de un programa

Paginación

Memoria Física es dividida lógicamente en pequeños trozos de igual tamaño [] Marcos

Memoria Lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos [] Paginas

El SO debe mantener una tabla de paginas por cada proceso, donde cada entrada contiene (entre otras) el Marco en la que se coloca cada pagina.

La dirección lógica se interpreta como: – un numero de pagina y un desplazamiento dentro de la misma.

El espacio de direcciones de un proceso no necesariamente debe estar “contiguo” en la memoria para poder ejecutarse. El hardware traduce direcciones lógicas a direcciones físicas utilizando las tablas de páginas que el SO administra

Segmentación Paginada

La paginación

\*Transparente al programador

\*Elimina Fragmentación externa.

Segmentación

\*Es visible al programador

\*Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección

Segmentación Paginada: Cada segmento es dividido en paginas de tamaño fijo

Motivación para Memoria Virtual

Podemos pensar también que, 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

\*Etc.

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

Como se puede trabajar

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

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

Alguna bibliografía lo llama “Working Set”

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.

¿Que se necesita para Memoria Virtual?

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.

Memoria Virtual con paginacion

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 el HW, lo consulta y desactiva el SO )

Fallo de páginas (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

ωColoca el Bit V en 1 en la página en cuestión

ωColoca 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 próximo

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 a memoria genera un page fault

Effective Access Time (EAT) EAT = (1 – p) x memory access + p x (page\_fault\_overhead + [swap\_page\_out] + swap\_page\_in + restart\_overhead

Podría ocurrir que no haya marcos disponibles, con lo cual habrá que descargar uno para lograr espacio para la nueva página entrante

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***

\*Direcciones de 32bits (20 bits numero de pagina – 12bits de desplazamiento)

Ejemplo

Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso = 2^20 (1.048.576)

El tamaño de cada página es de 4KB (2^12 )

El tamaño de cada PTE es de 4 bytes

Cantidad de PTEs que entran en un marco: 4KB/4B = 2^10

\*Tamaño de tabla de páginas

ω Cantidad de marcos necesarios para todas las PTEs de la tabla de páginas de un proceso = 2^20/2^10 = 2^10

ω Tamaño tabla de páginas del proceso: 2 10 \* 4bytes = 4MB por proceso

Direcciones de 64bits(52 bits numero de pagina – 12 desplazamiento)

Ejemplo

Cantidad de Page Table Entries (PTEs) máximas que puede tener un proceso = 2^52

El tamaño de cada página es de 4KB

El tamaño de cada PTE es de 4 bytes

ω Cantidad de PTEs que entran en un marco: 4KB/4B = 2^10

Tamaño de tabla de páginas

ω Cantidad de marcos necesarios para todas las PTEs de la tabla de páginas de un proceso = 2^52/2^10 = 2^42

ω Tamaño tabla de páginas del proceso = 2^42 \* 4bytes = 2^54

Más de 16.000GB por proceso!!!

***Tabla de 2 niveles (o más, multinivel)***

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 demasida 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

Tabla invertida: Hashing

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

Por ejemplo, si el espacio de direcciones es de 264 bytes, con páginas de 4 KB, necesitamos una tabla de páginas con 252 entradas

Si cada entrada es de 8 bytes, la tabla es de más de 30 millones de Gigabyes (30 PB)

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 refiera al espacio físico de la RAM, en vez del espacio de direcciones virtuales de un proceso

Usada en PowerPC, UltraSPARC, y IA-64

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 solucinar 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

ωSe busca indexadamente por número de página virtual

ωSi está presente en tabla, se extrae el marco de página y sus protecciones

ωSi no está presente en tabla, corresponde a un fallo de página

La forma de organizarla depende del HW subyacente

Tamaño de la Pagina

Pequeño

Menor Fragmentación Interna - Más paginas requeridas por proceso -Tablas de páginas más grandes - Más paginas pueden residir en memoria

Grande

Mayor Fragmentación interna - La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente - Mas rápido mover páginas hacia la memoria principal.

Relación con la E/S

Vel. De transferencia: 2 Mb/s

Latencia: 8 ms

Búsqueda: 20 ms

Pagina de 512 bytes

–1 pagina → total: 28,2 ms

– Solo 0,2 ms de transferencia (1%)

–2 paginas → 56,4 ms

Pagina de 1024 bytes

– total: 28,4 ms

– Solo 0,4 ms de transferencia

Translation Lookaside Buffer

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 paginas

Uno para obtener los datos

Para solucionar este problema, una memoria cache de alta velocidad es usada para almacenar entradas de páginas

-TLB

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

Dada una dirección virtual, el procesador examina la TLB

Si la entrada de la tabla de páginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física

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.

Se controla si la página está en la memoria ⎫Si no está, se genera un Page Fault

La TLB es actualizada para incluir la nueva entrada

El cambio de contexto genera la invalidación de las entradas de la TLB ⎝ Analizar. ¿Qué ocurre si el Quantum en Round Robin es chico?¿y al contrario?

Asignacion de marcos

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

Tamaño del Conjunto Residente

Asignación Dinámica

El número de marcos para cada proceso varía

Asignación Fija

Número fijo de marcos para cada proceso

Asignación equitativa – Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso

Asignación Proporcional: Se asigna acorde al tamaño del proceso.

Reemplazo de paginas

Qué sucede si ocurre un fallo de página y todos los marcos están ocupados [] “Se debe seleccionar una página víctima”

¿Cuál sería Reemplazo Optimo?

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 pagefaults de cada proceso

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

Algoritmos de Reemplazo

OPTIMO: 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

Thrashing (hiperpaginación)

Concepto: decimos que un sistema está en thrashing cuando pasa más tiempo paginando 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 scheduler long term aumenta el grado de multiprogramación.

2) El nuevo proceso inicia nuevos pagefaults, 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)

El 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).

El modelo de working set

Se basa en el modelo de localidad.

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

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

La 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 del thrashing por PFF

La técnica PFF (Page Fault Frecuency 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: Frecuencia de page faults

PFF alta: Se necesitan más frames

PFF baja: Los procesos tienen frames asignados que le sobran

Esquema de PFF

Establecer tasa de PF aceptable

Si la tasa actual es baja, el proceso puede perder frames

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 frames a mas 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 mas.

Se puede llegar a suspender un proceso si no hay más frames libres y todos los procesos estan 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

Ejemplos: En Linux : Proceso “kswapd” - En Windows: Proceso “system”

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

Cualquier modificación a la dirección de memoria es una modificación indirecta al archivo

El contenido de la pagina 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

Sobre el Área utilizada

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 esta en memoria, sino en swap, como podríamos saber en que parte del área de intercambio está?

Rta: El PTE de dicha pagina tiene el bit V=0 y todos los demás bits sin usar!

En linux

Permite definir un número predefinido de áreas de Swap

swap\_info es un arreglo que contiene estas estructuras

<linux/swap.h>

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

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

En número de área

El desplazamiento en el área (24 bits, lo que limita el tamaño máximo del área a 64 Gb)

*Subsistema de 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)

Aspectos de los dispositivos de I/O

Unidad de Transferencia

Dispositivos por bloques (discos):

ωOperaciones: Read, Write, Seek

Dispositivos por Caracter (keyboards, mouse, serial ports)

ωOperaciones: get, put

Formas de Acceso

Secuencial o Aleatorio

Tipo de acceso

• Acceso Compartido: Disco Rígido

• Acceso Exclusivo: Impresora

Tipo de acceso:

• Read only: CDROM

• Write only: Pantalla

• Read/Write: Disco

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

Interfaz Uniforme

Eficiencia

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

El uso de la multi-programación 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 ωFácil de usar y entender

ωNo es suficiente bajo algunas necesidades

No Bloqueante: El requerimiento de I/O retorna en cuanto es posible

ω Ejemplo: Interfaz de usuario que recibe input desde el teclado/mouse y se muestra screen.

ω Ejemplo: Aplicación de video que lee frames desde un archivo mientras va mostrandolo en pantalla.

Diseño – Software capa de usuario

Librerías de funciones

– Permiten acceso a SysCalls – Implementan servicios que no dependen del Kernel

Procesos de apoyo

– Demonio de Impresión (spooling)

Diseño – Software independiente SO

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.

Diseño – Controladores (Drivers)

Contienen el código dependiente del dispositivo

Manejan un tipo dispositivo

Traducen los requerimientos abstractos 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

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

Ejemplo en Linux

Linux distingue 3 tipos de dispositivos

Carácter: I/O programada o por interrupciones

Bloque: DMA

Red: Ports de comunicaciones

Los Drivers se implementan como módulos

Se cargan dinámicamente

Debe tener al menos estas operaciones:

init\_module: Para instalarlo

cleanup\_module: Para desinstalarlo.

Operaciones que debe contener para I/O

open: abre el dispositivo

release: cerrar el dispositivo

read: leer bytes del dispositivo

write: escribir bytes en el dispositivo

ioctl: orden de control sobre el dispositivo

Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo

Acceso al hardware

Funciones para acceso a los puertos de I/O

Leen o Escriben un byte en el puerto de E/S indicado

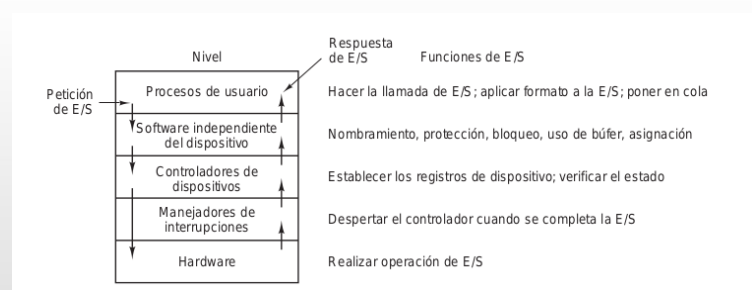
Diseño – Gestor de interrupciones

Atiende todas las interrupciones del HW

Deriva al driver correspondiente según interrupción

Resguarda información

Independiente del Driver

Ciclo de atención de un Requerimiento

Desde el Requerimiento de I/O hasta el Hardware

Consideremos la lectura sobre un archivo en un disco:

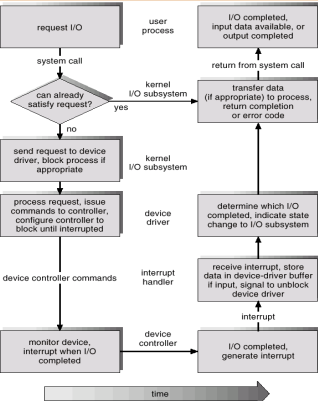
Determinar el dispositivo que almacena los datos

• Traducir el nombre del archivo en la representación del dispositivo.

Traducir requerimiento abstracto en bloques de disco (Filesystem)

Realizar la lectura física de los datos (bloques) en la memoria

Marcar los datos como disponibles al proceso que realizo el requerimiento

• Desbloquearlo

Retornar el control al proceso

Ciclo de vida de un requerimiento de I/O

Performance

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

Utiliza mucho la CPU para ejecutar los drivers y el codigo del subsistema de I/O

Provoca Context switches ante las interrupciones y bloqueos de los procesos

Utiliza el bus de mem. en copia de datos:

• Aplicaciones (espacio usuario) – Kernel • Kernel (memoria fisica) - Controladora

Mejorar la Performance

Reducir el número de context switches

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 mas inteligentes

– Polling, si se minimiza la espera activa.

Utilizar DM

Variedad en los dispositivos de I/O

Legible para el usuario

Usados para comunicarse con el usuario

ω Impresoras, Terminales: Pantalla, Teclado, Mouse

Legible para la máquina

Utilizados para comunicarse con los componentes electrónicos

ω Discos, Cintas, Sensores, etc.

Comunicación

Usados para comunicarse con dispositivos remotos

ω Líneas Digitales, Modems, Interfaces de red, etc.

Problemas que surgen

Amplia Variedad

Manejan diferentes cantidad de datos

En Velocidades Diferentes

En Formatos Diferentes

La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM

Hardware y software involucrado

Buses

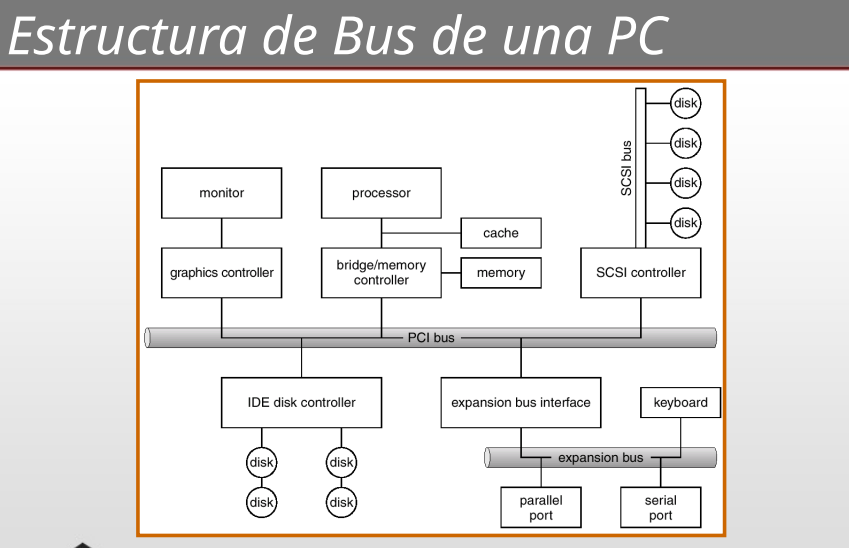
Controladores

Dispositivos

Puertos de E/S – Registros

Drivers

Comunicación con controlador del dispositivo: I/O Programada, Interrupciones, DMA



Comunicación: CPU - Controladora

¿Cómo puede la CPU ejecutar comandos o enviar/recibir datos de una controladora de un dispositivo?

La controladora tiene uno o mas registros: • Registros para señales de control • Registros para datos

La CPU se comunica con la controladora escribiendo y leyendo en dichos registros

Comandos de I/O

CPU emite direcciones

Para identificar el dispositivo

CPU emite comandos

Control – Que hacer?

ωEj. Girar el disco

Test – Controlar el estado

ωEj. power? Error?

Read/Write

ωTransferir información desde/hacia el dispositivo

Mapeo de E/S y E/S aislada