El sistema operativo se encarga de abstraer al software del hardware

- Comodidad: Hacer más cómodo el uso de la computadora
- Eficiencia: Uso más eficiente en los recursos de un sistema
- Evolución: Permitir la introducción de nuevas funciones al sistema sin interferir con funciones anteriores, soportar nuevos tipos de HW, brindar nuevos servicios, y ofrecer mejoras y alternativas a problemas existentes como la planificación del uso del procesador y del manejo de la memoria

El SO administra el uso del procesador, de la memoria y los dispositivos. Es el encargado de detectar errores de hardware y software y dar una respuesta a estos. También monitorea el funcionamiento y recoge estadísticas de uso.

El kernel es una porción de código que se aloja en memoria principal que se encarga de la administración de los recursos, manejo de memoria, administración de procesos y la comunicación y concurrencia de los mismos, además de gestionar el hardware

El procesador posee registros visibles para el usuario (datos, direcciones, stack pointer, etc.) que pueden ser usados en aplicaciones, y registros de control y estado (PC, IR, PSW) que sirven para el control del procesador, estos son usados por las subrutinas privilegiadas del SO.

La ejecución de una instrucción consiste en leerla de memoria y ejecutarla:

1. Se lee el PC para saber dónde buscar la siguiente instrucción y se la guarda en IR, se incrementa PC y se ejecuta la instrucción.

Una instrucción puede ser

- Procesador-memoria
- Procesador-I/O
- Procesamiento de Datos
- Control

Interrupciones:

Las interrupciones sirven para suspender el orden normal de ejecución de las instrucciones

- Program (generada por el resultado de la ejecución de una instrucción EJ overflow)
- Timer (Generada por un timer del procesador, para hacer X funciones cada X tiempo)
- I/0 (Generada por un controlador de I/O que indica que se completó una acción o hubo un error.)
- Hardware Failure (Generada por una falla como error de paridad en memoria o baja corriente.)

El interrupt handler es un programa/rutina que atiende determinada instrucción, suele ser parte del SO.

Busco una instrucción desde PC y la cargo en IR, aumento el PC, desactivo las interrupciones y ejecuto la instrucción, activo las interrupciones y si hubo alguna inicio el interrupt handler, pusheo el PSW y el PC a

la pila de control, guardo el contexto del proceso actual, cargo un nuevo PC según la interrupción, ejecuto la rutina de interrupción, restauro todos los valores antes de la interrupción y luego vuelvo a buscar otra instrucción.

Para manejar múltiples interrupciones se puede deshabilitar las interrupciones mientras una instrucción/interrupción está siendo atendida, o darle prioridades a las interrupciones y que se permita interrumpirse entre ellas.

Proceso: programa en ejecución, tiene 3 partes, código, datos y stack (datos temporales, parámetros, direcciones de retorno.)

Los procesos tienen dos pilas, una para el modo usuario y otra para el modo kernel

Atributos de un proceso: ID del proceso, ID del padre, ID de quien lo "disparó", ID de grupo que lo "disparó"

PCB: hay uno por proceso, contiene estado, PC y registros de la CPU del proceso, es lo primero que se crea y lo último que se borra al iniciarse un proceso.

Cada proceso tiene asignado un espacio de direcciones que ocupa, no incluye al PCB, en modo usuario un proceso solo puede acceder a su espacio de direcciones, en modo kernel además se pueden acceder a estructuras del kernel y al espacio de direcciones de otros procesos.

Contexto de un proceso: Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente. Son parte del contexto, los registros de cpu, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.

Context Switch: se produce cuando la CPU cambia de un proceso a otro, se debe guardar la info del proceso saliente y cargar la del entrante, este cambio deja oscilando a la CPU

Existen colas de procesos donde se encuentran todos los procesos del sistema, de procesos listos que son aquellos en memoria principal con estado listo y a la espera de ejecutarse y colas de dispositivos, que incluyen a los procesos esperando por un dispositivo de i/o

El espacio de direcciones de un proceso no incluye su PCB, en modo usuario solo puedo acceder al espacio de direcciones de mi proceso, en modo kernel al de otros, esta comprobación se realiza durante la ejecución, ya que el SO no puede anticipar todas las referencias a memoria que un proceso pueda realizar.

El S.O es un conjunto de módulos de software que se ejecuta como cualquier otro proceso, existen distintos enfoques para los SO's.

Enfoque 1: El Kernel no como un proceso:

Se ejecuta fuera de todo proceso.

Cuando se da una interrupción o system Call, el contexto del proceso se salva y se pasa el control al Kernel, que tiene su propio espacio de memoria y stack, cuando finaliza su actividad devuelve el control al proceso que corresponda, en este enfoque el Kernel no es un proceso.

Enfoque 2 : El Kernel "dentro" del Proceso:

El Software del kernel se ejecuta en el mismo contexto que los procesos de usuario (dentro)

El Kernel se puede ver como una colección de rutinas que el proceso utiliza.

Dentro de un proceso se puede ejecutar un programa de usuario y módulos del SO.

Cada proceso tiene su propio Stack para el Kernel.

El proceso se ejecuta en modo usuario y modo kernel.

La interrupción o una System Call es atendida en el contexto del proceso que se encontraba en ejecución. Pero en modo Kernel.

Estados de un proceso:

- Nuevo (new)
    - Un usuario "dispara" el proceso, que es creado por su proceso padre, en este estado se crean las estructuras asociadas al proceso y este queda en la cola de procesos en espera de ser cargado en memoria.
- Listo para ejecutar (ready)
    - El proceso se encuentra en la cola de procesos listos, El scheduler de largo plazo elige el proceso para cargarlo en memoria, el proceso solo necesita que se le asigne CPU.
- Ejecutándose (running)
    - El short term sheduler lo eligió para asignarle CPU, y la tendrá hasta que se termine su tiempo asignado (quantum) o necesite que se produzca un evento como una I/O
- En espera (waiting)
    - El proceso necesita que se cumpla un evento para continuar, sigue en memoria pero no tiene la CPU, cuando se cumpla el evento pasará al estado listo.
- Terminado (terminated)

- New-Ready: Por elección del scheduler de largo plazo (carga en memoria)
- Ready-Running: Por elección del scheduler de corto plazo (asignación de CPU)
- Running-Waiting: el proceso "se pone a dormir", esperando por un evento.
- Waiting-Ready: Terminó la espera y compite nuevamente por la CPU.
- running-ready: Se termina el quantum del proceso y pasa al estado ready para competir por la CPU

Para controlar la ejecución de procesos se tienen muchas colas, para formar las colas se enlazan las PCBs de los procesos, hay colas de procesos, de procesos listos para ejecutarse (ready), de dispositivos, etc.

Existen módulos que controlan la planificación, como por ejemplo el Scheduler de long term, short term y médium term. También existen el dispatcher (que hace el cambio de contexto, despacha el proceso elegido por el short term) y el loader (Carga en memoria el proceso elegido por el long term), que pueden no existir como módulos independientes pero su función debe ser cumplida.

Long term Scheduler: Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria. ⌡ Puede no existir este scheduler y absorber esta tarea el de short term.

Medium Term Scheduler (swapping): Si es necesario, reduce el grado de multiprogramación, Saca temporariamente de memoria los procesos que sea necesario para mantener el equilibrio del sistema. , Términos asociados: swap out (sacar de memoria), swap in (volver a memoria).

Short Term Scheduler: Decide a cuál de los procesos en la cola de listos se elige para que use la CPU.

Es necesario determinar cuál de todos los procesos listos se selecciona para ejecutarse, para eso existen los algoritmos de planificación, estos son parametrizados ya que su comportamiento debe poder ser modificado por el usuario.

En los algoritmos Apropiativos (preemtive) existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU

En los algoritmos No Apropiativo (nonpreemptive) los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU

– Se bloquea por E/S, finaliza, etc.

– No hay decisiones de planificación durante las interrupciones de reloj

Según el ambiente se puede necesitar algoritmos de planificación con diferentes metas, por eso se separan en categorías como por ejemplo:

Procesos Batch: No existen usuarios que esperen una respuesta en una terminal y se pueden utilizar algoritmos no apropiativos.

Metas propias de este tipo de algoritmos:

- Rendimiento: Maximizar el número de trabajos por hora
- Tiempo de Retorno: Minimizar los tiempos entre el comienzo y la finalización
- Uso de la CPU: Mantener la CPU ocupada la mayor cantidad de tiempo posible

Ejemplos de Algoritmos: FCFS – First Come First Served SJF – Shortest Job First

Procesos interactivos:

- No solo interacción con los usuarios
- Un servidor, necesita de varios procesos para dar respuesta a diferentes requerimientos
- Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU

Metas propias de este tipo de algoritmos:

- Tiempo de Respuesta: Responder a peticiones con rapidez
- Proporcionalidad: Cumplir con expectativas de los usuarios ω Si el usuario le pone STOP al reproductor de música, que la música deje de ser reproducida en un tiempo considerablemente corto.

Ejemplos de Algoritmos: Round Robin, Prioridades, Colas Multinivel, SRTF

Un proceso siempre es creado por otro proceso, este puede tener varios hijos formándose así un árbol de procesos.

Creación de un proceso:

1. Crear la PCB
2. Asignar PID (Process IDentification) único
3. Asignarle memoria para regiones Stack, Text y Datos
4. Crear estructuras de datos asociadas – Fork (copiar el contexto, regiones de datos, text y stack)

Un proceso padre puede funcionar concurrentemente con su hijo o esperar a que su hijo termine para continuar con su ejecución.

Creación: En UNIX: system call fork() crea nuevo proceso ⎫ system call execve(), usada después del fork, carga un nuevo programa en el espacio de direcciones.

En Windows: ⎫ system call CreateProcess() crea un nuevo proceso y carga el programa para ejecución.

Un proceso padre puede matar a su hijo, y un proceso luego de un exit retorna el control al sistema operativo.

Independiente: el proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato.

Cooperativo: afecta o es afectado por la ejecución de otros procesos en el sistema.

Un proceso cooperativo sirve para compartir información (por ejemplo, un archivo) ⎭ Para acelerar el cómputo (separar una tarea en sub-tareas que cooperan ejecutándose paralelamente) ⎭ Para planificar tareas de manera tal que se puedan ejecutar en paralelo.

Memoria:

Reubicación: El programador no conoce donde está ubicado el programa en ejecución, este puede variar durante su ejecución, debe existir un traductor para saber la dirección actual del proceso.

Compartición: consiste en permitir que varios procesos puedan acceder al mismo espacio de memoria por ejemplo para acceder a una librería, evitando las copias innecesarias de instrucciones.

El espacio de direcciones de un proceso varía dependiendo de la arquitectura del PC (32/64bits), este espacio es independiente de la ubicación real del proceso.

Direcciones lógicas: referencia a una localidad en memoria independiente de la asignación actual de los datos en la memoria, estas se traducen a direcciones físicas.

Direcciones físicas: Es la dirección absoluta en la memoria principal.

Para convertir una dirección lógica en física se utilizan dos registros auxiliares, Registro base que es la dirección de comienzo del proceso, y registro límite que es la dirección final del proceso. Su valor se fija cuando el proceso es cargado a memoria, con la dirección lógica y el registro base se genera una dirección física, esta se compara con el registro limite y si es invalida se genera una interrupción al SO.

Las direcciones en los programas fuentes son simbólicas, el compilador las convierte en direcciones reubicables, y el linkeditor en direcciones absolutas, este proceso se conoce como binding de direcciones.

Al hacer Swap out y Swap in de un proceso, si trabajamos con direcciones físicas se debe asignar al mismo espacio de memoria que ocupaba antes, en cambio con las lógicas se puede cargar en cualquier lado.

El mapeo de direcciones lógicas y físicas es realizado por el dispositivo de hardware MMU (Memory Managment Unit), se encuentra en el cpu y es reprogramado por el kernel.

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.

Esquemas de asignación de Memoria:

Única partición: Los procesos ocupan una única partición de memoria y la protección se da por un límite y un registro de realocación.

Esquemas con múltiples particiones: La memoria se divide en varias particiones.

Los procesos se colocan en las particiones según su tamaño.

Las particiones pueden ser fijas o dinámicas.

Particiones fijas (mismo tamaño): un proceso siempre se puede colocar en cualquier partición libre. Con esta técnica el tamaño de los procesos se limita al de las particiones, y los procesos pequeños desperdician toda una partición para su ejecución.

Particiones Fijas (de distinto tamaño): los tamaños de las particiones son fijos pero varian, esto soluciona en cierta medida los problemas anteriores pero aumentan la complejidad a la hora de seleccionar una partición ideal para un proceso, para esto el algoritmo de ubicación puede tener una cola de procesos para cada partición, o una cola de procesos para todas las particiones.

Particiones dinámicas: El número y tamaño de las particiones varia, los procesos se colocan en particiones de tamaño exactamente igual al suyo (generados dinamicamente). Cada vez que entra y sale un proceso se genera huecos en la memoria, en los que eventualmente un proceso no podría entrar, pero si entraría si unimos todos los huecos (compactación para evitar fragmentación externa)

Algoritmos de ubicación para esquemas de particiones dinámica: ⎫First Fit aloja el proceso en el primer lugar libre, es la más rápida y eficiente técnica⎫Best Fit busca el hueco más parecido al tamaño del proceso, tarda mucho y genera fragmentación externa ⎫Next Fit, funciona como First FIt pero en vez de empezar a fijarse desde el principio se fija como si fuera una lista circular, alocando más rápido los huecos grandes del final pero produciendo resultados ligeramente peores que First Fit.

Paginación: La memoria se divide lógicamente en pequeños trozos de igual tamaño, Marcos.

El espacio de direcciones de cada proceso es dividido en trozos del tamaño de los marcos, Paginas.

El SO mantiene una taba de páginas por cada proceso que Indica el marco donde está situada cada página, la dirección lógica consiste en un número de página y un desplazamiento dentro de la misma.

Segmentación: soporta el punto de vista de un usuario.

Un programa es una colección de segmento. Un segmento es una unidad lógica como procedimientos, funciones, etc.

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 Segmento: Permite mapear la dirección lógica en física. Cada entrada contiene:

- Base: Dirección física de comienzo del segmento
- 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

La paginación es transparente al programador, Elimina Fragmentación externa. La Segmentación es visible al programador y facilita modularidad, permite estructuras de datos grandes y da mejor soporte a la compartición y protección.

El conjunto residente o working set es la porción del espacio de direcciones del proceso que se encuentra en memoria. El HW se encarga de detectar una porción del proceso que no está en el working set.

Esto permite que más procesos estén en memoria al mismo tiempo para evitar que la CPU este ociosa y permite evitar limitar el tamaño de un proceso.

Para poder utilizar Memoria Virtual el HW debe soportar paginación por demanda y/o segmentación., se necesita un dispositivo de memoria secundaria (disco) y el SO debe soportar el movimiento de páginas/segmentos entre MP y MS.

Memoria virtual con paginación:

Cada proceso tiene su tabla de páginas, donde cada entrada hace referencia al marco donde está la página en memoria principal. También tienen bits de control, el Bit V indica si la página esta en memoria, el Bit M indica si la página fue modificada (para saber que deben reflejarse los cambios en memoria secundaria.)

Page fault: ocurre cuando el proceso intenta usar una dirección que está en una página que no está en el wortking set (Bit V=0).

El HW lo detecta y genera un trap al SO.

El SO coloca el proceso en espera mientras gestiona la carga de la p{agina necesaria, durante esta operación de E/S, otros procesos toma el control de la CPU, la carga de la página consiste en buscar un marco vacío y copiar en el marco la página necesaria, una vez realizado se avisa con una interrupción, entonces el SO actualiza la tabla de páginas en el proceso poniendo el bit V de la página en 1 y la dirección base del marco donde colocó la página, se pone en ready el proceso que generó el fallo de página.

Una tabla de páginas puede ser de un nivel, de múltiples niveles o tabla invertida (hashing), esta elección depende del HW. La tabla invertida se usa en arquitecturas donde el espacio de direcciones es muy grande, se tiene una entrada por cada marco y hay una sola tabla para todo el sistema, el número de página se transforma en un valor de HASH, y el HASH se usa como índice para encontrar el marco asociado

Tamaño de la Pagina ⌋ Pequeño ⌉ Menor Fragmentación Interna. ⌉ Más paginas requeridas por proceso ⌊ Tablas de páginas mas grandes. ⌉ Más paginas pueden residir en memoria ⌋ Grande ⌉ Mayor Fragmentación interna ⌉ La memoria secundaria esta diseñada para transferir grandes bloques de datos más eficientemente ⌊ Mas rápido mover páginas hacia la memoria principal.

Cuando se da un pagefault y no se tienen marcos vacíos, se busca una página víctima, lo ideal es que la victima no sea referenciada en un futuro cercano, esto se logra prediciendo el futuro a través de la lectura de las acciones pasadas.

Reemplazo Global ⎫El fallo de página de un proceso puede reemplazar la página de cualquier proceso. ⎫Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él. ⎫El SO no controla la tasa de page-faults de cada proceso. ⎫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.

OPTIMO ⎭FIFO ⎭LRU (Least Recently Used) ⎭2da. Chance ⎭NRU (Non Recently Used).

Thrashing: es cuando un Sistema pasa más tiempo paginando que ejecutando procesos reduciendo así la performance.

Para evitar el thrashing se debe seleccionar un tamaño de working set optimo, si es muy chico no cubrirá la localidad, si es grande puede tomar varias localidades.

Para prevenir el thrashing el SO monitorea cada proceso asignándole tantos frames como la medida del working set del proceso requiera, si sobran frames entonces ejecuta otro proceso. Si la tasa de frames que necesita un proceso aumenta por encima de lo tolerable debe suspenderse otro proceso para cubrir las necesidades del proceso actual, de esta manera se aumenta el grado de multiprogramación optimizando el uso de la cpu.

⎭ 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

La copia en escritura (Copy-on-Write, COW) permite a los procesos padre e hijo compartir inicialmente las mismas páginas de memoria ⎫Si uno de ellos modifica una página compartida la página es copiada ⎭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: ⎫ 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. ⎫ 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.

⎭ 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: ⎫ Limpia las páginas modificadas sincronizándolas con el swap ⎫ Reduce el tiempo de swap posteriormente ya que las páginas están “limpias” ⎫ Puede sincronizar varias páginas contiguas reduciendo el tiempo total de transferencia ⎫ Mantener el número de páginas libres en el sistema a un cierto número ⎫ No liberarlas del todo hasta que haga falta realmente Demonio de Paginación - Ejemplos ⎭ En Linux ⎫ Proceso “kswapd” ⎭ En Windows ⎫ Proceso “system”

I/O:

Existen varios dispositivos de I/O,

Legible para el usuario, como las impresoras, pantallas, teclados y mouse.

Legible para la máquina, como los discos, las cintas magnéticas, etc.

Y de Comunicación, como las líneas digitales, los módems, etc.

Como estos son muy variados y trabajan a velocidades diferentes con cantidades y formatos de datos distintos (y en gran mayoría más lentamente que la cpu y la RAM) se busca una manera de abstraerse de su funcionamiento específico. Para eso muchos dispositivos de HW y módulos de SW interactúan para resolver las operaciones de I/O. (Buses, Controladores, drivers, puertos, interrupciones, etc.) se sigue un estándar de resolución de I/o que permite que distintos dispositivos puedan comunicarse correctamente con todas las computado ras, el sistema operativo se comunica con el controlador y el controlador con el dispositivo.

Comunicación CPU-controladora: La controladora tiene registros para señales de control y para datos, la CPU se comunica con la controladora del dispositivo de I/O escribiendo y leyendo en dichos registros.

Comandos de I/O: Cpu emite direcciones para identificar el dispositivo.

Cpu emite comandos de control, para consultar estados y para realizar operaciones de I/O.

E/s programada es sincrónica realiza polling, deja la cpu ociosa, ineficiente, la que es por interrupciones no es sincrónica, el cpu atiende otros procesos hasta que el controlador del dispositivo avisa con una interrupción que ya está lista la operación de I/O, en ese momento el proceso que se está atendiendo actualmente se bloquea y se restaura la ejecución del proceso que hizo la llamada de I/O.

Mapeo de la I/O: Los dispositivos y memoria comparten el espacio de direcciones, los registros del controlador están en la memoria, como se comparte la memoria para los registros de control no hay instrucciones especiales, la I/O se maneja como un acceso a memoria común y corriente

La técnica Aislada separa el espacio de direcciones, entonces si son necesarias instrucciones especiales y puertos para controlar las operaciones de I/O

La DMA es un mecanismo de I/O que lee una GRAN cantidad de información, el SO genera una interrupción de lectura al DMA, el DMA se lo manda al controlador, el controlador envía los bytes a la memoria directamente sin utilizar la CPU, cuando termina la transferencia se da una interrupción para avisar que terminó la operación.

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

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

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 archivo en el dispositivo. ⎫Lectura física de los datos en la memoria ⎫Marcar los datos como disponibles al proceso que realizo el requerimiento mediante una interrupción • marcarlo como ready nuevamente ⎫Retornar el control al proceso

Driver de dispositivo: 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 de los dispositivos están asociadas a una función del 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

Para mejorar la performance:

I/O es uno de los factores que mas afectan a la performance del sistema: ⌠Utiliza mucho la CPU para executar 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

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 DMA

FileSystems:

Se encarga de abstraer al programador y al usuario del manejo de archivos, proveyendo acciones como Crear ⌠Borrar ⌠Buscar ⌠Copiar ⌠Leer ⌠Escribir ⌠Etc.

Un archive tiene los siguientes atributos,⌡ Nombre ⌡ Identificador ⌡ Tipo ⌡ Localización ⌡ Tamaño ⌡ Protección, Seguridad y Monitoreo ⌠Owner, Permisos, Password ⌠Momento en que el usuario lo modifico, creo, accedio por ultima vez.

Derechos de acceso: Execution ⌠El usuario puede ejecutar ⌡Reading ⌠El usuario puede leer el archivo, ⌡Appending ⌠El usuario puede agregar datos pero no modificar o borrar el contenido del archivo Updating ⌠El usuario puede modificar, borrar y agregar datos. Incluye la creación de archivos, sobreescribirlo y remover datos ⌡Changing protection ⌠El usuario puede modificar los derechos de acceso ⌡Deletion ⌠El usuario puede borrar el archivo

Owners (propietarios) ⌠Tiene todos los derechos ⌠Pueden dar derechos a otros usuarios. Se determinan clases: ωUsuario específico ωGrupos de usuarios ωTodos (archivos públicos)

Un directorio es un archivo que contiene información de otros archivos, sirve para localizar archivos rápidamente, permitir el mismo nombre de dos archivos distintos y agrupar lógicamente archivos.

La memoria virtual con paginación por demanda nace de la realización de que ya que no siempre se necesita la totalidad de un proceso en memoria para ejecutarlo, si solo cargamos en memoria las partes de un proceso que si son necesarias actualmente estaríamos dejando espacios para que más procesos hagan lo mismo, y cuanto mayor sea la cantidad de procesos ubicados en memoria al mismo tiempo mayor es el grado de multiprogramación.

El proceso que se sigue es el siguiente:

1. Se intenta leer la página requerida
2. Si la página requerida ya está en memoria, simplemente se lee.
3. Si no está en memoria se intenta cargar la página.
4. Cuando la página sea cargada, se reintenta la instrucción.

El comprobar si una página que necesita ser referenciada se encuentra en memoria principal es realizado por el HW.

Al buscar una página, si esta no está en memoria, necesitará ser cargada. A este proceso se le llama **fallo de página**.
Al iniciar la ejecución de un programa, la tabla de páginas cuenta con todas sus entradas inválidas por lo cual el paginador fallará hasta tener lo necesario para iniciar el programa. Luego de esta carga inicial se comprobará si la siguiente página a utilizar ya está en memoria, en caso de que la página se encuentre, ésta es leída, pero cuando la página no es encontrada tenemos dos posibilidades:

- Si existe un frame libre, se carga y se lee.
- Si no tenemos frames libres, se intercambia la página de algún frame por la información a utilizar, esta página victima puede tomarse del espacio de memoria del mismo proceso si se utiliza técnicas de reemplazo local, o de cualquier proceso si se utiliza una técnica de reemplazo global.

El criterio utilizado para seleccionar qué página será intercambiada varía dependiendo de la implementación del sistema (existen varios algoritmos como LRU). Muchos de los problemas que presenta el sistema de paginación por demanda son debidos a los fallos de página y principalmente a saber cuál es la página más conveniente para intercambiar. Esto se debe a que no podemos saber cuáles páginas serán utilizadas prontamente y cuales no se volverán a utilizar, se busca analizar el comportamiento pasado para predecir que página no será referenciada en el futuro cercano.

El encargado de administrar las paginas es el S.O, este tiene una tabla donde mapea las páginas con los marcos en las cuales la página se encuentra, esta tabla puede ser de un nivel, multinivel (una tabla que hace referencia a otras tablas) o puede ser una tabla invertida que se accede mediante un índice obtenido con una función de Hash, El método a utilizar para la tabla dependerá del HW.

Al utilizar Memoria virtual es importante evitar el Thrashing o hiperpaginación, que consiste en pasar más tiempo resolviendo PageFaults que procesando instrucciones de procesos, el grado de hiperpaginación es detectado por el S.O, para evitar el Thrashing debemos elegir un tamaño de página adecuado, ni muy grande como para no desperdiciar espacio ni muy chico. , un tamaño de Working Set

adecuado que permita suficientes páginas de nuestro proceso pero no muy grande como para no perder el sentido de la paginación por demanda, y que el S.O administre correctamente la frecuencia de fallos de página de los procesos.

Ventajas

- Al no cargar las páginas que no son utilizadas **ahorra memoria** para otras aplicaciones.
- Al mejorar el uso de la memoria, mejora el grado de **multiprogramación**.
- **Carga inicial más rápida** ya que solo lee del disco lo que se utilizará.
- Capacidad de hacer funcionar programas que ocupan más memoria que la poseída.
- **COW (Copia en escritura)**: Permite utilizar las mismas páginas para dos procesos (padre-hijo) hasta que uno de estos las modifique.

Una SystemCall Es la forma en que los programas de usuario acceden a los servicios del SO, consiste de los siguientes pasos:

1. El proceso de Usuario llama a una Syscall.
2. Se guarda en el stack del usuario los parámetros de la syscall y se lanza una interrupción.
3. Se cambia a modo Kernel.
4. Se guarda el PC (Program Counter: indica la dirección de la siguiente instrucción a ejecutar) y el PSW (Proram Status Word: indica el estado del programa actual) en la pila del usuario.
5. Se carga en el PC la dirección de la subrutina que atiende la interrupción.
6. Se sacan los parámetros de la SysCall de la pila del usuario para que ver que llamada tiene que atender.
7. Se guarda la dirección de la pila del usuario en el PCB del proceso y se pasa a utilizar la pila del kernel.
8. Se guardan los parámetros de la Syscall en la pila del kernel y se ejecuta la SysCall.
9. Si la SysCall bloquea el proceso se ejecuta el short term scheduler para que un nuevo proceso tome la cpu y esta no quede ociosa.
    a. Se da el context switch
    b. se guarda en la pila la dirección que el HW dejo previo a que el proceso seleccionado sea suspendido.
10. Más allá de que se haya bloqueado o no el proceso, quien ahora tengo el control de la CPU va a cambiar el modo a User Mode.
11. Ejecutar la sentencia RET para obtener de la pila la RET la dirección de retorno a ejecutar.
12. Se obtienen de la pila el PSW y PC y se continúa la ejecución del proceso actual.

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 ⌡ File Allocation Table (FAT) es simple ⌉Sólo una entrada que incluye Bloque de inicio y longitud ⌡ El archivo puede ser leído con una única operación ⌡ Puede existir fragmentación externa ⌉Compactación

Problemas de la técnica ⌉Encontrar bloques libres continuos en el disco ⌉Incremento del tamaño de un archivo

Formas de Asignación - Encadenada ⌡Asignación en base a bloques individuales ⌡Cada bloque tiene un puntero al próximo bloque del archivo ⌡File allocation table ⌉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 contiguous

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

Formas de Asignación - Indexada ⌡Asignación en base a bloques individuales ⌡No se produce Fragmentación Externa ⌡El acceso “random” a un archivo es eficiente ⌡File Allocation Table ⌉Única entrada con la dirección del bloque de índices (index node / i-node)

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 bloques libres ⌞ Hay que realizar varias operaciones de E/S para obtener un grupo libre. ⌡Problemas con la pérdida de un enlace ⌡Dificil 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.

Recuento ⌡Variante de Indexación ⌡Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación contigua). ⌡En lugar de tener N direcciones libres (índice) se tiene: ⌉La dirección del primer bloque libre ⌉Los N bloques libres contiguos que le siguen. (#bloque, N siguientes bloques libres)

Control sobre cuáles de los bloques de disco están disponibles.

Tabla 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 Eje: Disco 16 Gb con bloques de 512 bytes ⌡ 32 Mb.

System V UNIX

UNIX - Estructura del Volumen

- ・Boot Block: Código para bootear el S.O.
- ・Superblock: Atributos sobre el File System
- ・I-NODE Table: Tabla que contiene todos los I-NODOS
- ・I-NODO: Estructura de control que contiene la información clave de un archivo
- ・Data Blocks: Bloques de datos de los archivos

Primero tenes que describir la estructura de bloques del System v, osea boot block, superblock , tabla de inodos y bloques de datos. Despues te dice que detalles la creacion de un archivo, empezando por crear el i-nodo, almacenarlo en la tabla, modificacion de estructuras en su creacion como el superbloque, tabla de archivos y tabla de descriptores de archivos. Después tenés que saber la forma de acceder a ese path o de almacenar un archivo en un determinado directorio entrando con los i-nodos de los archivos.