Cosas que hay que investigar:

- Vector de interrupciones
- PC (program counter)
- PCB

ISO RESUMEN: estudiar buffer cache

Sistemas de archivos. Si quisiera BORRAR un archivo por ejemplo /home/usuario/final.txt, que estructuras se verian afectadas? Como?

Hay un apunte dando vueltas.... Pero, básicamente se ven afectadas las listas de i-nodos libres, la lista de i-nodos ocupados, lista de bloques ocupados y lista de bloques libres... junto con el Superblock... También tenes que especificar como es el proceso, como se recorre la lista,

PREGUNTAS Y RESPUESTAS

System call y procesos.

1) System calls: Definicion, explicar cómo pueden ser implementadas para ello tenga en cuenta: participación o apoyo de hw, modos de ejecución, stack (pilas)

2) System calls: Suponiendo que un proceso x ejecuta una system call que ocasiona que el proceso sea bloqueado por ejemplo write() , analice y describa que eventos, actividades, se suceden hasta que otro proceso Y

comienza su ejecución (tener en cuenta pila, notificaciones, dispacher, etc)

3) Describa las estructuras de datos y actividades necesarias para implementar el manejo de procesos dentro de un sistema operativo (ayuda: tenga en cuenta: estados de un proceso, estructuras necesarias para su manejo y actividades del sistema operativo durante la planificación de los procesos).

4) Un procesos x desea ejecutar la system call fork(). Describa lo que sucede (pasos) desde que la misma es ejecutada en el x hasta que la misma es completada y se le puede brindar el resultado al proceso. Indique cuales de las estrutucturas indicadas en el punto anterior son utilizadas y/o creadas así como el apoyo/participación del hardware y las tareas llevada a cabo por el sistema operativo.

5) Suponiendo que un proceso x utiliza una system call que ocasiona que el mismo sea bloqueado. Por ejemplo read(). Analice y describa que eventos, actividades, se suceden hasta que otro proceso Y (que se encontraba en estado de listo) comienza su ejecución (tener en cuenta las estructuras y eventos relacionados a planificación, apoyo de hw, modos de ejecución, stacks, modulos de planificación, etc) .

6) Se requiere implementar el concepto de “proceso” junto con el algoritmo de planificaciones de cpu round robin en un sistema operativo. ¿Cómo lo implementaría? Tenga en cuenta apoyo del hw que necesita y con que fin (interrupciones, modos de ejecución, etc) asi como las estructuras de datos que necesitará el mantenidas por el sistema operativo, asi como que tareas deberá realizar el mismo, implementación del quantum, tareas realizadas cada vez que un quatum se consume. Etc.

7) System calls:¿ por qué son necesarias? Por que no pueden ejecutars en modo usuario?

8) Se requiere ejecutar la instrucción que implica una lectura de uan fila de una nbase de datos que esta en disco. Considerando que el controlador no sabe de bas de datos… como se trasnforma esa instrucción en uan orden para qel controlador de disco? Cual seria ese comando? Load o store?

9) Describa estados de un proceso. Planificadoes (schedulers), su relación con las transiciones entre los estados enumerados.

MEMORIA

10) Memoria virtual con paginación por demanda: funcionamiento, estructuras utilizadas, participación del hw y actividades desarrolladas por el sistema operativo, fallo de paginas.

11) Hiperpaginacion: definición, formas de detectarlas, modelo de working set.

12) ¿Qué es memoria virtual? ¿Cómo implementaría memoria virutal con paginación por demanda? Describa el proceso de su funcionamiento identificando momentos en los que se suceden interrupciones o traps, cambios de estado de los procesos, cambios de modos de ejecución del procesador, tareas llevadas a cabo por el sistema operativo en la gestión de la memoria virtual y el apoyo requerido del hardware.

13) Paginación y segmentación. Como funcionan (tenga en cuenta : estructuras necesarias, participación del hw, etc). Cite ventajas y desventajas de un esquema respecto al otro.

14) Hiperpaginacion (trashing): Definición, formas de detectarla y tratarla. Cómo puede ser tratada? Relación con working set.

15) Memoria virtual con paginación por demanda: funcionamiento, estructuras utilizadas, participación del hw y actividades desarrolladas por el sistema operativo. Fallos de páginas. Definición.

16) Describa como se implementa memoria virtual utilizando segmentación por demanda. Tenga en cuenta estructuras utilizadas, participación del hw, tareas del sistema operativo. Ventajas y desventajas respecto a paginación por demanda.

17) Memoria virtual. Paginación con demanda
a. Cómo y quién realiza la traducción lógica en física. Tener en cuenta HW, actividades del SO, estructuras de datos, etc.

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.

18)

Sistema de archivos.

SISTEMA DE E/S (buffer cache)

19) File system, estructuras del system v, crear archivos. /home/user/final.txt estructras de las indicadas son utilizadas (creadas, modificadas y consultadas)

20) Describa la estructura del sistema de archivos de UNIX system v ¿si quisiera crear (misma pregunta para borrar) un nuevo archivo, por ejemplo /home/usuario/final.txt que estructuras de las indicadas son utilizadas (modificadas, creadas y/o utilizadas) ¿Cómo?

1) Buffer cache: ¿Para que es utilizado? Describa cómo funciona el buffer cache de System v (visto en la teoría)

2) Describa la relación existente entre el buffer cache y el sistema de archivos (file system) y entre el buffer cache y el sistema de E/S al momento de resolver por ejemplo una system call “read”.

3) Buffer cache: relacione los 2 subsitemas de un so y su interaccion. Para un desarrollo ordenado utilice la syscall read(fs,buff, count) indicando las tareas que se suceden desde que un proceso la ejecuta hasta que le es devuelto el control del mismo.

4) Analice en el sistema de archivos de UNIX system v la cracion de un nuevo archivo (por ejemplo: en el directorio /hombre/usuario). Para el análisis tenga en cuenta que pasos se suceden, que estructuras del sistema de archivos son utilizadas y modificadas.

5) Describa el sistema de archivos en Unix System V. describa los pasos necesarios para abrir un archivo open('/home/iso/finales.doc').

RESUMEN

PC: program counter: Contiene la dirección de la proxima instrucción a ser ejecutada.

IR: instruction register : contiene la instruccion a ser ejecutada

MAR: memory address register

MBR: memory buffer register

I/O: entrada/salida address register

I/O: entrada/salida buffer register

PSW (program status Word): contiene códigos de resultados de operaciones, habilita/deshabilita interrupciones, indica el modo de ejecución (supervisor/user).

PCB (process control block): Estructura asociada al proceso. Una por proceso. Contiene información asociada con cada proceso: Estado, Contenido del PC (program counter), Contenido de los Registros de la CPU. Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina Información relacionada con: Planificación, Ubicación en memoria, Accounting, Estado de I/O.

Estados de un proceso:

Terminado (terminated): el proceso ha terminado su ejecución y ya no compite mas por el cpu.

Nuevo: Un usuario "dispara" el proceso. Un proceso es creado por otro proceso: su proceso padre. En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.

Listo: El scheduller de largo plazo elige el proceso para cargarlo en memoria. El proceso sólo necesita que se le asigne CPU. Está en la cola de procesos listos (ready queue).

En ejecución (running): El scheduler de corto plazo lo eligió para asignarle CPU. Tendrá la CPU hasta que se termine el período de tiempo asignado (quantum) o hasta que necesite que se produzca un evento determinado.

En Espera: El proceso necesita que se cumpla el evento esperado para continuar. El evento puede ser la terminación de una I/O solicitada, o la llegada de una señal por parte de otro proceso. Sigue en memoria, pero no tiene la CPU. Al cumplirse el evento, pasará al estado de listo.

Conceptos a tener en cuenta:

Dispatcher: hace cambio de contexto, cambio de modo de ejecución...”despacha” el proceso elegido por el short term (es decir, “salta” a la instrucción a ejecutar).

Cuando esto ocurre, el sistema operativo ejecuta inmediatamente la rutina de **cambio de contexto**. Esta rutina realiza las siguientes operaciones en el orden indicado:

1. **Salvar el estado del programa que se estaba ejecutando**. El estado, también denominado *contexto*, consiste en los valores de todos los registros del microprocesador. Se copian en la memoria principal.
2. **Seleccionar otro programa para ejecutar**. Entre todos los programas que estén preparados para ejecutarse, la rutina selecciona uno de ellos siguiendo algún algoritmo equitativo.
3. **Restaurar el estado del programa seleccionado**. Para ello, se toma el estado previamente copiado en la memoria principal y se vuelca en los registros del microprocesador.
4. **Ejecutar el programa seleccionado**. La rutina termina su ejecución saltando a la instrucción que estaba pendiente de ejecutar en el programa seleccionado.

Loader: carga en memoria el proceso elegido por el long term.

Scheduler de long term: 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.

Scheduler de medium term: Si es necesario, reduce el grado de Multiprogramación. Saca temporariamente de memoria los procesos que sea necesario para mantener el equilibrio del sistema. Terminos asociados: swap out (sacar de memoria), swap in (volver a memoria).

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

The interrupt handling will do the switch to kernel mode

The only way an user space application can explicitly initiate a switch to kernel mode during normal operation is by making an system call such as open, read, write etc.
Whenever a user application calls these system call APIs with appropriate parameters, a software interrupt/exception(SWI) is triggered.

As a result of this SWI, the control of the code execution jumps from the user application to a predefined location in the Interrupt Vector Table [IVT] provided by the OS.

This IVT contains an adress for the SWI exception handler routine, which performs all the necessary steps required to switch the user application to kernel mode and start executing kernel instructions on behalf of user process.

SYSTEM CALL: (investigar más sobre estas y buscar información en internet)

Es la forma en que los programas de usuario acceden a los servicios del S.O.

Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques o tablas en memoria o pilas.

count = read (file, buffer, nbytes);

Se ejecutan en modo supervisor.

Categorías de system calls:

- Control de procesos
- Manejo de archivos
- Manejo de dispositivos
- Mantenimiento de información del sistema
- Comunicaciones

<u>System call y procesos.</u>

1) **System calls: Definicion, explicar cómo pueden ser implementadas para ello tenga en cuenta: participación o apoyo de hw, modos de ejecución, stack (pilas)**

System calls: es la forma en que los programas de usuario acceden a los servicios del sistema operativo. Los parámetros asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques, tablas de memorias o pilas. Count=read(file, buffer, nbytes). Se ejecutan en modo supervisor. Pueden ser implementadas de la siguiente forma:

1 User o Kernel Mode: Modo de ejecución en el que se encuentra la CPU user (U) o kernel (K)
☐☐☐***Hard o Soft:*** Si el que realiza la operación es el Hardware(H) o el Software(S)
☐☐☐***Stack Utilizado:*** Indica si el stack que se esta utilizando es el de Usuario (U) o de Kernel (K)

<u>GlibC:</u> biblioteca estándar de lenguaje C de GNU

1-U, 2-S, 3-U: El proceso de Usuario llama a una Syscall por medio de la GlibC. La GlibC pone los parámetros para la syscall en el Stack y eleva una interrupción.

1-U, 2-H, 3-U: Cambia a modo kernel.

1-K, 2-H, 3-U: Coloca el PC y PSW en el,stack (puede que se coloquen mas registros, dependiente de la arquitectura). Se

coloca en el PC la dirección de la rutina de atención de interrupción que se extrae de la IDT y se continua la ejecución.

1-K, 2-S, 3-U: Se sacan los parámetros a la syscall del stack. De ser necesario se pueden guardar otros mas registros del proceso actual en el Stack o en la PCB, depende de la implementación del SO. Se cambia a kernel stack, guardando la dirección del stack en User Mode en la PCB.

1-K, 2-S, 3-K: Se colocan los parámetros para la syscall en el Stack. Se ejecuta la Syscall. Si la Syscall bloquea el Proceso entonces De ser necesario se guarda mas información sobre el proceso bloqueado (registros, estados, etc.). Se ejecuta el Short Term Scheduller para seleccionar un nuevo proceso. Se realiza el context switch entonces Se cargan los registros del nuevo proceso, Se acomoda la dirección del Stack, dejando apuntando a la dirección que el HW dejo previo a que el proceso seleccionado sea suspendido. Se acomodan los datos necesarios en la PCB o estructuras utilizadas.
1-K, 2-S, 3-U: Se cambia a modo usuario.
1-U, 2-S, 3-U: Se ejecuta ret.
1-U, 2-H, 3-U: Se sacan de la pila el PSW y PC
1-U, 2-S, 3-U: Continúa la ejecución del proceso actual

2) **System calls: Suponiendo que un proceso x ejecuta una system call que ocasiona que el proceso sea bloqueado por ejemplo write() , analice y describa que eventos, actividades, se suceden hasta que otro proceso Y comienza su ejecución (tener en cuenta pila, notificaciones, dispacher, etc)**

Cuando hay un trap o una interrupción, el bit de modo se pone en modo supervisor. No es el proceso de usuario quien hace el cambio. Lo hace el HW al detectar una trampa o una interrupción.

Se emite una interrupción para invocar al Kernel ☐int $0x80

Se llama al Interruption Handler ☐System Call Handler

El proceso indica con un número la System Call que desea invocar ☐Syscall Number

3) **Describa las estructuras de datos y actividades necesarias para implementar el manejo de procesos dentro de un sistema operativo (ayuda: tenga en cuenta: estados de un proceso, estructuras necesarias para su manejo y actividades del sistema operativo durante la planificación de los procesos).**

Proceso: Es dinámico,Tiene program counter y Su ciclo de vida comprende desde que se lo "dispara" hasta que termina

Un proceso tiene, como mínimo 3 partes o regiones: Código (también llamado texto). Datos (variables globales) y Stack (contiene datos temporarios como parámetros de subrutinas, variables temporales y direcciones de retorno).

Un proceso cuenta con 2 stack: uno para modo **usuario** y otro para modo **kernel**

☐Se crea automáticamente y su medida se ajusta en run-time.

☐Está formado por *stack frames* que son *pushed* (al llamar a una rutina) y *popped* (cuando se retorna de ella)

☐El *stack frame* tiene los parámetros de la rutina(variables locales), y datos necesarios para recuperar el stack frame anterior (el contador de programa y el valor del stack pointer en el momento del llamado)

Atributos del proceso: Identificación del proceso, y del proceso padre

- Identificación del usuario que lo "disparó"

- Si hay estructura de grupos, grupo que lo disparó

- En ambientes multiusuario, desde que terminal y quien lo ejecuto.

PCB (process control block): Estructura asociada al proceso. Una por proceso. Contiene información asociada con cada proceso: Estado, Contenido del PC (program counter), Contenido de los Registros de la CPU. Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina Información relacionada con: Planificación, Ubicación en memoria, Accounting, Estado de I/O.

Espacio de direcciones de un proceso: Es el conjunto de direcciones de memoria que ocupa el proceso.

-No incluye su PCB o tablas asociadas

-Un proceso en modo usuario puede acceder sólo a su espacio de direcciones; en modo kernel, a estructuras del kernel o a espacios 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.

Cambio de contexto: Se produce cuando la CPU cambia de un proceso a otro.

- Se debe resguardar info del proceso saliente, que pasa a espera y retornará después la CPU.

- Se debe cargar la información asociada al nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada.

- Es tiempo no productivo de CPU

- El tiempo que consume depende del soporte de HW

Colas de planificación de procesos: De trabajos o procesos: de todos los procesos en el sistema

-De procesos listos: residentes en memoria principal, en estado de listo y esperando para ejecutarse

-De dispositivos: procesos esperando por un dispositivo de I/O

ESTADO DE LOS PROCESOS:

Terminado (terminated): el proceso ha terminado su ejecución y ya no compite mas por el cpu.

Nuevo: Un usuario "dispara" el proceso. Un proceso es creado por otro proceso: su proceso padre. En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.

Listo: El scheduller de largo plazo elige el proceso para cargarlo en memoria. El proceso sólo necesita que se le asigne CPU. Está en la cola de procesos listos (ready queue).

En ejecución (running): El scheduler de corto plazo lo eligió para asignarle CPU. Tendrá la CPU hasta que se termine el período de tiempo asignado (quantum) o hasta que necesite que se produzca un evento determinado.

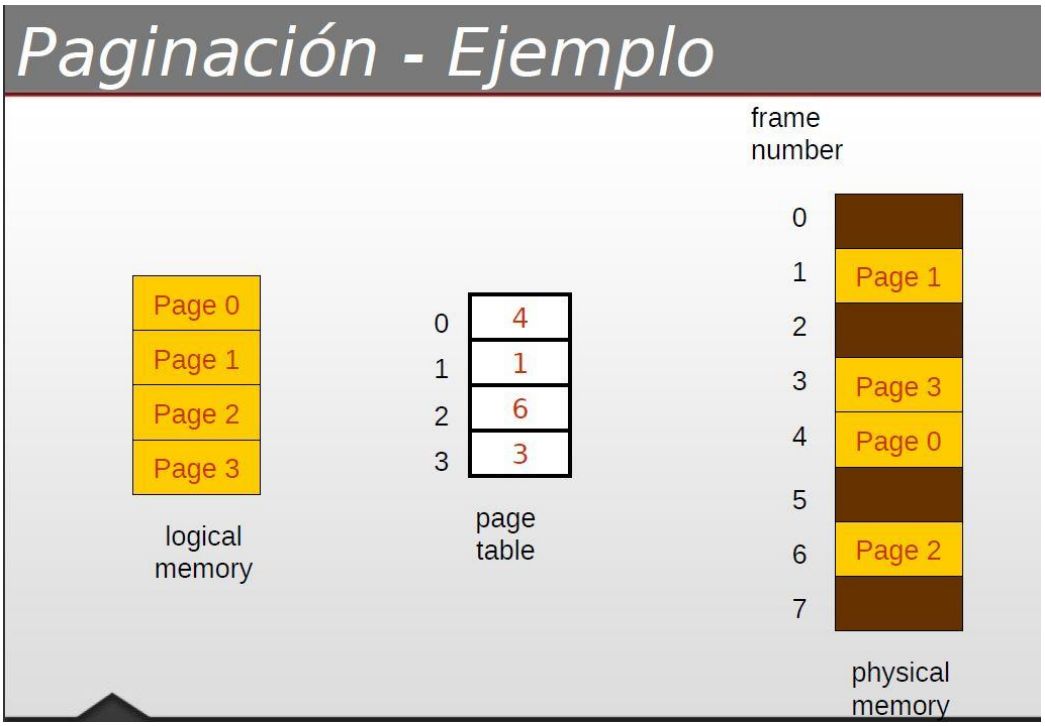
En Espera: El proceso necesita que se cumpla el evento esperado para continuar. El evento puede ser la terminación de una I/O solicitada, o la llegada de una señal por parte de otro proceso. Sigue en memoria, pero no tiene la CPU. Al cumplirse el evento, pasará al estado de listo.

MEMORIA

1) ***Paginación y segmentación. Como funcionan (tenga en cuenta: estructuras necesarias, participación del hw, etc). Cite ventajas y desventajas de un esquema respecto al otro.***

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 tabla de páginas por cada proceso que: Indica el marco donde está situada cada página Y la dirección lógica consiste en un número de página y un desplazamiento dentro de la misma.

Ejemplo:



Con paginación vimos que el espacio de direcciones de un proceso no necesariamente debe estar "contiguo" en la memoria para poder ejecutarse.

-El HW traduce direcciones lógicas a direcciones físicas utilizando las tablas de páginas que el SO administra

Segmentación: soporta "el punto de vista de un usuario".

Un programa es una colección de segmento. Un segmento es una unidad lógica como Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc.

Similar a particiones dinámicas.

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-tablelengthregister (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.

Conclusión:

La técnica de paginación permite dividir los espacios de memoria en secciones físicas de igual tamaño, pero al ser las secciones del igual tamaño ocurren mucho problemas ya habrá aplicaciones que requieran más del tamaño asignado, es por ello que también se utiliza la segmentación con el objetivo se subdividir espacios de memoria en bloques o segmentos de diferentes longitudes, pero a pesar de ser bastante buena, también ocurren ciertos problemas ya que al asignar la memoria necesaria para cada proceso, nos van quedando huecos en la memoria.

Es por ello que se está utilizando lo mejor de cada técnica en una versión llamada segmentación paginada

2) ***Memoria virtual con paginación por demanda: funcionamiento, estructuras utilizadas, participación del hw y actividades desarrolladas por el sistema operativo, fallo de pagina***.

<u>RESPUESTA COPIADA DEL RESUMEN</u>

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 memoriase 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.

RESPUESTA HECHA POR MI

Motivación para el uso de MV:

Podemos pensar también que, no todo el espacio de direcciones del proceso se necesito en todo momento:

-Rutinas o Librerías que se ejecutan una única vez (o nunca)

-Partes del programa que no vuelven a ejecutarse

-Etc.

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.

Ventajas de la MV:

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 mas 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

Requisitos para la MV:

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 (memoria principal) y MS(memoria secundaria).

MV 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.)

Fallos de página: 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 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 y 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

Tabla de página:

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 organizarlas:

-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

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 víctima 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).

3) ***HIPERPAGINACION***

Respuesta de internet

hiperpaginación (en inglés, *thrashing*) a la situación en la que se utiliza una creciente cantidad de recursos para hacer una cantidad de trabajo cada vez menor. A menudo, se refiere a cuando se cargan y descargan sucesiva y constantemente partes de la imagen de un proceso desde y hacia la memoria principal y la memoria virtual o espacio de intercambio. En un estado normal, esto permite que un proceso bloqueado y no listo para correr deje lugar en memoria principal a otro proceso listo. Cuando se produce hiperpaginación, los ciclos del procesador se utilizan en llevar y traer páginas (o segmentos, según sea el caso) y el rendimiento general del sistema se degrada notablemente

Las formas de evitar la hiperpaginación fueron un área importante de investigación en los años 70 y se desarrollaron varios algoritmos complejos pero efectivos. Los mismos se basan en la idea de intentar adivinar qué páginas serán utilizadas próximamente,

basados en su historia reciente y utilizando como hipótesis el principio de cercanía de referencias. Estos son los denominados algoritmos de reemplazo de páginas.

Desde un punto de vista más práctico, se puede evitar la hiperpaginación:

Aumentando la cantidad de memoria RAM (generalmente la mejor solución a largo plazo).

Disminuyendo la cantidad de aplicaciones corriendo en la computadora.

Ajustando el tamaño de la partición de intercambio.

RESPUESTA DE LAS DIAPOSITIVAS

Hiperpaginacion: 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 SO monitorea el uso de la CPU.

2) Si hay baja utilización aumenta el grado de multiprogramación.

3) Si el algoritmo de reemplazo es global, pueden sacarse 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 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.

-Veremos algunas técnicas como la estrategia de Working Set, con el modelo de localidad y la estrategia de 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.

-Δ chico: no cubrirá la localidad

-Δ grande: puede tomar varias localidades

→ m = cantidad frames disponibles

→ WSSi= medida del working set del proceso pi.

→ sumatoria 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

- 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

-La ventana es móvil

Sistema de archivos. Y IN/OUT

SISTEMA DE E/S (buffer cache)

Conceptos a tener en cuenta:

Polling en computación hace referencia a una operación de consulta constante, generalmente hacia un dispositivo de hardware, para crear una actividad sincrónica sin el uso de interrupciones, aunque también puede suceder lo mismo para recursos de software.

Esto, aplicado a programación puede ser visto como una pobre implementación en búsqueda del sincronismo de procesos. Por ejemplo, se podría consultar constantemente un directorio del sistema de archivos para indicarle al usuario cuándo llegan nuevos contenidos a la misma. Sin embargo, estas constantes consultas degradarían el rendimiento del equipo y probablemente sería mejor implementar la solución por otro medio, en

particular, pidiéndole al sistema operativo que informe de transferencias a ese directorio en particular.

6) Buffer cache: ¿Para que es utilizado? Describa cómo funciona el buffer cache de System v (visto en la teoría)

Definición: Buffers en memoria principal para almacenamiento temporario de sectores de disco. Contienen una copia de algunos sectores de disco.

Objetivo: MINIMIZAR LA FRECUENCIA DE ACCESO AL DISCO

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

-Se copia al espacio de direcciones de usuario.

-O se trabaja como memoria compartida (no se copia permitiendo acceso a varios procesos).

Buffer cache de System V:

Objetivo y estructura

-Minimizar la frecuencia de acceso a disco

-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: el header y el lugar donde se almacena el bloque de disco traído a memoria

El header

-Identifica por nro. de dispositivo y nro. de bloque

-Tiene punteros:

- 2 punteros para la hash queue
- 2 punteros para la free list
- un puntero al bloque en memoria

-Estado (indicación)

Estados de los buffers

-Free o disponible

-Busy o no disponible (en uso por algún proceso)

-El kernel está escribiendo a disco o leyendo del disco.

-Delayed write: buffers que hayan sido modificados en memoria, pero el bloque original en disco todavía no fue actualizado.

Free List

-Organiza los buffers disponibles, es decir, los buffers donde se puede cargar un nuevo bloque de disco.

-No necesariamente los bloques están vacios

-Se ordena según LRU (least recent used)

Hash Queues

-Son colas para optimizar la búsqueda de un buffer en particular

-Se organizan según una función de hash usando (dispositivo,#bloque)

Búsqueda/recuperación de un buffer: 1er escenario

-El kernel encuentra el bloque en la hash queue .

-Está disponible (está en la free list).

-Se remueve ese buffer de la free list

-Pasa a estado busy

-El proceso puede usar el bloque

21) Describa la estructura del sistema de archivos de UNIX system v ¿si quisiera crear (misma pregunta para borrar) un nuevo archivo, por ejemplo /home/usuario/final.txt que estructuras de las indicadas son utilizadas (modificadas, creadas y/o utilizadas) ¿Cómo?

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

Se caracteriza por:

- Poseer una estructura jerarquica
- Realizar un tratamiento consistente de los datos de los archivos
- Poder crear y borrar archivos
- Permitir un crecimiento dinamico de los archivos
- Proteger los datos de los archivos
- Tratar a los dispositivos y periféricos (terminales, unidades de disco, citas, etc) como si fuesen archivos.

Estructura:

-El bloque de arranque (boot): Ocupa la parte del principio del sistema de archivos, normalmente en el primer sector, y puede contener el código

de arranque. Este código es un pequeño programa que se encarga de buscar el sistema operativo y cargarlo en memoria.

-El superbloque: describe el estado de un sistema de archivos. Contiene informacion a cerca de su tamaño, total de archivos que puede contener, que espacio libre queda, etc.

- La lista de inodos (nodo índice): Se encuentra a continuación del superbloque. Esta lista tiene una entrada por cada archivo, donde se guarda una descripción del mismo: situación del archivo en el disco, propietario, permisos de acceso, fecha de actualización, etc. El administrador del sistema es el encargado de especificar el tamaño de la lista de inodos al configurar el sistema.

-Los bloques de datos: empiezan a continuación de la lista de inodos y ocupa el resto del sistema de archivos. En esta zona es donde se encuentra situado el contenido de los archivos a los que hace referencia la lista de inodos. Cada uno de los bloques destinados a datos solo puede ser asignado a un archivo, tanto si lo ocupa totalmente como si no.

4.4.2. read (Lectura de Datos de un Archivo).

- nº_bytes_leídos = **read**(descriptor, buffer, número_bytes_a_leer)
- Acciones:
  - Obtener la entrada en la tabla de archivos a partir del descriptor del archivo.
  - Comprobar los permisos.
  - Obtener el inodo (*inode*) siguiendo los punteros.
  - Hasta el fin de la lectura o error o no quedan bytes en el archivo:
    - Desplazamiento ⇒ nº de bloque de disco (*bmap*).
    - Calcula el desplazamiento en el bloque para comenzar la E/S.
    - Calcular el nº de bytes a leer dentro del bloque.
    - Si el nº bytes a leer = 0 ⇒ Fin de búsqueda.
    - Si no:
      - Leer el bloque en un buffer (*buffer caché*).
      - Copiar los datos del buffer ⇒ dirección destino del proceso de usuario.
      - Actualiza el desplazamiento en el archivo, dirección en el proceso de usuario, nº de bytes que quedan por leer.
  - Actualizar el desplazamiento en la tabla de archivos.

4.4.5. creat (Creación de un Archivo).

- Crea un archivo nuevo en el sistema de archivos. Tiene la misma funcionalidad que **open**.
- Semántica de la llamada:
  - Si el archivo no existía ⇒ se crea uno nuevo.
  - Si ya existía ⇒ trunca su contenido.
- descriptor = **creat**(nombre, modos)
- Acciones (algoritmo):
  - Obtener el inodo (*inode*) a partir del nombre (*pathname*).
  - Si el archivo no existía ⇒ Asignar el inodo (*inode*) libre y crear la entrada en el directorio padre si tenemos permisos de escritura en el directorio padre.
  - Si ya existía:
    + Si el acceso no está permitido ⇒ Retornar error.
    + Si el acceso está permitido:
      * Truncar el archivo si tenemos permiso de escritura sobre el archivo.
      * Liberar todos sus bloques de datos.
  - En cualquier caso:
    + Asignar una entrada en la tabla de archivos, cuenta de referencia = 1.
    + Asignar una entrada en la tabla de descriptores de archivos.

4.2. ESTRUCTURA GENERAL DE UN SISTEMA DE ARCHIVOS DE UNIX.

- Los sistemas de archivos suelen estar situados en dispositivos de almacenamiento modo bloque, tales como discos o cintas.

- Un sistema UNIX puede manejar uno o varios discos físicos, cada uno de los cuales puede contener uno o varios sistemas de archivos. Los sistemas de archivos son particiones lógicas del disco.

- Hacer que un disco físico contenga varios sistemas de archivos permite una administración más segura, ya que si uno de los sistemas de archivos se daña, perdiéndose la información que hay en él, este accidente no se habrá transmitido al resto de los sistemas de archivos que hay en el disco y se podrá seguir trabajando con ellos para intentar una restauración o una reinstalación.

- El *kernel* del sistema operativo trabaja con el sistema de archivos a un nivel lógico y no trata directamente con los discos a nivel físico. Cada disco es considerado como un dispositivo lógico que tiene asociados unos números de dispositivo (*minor number* y *major number*). Estos números se utilizan para acceder al controlador del disco. Un controlador del disco se va a encargar de transformar las direcciones lógicas (*kernel*) de nuestro sistema de archivos a direcciones físicas del disco.

- Un sistema de archivos se compone de una secuencia de bloques lógicos, cada uno de los cuales tiene un tamaño fijo (homogéneo). El tamaño del bloque es el mismo para todos el sistema de archivos y suele ser múltiplo de 512.

4.3.3.2. Directorios.

- Los directorios son los archivos que nos permiten darle una estructura jerárquica a los sistemas de archivo de UNIX. Su función fundamental consiste en establecer la relación que existe entre el nombre de un archivo y su inodo correspondiente.
- En el UNIX System V, un directorio es un archivo cuyos datos están organizados como una secuencia de entradas, cada una de las cuales tiene un número de inodo y el nombre de un archivo que pertenece al directorio. Al par (inodo , nombre_del_archivo) se le conoce como enlace (link) y puede haber varios nombres de archivos, distribuidos por la jerarquía de directorios, que estén enlazados con un mismo inodo.
- El tamaño de cada entrada del directorio es de 16 bytes, dos dedicados al inodo y 14 dedicados al nombre del archivo. En la siguiente figura podemos ver la estructura típica de un directorio.

Desplazamiento	Número de inodo (*inode*)	Nombres de archivos
0	83	.
16	2	..
32	1798	init
48	1276	fsck
64	85	clri
80	1268	motd
96	1799	mount
112	88	mknod
128	2114	passwd
144	1717	umount
160	1851	checklist

Figura 4.5. Ejemplo de estructura de un directorio para el UNIX System V.

- Las dos primeras entradas de un directorio reciben los nombres de "." y "..". El archivo "." tiene asociado el inodo correspondiente al directorio actual y al archivo ".." se le asocia el inodo del directorio padre del actual. Estas dos entradas están presentes en todo directorio, y en el caso del directorio "/" (raíz), el programa mkfs (make file system, programa mediante el cual se crea un sistema de archivos) se encarga de que el archivo ".." se refiera al propio directorio raíz.
- El *kernel* maneja los datos de un directorio con los mismo procedimientos con que se manejan los datos de los archivos regulares (ordinarios), utilizando la estructura inodo (*inode*) y los bloques de acceso directos e indirectos.

- A pesar de que el tamaño del bloque es homogéneo en un sistema de archivos, puede variar de un sistema a otro dentro de una misma configuración UNIX con varios sistemas de archivos. El tamaño elegido para el bloque va a influir en las prestaciones globales del sistema. Por un lado, interesa que los bloques sean grandes para que la velocidad de transferencia entre el disco y la memoria sea grande. Si embargo, si los bloques lógicos son demasiado grandes, la capacidad de almacenamiento del disco se puede ver desaprovechada cuando abundan los archivos pequeños que no llegan a ocupar un bloque completo. Valores típicos para el tamaño del bloque: 512, 1024 y 2048 (bytes).

- En la Figura 4.1 podemos ver, en un primer nivel de análisis, la estructura que tiene un sistema de archivos UNIX:

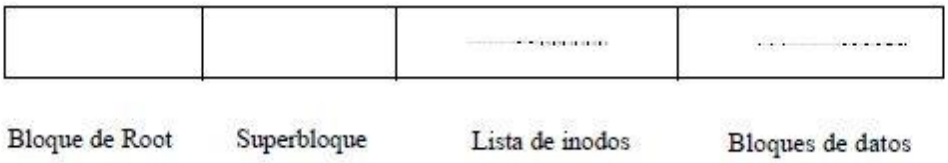


Figura 4.1. Primer nivel en la estructura de un sistema de archivos.

- En la figura anterior podemos observar cuatro partes:
  - El *bloque de arranque* (boot). Ocupa la parte del principio del sistema de archivos, normalmente en el primer sector, y puede contener el código de arranque. Este código es un pequeño programa que se encarga de buscar el sistema operativo y cargarlo en memoria.
  - El *superbloque* describe el estado de un sistema de archivos. Contiene información acerca de su tamaño, total del archivo que puede contener, qué espacio libre queda, etc.
  - La lista de inodos (nodos índice). Se encuentra a continuación del superbloque. Esta lista tiene una entrada por cada archivo, donde se guarda una descripción del mismo: situación del archivo en el disco, propietario, permisos de acceso, fecha de actualización, etc. El administrador del sistema es el encargado de especificar el tamaño de la lista de inodos al configurar el sistema.
  - Los bloques de datos empiezan a continuación de la lista de inodos y ocupa el resto del sistema de archivos. En esta zona es donde se encuentra situado el contenido de los archivos a los que hace referencia la lista de inodos. Cada uno de los bloques destinados a datos sólo puede ser asignado a un archivo, tanto si lo ocupa totalmente como si no.