Respuestas a Finales by
Cristian Barreto y Facundo Díaz

1) Procesos y SystemCalls

a.- Se quiere implementar el concepto de "proceso" junto con el "algoritmo de planificación de CPU Round Robin. ¿Cómo los implementaría? Tenga en cuenta apoyo del HW que necesitaría y con que fin (interrupciones, modos de ejecución, etc) asi como las estructuras de datos que necesitara el SO junto con tareas que deberá realizar el mismo, implementación del Quantum, tareas realizadas cada vez que un Quantum se consume, etc.

Un proceso es un programa en ejecución.
Un proceso tiene como minimo:

- Sección de código (texto)
- Sección de datos (variables globales)
- Stacks (datos temporarios, parámetros, direcciones de retorno)

Stacks
Un proceso cuenta con 1 o mas stacks, en general: usuario y kernel, y se crean automáticamente junto con el proceso
Su medida se ajusta run-time (en tiempo de ejecución)
Esta formado por stackframes que son pushed (al llamar a la rutina) y popped (cuando se retorna de ella)
El stackframe tiene los parámetros de la rutina, y los datos necesarios para recuperar el stackframe anterior (el contador del programa y el valor del stack pointer SP en el momento llamado)

Process Control Block (PCB)
Estructura de datos asociada al proceso. Una por proceso y no esta dentro de su espacio de direcciones
Contiene información asociada con cada proceso: estado, pc, registros de la CPU, pid, prioridad en caso de que tenga, información relacionada con la planificación, etc
El SO necesita de esta estructura para administrarlo y ejecutarlo correctamente
Es lo primero que se crea cuando se lo crea y lo ultimo que se borra cuando se lo borra

Cambio de contexto (ContextSwitch)
Se produce cuando la CPU cambia de un proceso a otro
Se debe guardar información del proceso saliente en la PCB, que pasa a esperar y luego se le retornara la cpu
Se debe cargar información del proceso entrante y continuar desde su ultima instrucción
Es tiempo no productivo de CPU

El algoritmo Round Robin es un algoritmo de planificación de procesos apropiativo simple de implementar, en la que la lista de procesos se planifica por FIFO (first in first out). Dentro de cada proceso el SO asigna una porción de tiempo (quantum) equitativa y ordenada, tratando a todos los procesos con la misma prioridad.

Si al proceso se le termina su quantum, se provoca una interrupción en la que se cambia a modo supervisor, y se le saca la CPU al proceso para realizar el cambio de contexto. El proceso al que se le saco la cpu pasa al estado de "ready".
El cambio de contexto consiste en guardarse toda la información del proceso saliente en su PCB y cargar toda la nueva información del proceso entrante. Este trabajo lo realiza el Dispatcher, que es el modulo encargado de hacer el cambio de contexto o cambio de modo de ejecución, despachando el proceso elegido por Short Term Scheduler.
Si al proceso termina su ráfaga de CPU antes que finalice su Quantum, el Dispatcher le asigna inmediatamente el CPU otro proceso.
Este algoritmo es utilizado por procesos interactivos, en los que es necesario que ningún proceso se apropie de la CPU (por ejemplo para interaccion con el usuario).

b)_SystemCalls. Definicion.Explicar como puede ser implementadas, para ello tenga en cuenta: participación o apoyo del Hardware, Modos de Ejecución, stacks(pilas) y todo lo que considere pertinente.

Las llamadas al sistema son la forma en que un programa de usuario accede a los servicios del Sistema Operativo. Estas llamadas generalmente son instrucciones de lenguaje ensamblador. Desde el punto de vista del programador son tan simples como la llamada a una funcion. Desde el punto de vista del kernel, necesita :"**ejecucion en modo provilegiado, pasaje de parámetros, y retorno de resultados entre modos de ejecucion diferentes**".

Como implementar una System call ?

- El primer paso es definir el proposito.
- Definir argumentos, valor de retorno, codigos de error.
- Ser lo mas general posible, contar con portabilidad y robustez.
- Deben verificar que los parámetros son validos y legales. La seguridad y estabilidad del sistema están en juego. También verificar que sean correctos
- Un chequeo importante: verificar los punteros que el usuairo provee. Se debe asegurar que:
    - → El puntero apunta a una region en espacio de usuarios
    - → El puntero apunta a una region en el espacio del procesos
    - → Si lee o escribe en la memoria, que esta este habilitada para la accion

Modo de Ejecucion

El SO, le tiene que ofrecer al usuario un modo de "acceso seguro", este es un servicio que ofrece el SO, no solo se refiere a seguro de terceros, sino de si mismos, ya que si lo accedo a lugares que no debo y ejecuto instrucciones que son privilegiadas sin darme cuentan puedo equivocarme y generar un gran destrozo; que puede ir desde borrar un archivo no importante, hasta borrar un archivo escencial del SO. Entonces el "acceso seguro" del que hablamos es el bit de modo que figura en el **PSW (Program Status Word),** que es un registro de control y de estado de la CPU. Este bit de modo indica si el procesador esta ejecutando en modo supervisor o en modo usuario (***PRIVILEGIADA=0, USUARIO=1***).
Si al llamar a una SystemCall la misma tiene una funcion en la que se le pasan parametros, esta tiene que poder guardar esos datos en algún lugar para poder llevar a cabo su ejecución (siempre en modo supervisor). Para esto se utiliza la pila, que es una estructura con un espacio reservado de memoria que se comporta como una colección de elementos, donde solo es posible acceder al elemento que esta en tope de la pila mediante el registro **SP (Stack Pointer)**.

Esta estructura servira como puente en el pasaje de parámetros, y tambien para guardar resultados y retornar los resultados que realiza la funcion hacia el usuario.

c)_Suponiendo que un proceso X ejecuta una SysCall bloqueante, por ejemplo ”read”, desarrolle que eventos se suceden hasta que un nuevo proceso Y es asignado a la CPU (tener en cuenta planificadores,dispatcher,etc).

Las SysCall ejecutan una serie de pasos. Cuando se ejecuta el proceso X , que hace la llamada al sistema “read”, el programa que invoca la llamada, apila los parametros que usa el procedimiento en la pila. Luego apila la llamada al procedimiento propiamente dicha. Luego se interrumpe al CPU para pasar de modo usuario a modo supervisor. Aca entra en juego el Dispatcher que le saca el procesador al proceso que estaba corriendo, bloqueandolo, y se lo da al proceso que interrumpio, en este caso el proceso Y. Antes de entregar el procesador al proceso que interrumpio, se tuvo que haber guardado el contexto actual para luego restaurarlo, este cambio de contexto se hace en modo supervisor el cual tambien debe generarse una interrupción para el cambio de modo.
Despues el kernel tiene una tabla de llamadas a sistema donde busca el numero de la SystemCall (indentificador de cada SysCall). Luego comienza la ejecucion del manejador de SystemCall, se ejecuta la funcion determinada, y cuando este termine, (osea termino la funcion), el Dispatcher devuelve el control al procedimiento X en el espacio de usuario. Por lo que antes tuvo que haber un cambio de modo de Supervisor a Usuario.
Y por ultimo se modifican los apuntadores de la pila para eliminar los parámetros que se hayan metido con la llamada al procedimiento X.

Otra opcion a explicar es...

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 (fork, read, write, tuvieja, etc..).
2) → Se guarda en el stack del usuario los parametros de la syscall y se lanza una interrupcion.
3) → Se cambia a modo Kernel.
4) → Se guarda el PC y el PWS(Program Status Word: indica el estado del programa actual) en la pila del usuario.
5) → Se carga en el PC la direccion de la subrutina que atiende la interrupcion.
6) → Se sacan los parametros de la SysCall en la pila del Usuario para ver que llamanda tiene que atender.
7) → Se guarda la direccion de la pila del usuario en el OVB del proceso y se pasa a utilizar la pila del kernel.
8) → Se guardan los parametros de la SusCall 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 direccion que el HW dejo precio ea que el proceso seleccionado sea suspendido
10) → Mas alla de que se haya bloqueado o no el proceso, quien ahora tiene el control de la CPU va a cambiar el modo a User Mode.
11) → Ejecutar la sentencia RET para obtener de la pila la direccion de retorno a ejecutar.
12) → Se obtienen de la pila el PSW y PC y se continua la ejecucion del proceso actual.

d)_Porque son necesarias? Porque no pueden ejecutarse en modo usuario?

Una **llamada al sistema** es una interfaz entre una aplicación del espacio del usuario y un servicio que provee el kernel. Debido a que el servicio se provee en el kernel, no es posible realizar una llamada directa; en cambio, se debe cruzar el límite entre el espacio del usuario y el kernel. La manera de hacerlo difiere según la arquitectura específica.
Estas mismas son importantes por que sino el usuario no tendria como comunicarme con el Kernel y por lo tanto no podria hacer ninguna funcion especial directa del kernel, es decir no puedo hacer ninguna funcion para que es sistema pueda realizar una operación escencial o servicios de la misma(ej manejo de CPU, memoria , E/S, partes del filesystem, etc..).
Se deben ejecutar en modo privilegiado o supervisor, porque si el usuario tuviera acceso a estas funciones, se podrian generar varios conflictos, ya que se podria implementar la incorrecta ejecucion de procesos(ya sea voluntaria o involuntariamente) y provocar deadlock o innanicion en los mismos procesos o en otros que precisan su correcta ejecucion, por lo tanto, se requiere un cambio de contexto(Usuario-Supervisor), cuando estos servicios entran en ejecucion.

e)_Se quiere ejecutar la instrucción que implica una lectura de una fila de una base de datos que esta en disco. Considerando que el controlador no sabe de base de datos ...¿Como se transforma esa instrucción en una orden para el controlador de disco? ¿Cual seria ese comando? Load o Store?

El comando seria load

f.- ¿Cómo trabaja systemcall fork() en Linux? Indique el apoyo por parte del HW

Los procesos en Linux tienen una estructura jerarquica, es decir, un proceso padre puede crear un nuevo proceso hijo y asi sucesivamente. La forma de crear un nuevo proceso hijo es mediante la SystemCall fork().
Cuando utilizamos la SystemCall fork() el proceso creado es una copia exacta del padre (mismas instrucciones, mismas variables), pero con un PID diferente y la memoria que ocupa tambien es diferente.
Las variables que se le asignan son una copia de las del padre y no una referencia, por lo que si se modifican en un proceso no se ve reflejado en el otro.
Por lo general después de hacer un fork() se hace un execve(),que carga un nuevo programa en el espacio de direcciones.
Un proceso padre puede tener 1 o mas procesos hijos.
Con respecto a la ejecución → el padre puede continuar ejecutándose concurrentemente con su hijo, o el padre puede esperar a que el/los procesos hijo terminen para continuar su ejecución

g.- Describa los estados de un proceso. Planificadores (schedulers), su relación con las transiciones entre los estados enumerados.

En su ciclo de vida, el proceso pasa por diferentes estados:

- NEW → el proceso esta en este estado ni bien es creado. Se crean las estructuras asociadas (PCB, stacks, etc) y el proceso queda en la cola de procesos, normalmente en espera de ser cargado a memoria

- READY → el proceso esta en estado de ready cuando el Long Term Scheduler lo eligió para cargarlo en memoria. El proceso queda en la cola de procesos listos en memoria principal esperando que se le asigne la CPU
- RUNNING → el Short Term Scheduler elije un proceso de la cola de listos y le asigna la CPU. Tendrá la CPU hasta que se termine su periodo de tiempo (Quantum) termine, o se agote su ráfaga, o hasta que se necesite realizar alguna operación de E/S
- WAITING → el proceso necesita que se cumpla el evento esperando para continuar. El evento puede ser la terminación de una E/S, 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 Ready.
- TERMINATED → proceso terminado

Transiciones:

- 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-Terminated → el proceso terminó su ráfaga
- Caso especial Running-Ready → cuando el proceso termina su Quantum sin haber necesitado ser interrumpido por un evento, pasa al estado de ready para volver a competir por el CPU, ya que no esta esperando por ningún evento

Modulos de planificación:

Son modulos del SO que realizan distintas tareas asociadas a la planificación de los procesos
Se ejecutan ante aquellos eventos que lo requieren, ejemplo:

- Creación/terminación de un proceso
- Evento de sincronización o de E/S
- Finalización de lapso de tiempo
- Etc

Estos modulos son el **Long Term Scheduler, Medium Term Scheduler, Short Term Scheduler, Dispatcher y Loader**

Long Term Scheduler → controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria. Puede no existir este shcheduler y absorver esta tarea el del short term

Medium Term Scheduler → si es necesario, reduce el grado de multiprogramación. Saca temporariamente de memoria los procesos que sea necesario para mantener el equilibrio del sistema. Swap-in, Swap-out de procesos a memoria

Short Term Scheduler → decide cual de los procesos en la cola de listos se elige para asignarle la CPU

Dispatcher →hace cambio de contexto, cambio de modo de ejecución. "Despacha" el proceso elegido por el short term (salta a la instrucción a ejecutar)

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

2) Administracion de memoria

a.- Paginacion y segmentación. Segmentacion paginada. Como funcionan (Tenga en cuenta: estructura necesarias, participación del HW y tareas del SO. Cite ventajas y desventajas de un esquema respecto al otro.

Paginación es una técnica de administración de memoria en la que la memoria principal es dividida lógicamente en pequeños trozos de igual tamaño llamado **"marcos",** y el espacio de direcciones de cada proceso es dividido en trozos de igual tamaño que los marcos llamadas **"paginas".** De esta forma, la cantidad de memoria desperdiciada por un proceso es el final de su ultima pagina, lo que minimiza la fragmentación interna y evita la externa.
Con esta técnica, la memoria se encuentra ocupada con paginas de diferentes procesos.
El sistema operativo mantiene una lista con los marcos libres para su uso, y una tabla de paginas por cada proceso, que contiene el marco en la que esta situada cada pagina de su proceso. De esta forma, las paginas de un proceso pueden no estar contiguamente ubicadas en memoria, pueden intercalarse con las paginas de otros procesos.
Las direcciones lógicas que maneja la CPU ahora consisten en un numero de pagina y un desplazamiento dentro de la misma. El numero de pagina es usado como un índice dentro de la tabla de paginas y una vez obtenida la dirección del marco de memoria, se utiliza el desplazamiento para ubicar la dirección real o dirección física. Este proceso lo realiza el **MMU (Memory Management Unit)**, que es dispositivo de hardware encargado de los accesos a memoria por parte de la CPU, la protección de la misma, el control de la cache, y como ya dicho, de la traducción de direcciones lógicas a físicas (explicado completamente en paginación por demanda).

Paginación por demanda
El único inconveniente de paginación pura es que todas las páginas de un proceso deben estar en memoria para que se pueda ejecutar. Esto hace que si los programas son de tamaño considerable, no puedan cargarse muchos a la vez, disminuyendo el grado de multiprogramación del sistema. Para evitar esto, y aprovechando el principio de cercanía de referencias donde se puede esperar que un programa trabaje con un conjunto cercano de referencias a memoria (es decir con un working set más pequeño que el total de sus páginas), se permitirá que algunas páginas del proceso sean guardadas en un **Area de Intercambio** o **Memoria Virtual**, que es una zona del disco (memoria secundaria) reservada para alojar parte de procesos que no caben en memoria principal mientras no se necesiten.
En este caso, el sistema operativo mantiene además el conocimiento sobre qué páginas están en memoria principal y cuáles no, usando la tabla de paginación. En cada entrada de la tabla de paginas (**Page Table Entry** o **PTE**), existe un bit de presencia, **Bit V,** que esta en 1 o activado si la pagina se encuentra en memoria principal, o en 0 si no se encuentra, y otro **Bit M,** que es el bit de modificación, que advierte que la pagina fue modificada en memoria principal y debe descargarse a disco antes de abandonar la misma (hay otros bits que indican otras cosas).

Cuando la CPU intenta acceder a una dirección de memoria lógica, la **MMU** (explicación de aca hasta el final) realiza una búsqueda en una memoria cache especial llamada **TLB (Translation Lookaside Buffer)**, que mantiene las tablas de paginas usadas hace menos tiempo. En esta memoria están las **PTE** donde se pueden rescatar las direcciones físicas corresponientes a algunas direcciones lógicas de forma directa. Cuando la dirección requerida por la CPU se encuentra en la TLB, (es decir que se encuentra la dirección física para esa dirección lógica), se entrega y a esto se lo conoce como un "TLB hit". En otro caso, cuando la dirección buscada no se encuentra en la TLB (fallo en TLB), el procesador busca en la tabla de páginas del proceso utilizando el número de página como entrada a la misma. Si el bit de presencia (Bit V) está activado, se carga esta PTE en el TLB y se devuelve la dirección física. En caso contrario, se informa al sistema operativo de la situación, mediante un "fallo de página", esto es, buscar la pagina en el área de intercambio y cargarla memoria física) usando uno de los algoritmos de reemplazo de páginas, para continuar con la ejecución desde la instrucción que causó el fallo.

Ventajas de la paginación:

- Transparente al programador
- Elimina fragmentación externa
- Posibilidad de cargar solo una parte del programa en memoria, el resto se cargara cuando se necesite
- Fácilidad de controlar las paginas ya que tienen todas el mismo tamaño
- No es necesario que las paginas estén contiguas en memoria

Desventajas de paginación:

- El costo de hardware y software se incrementa por la traducción de direcciones.
- Problema de fragmentación interna, si se requieren 5k para un programa, pero las paginas son de 4k, necesitaremos 2 paginas (8k), con lo que quedan 3k sin utilizar. La suma de estos espacios libres puede ser mayor que el de varias paginas, pero no podrán ser utilizados.
- Si ocurren muchos fallos de pagina la performance baja notablemente

Segmentación es una técnica de administración de memoria que se acerca mas al "punto de vista del usuario". Los programas se desarrollan, generalmente, en torno a un programa principal desde el que se bifurca a otras partes como procedimientos, funciones, pilas, etc. Desde este punto de vista, un programa es un conjunto de componentes lógicos o segmentos de tamaño variable, es decir, el espacio lógico de direcciones se considera como un conjunto de segmentos, cada uno definido por un identificador.
La segmentación de un programa la realiza el compilador.
Cada dirección lógica consiste en 2 partes: **selector del segmento, y desplazamiento dentro del segmento.**
La implementación mas obvia dentro de un espacio de memoria segmentado es asignar un segmento distinto a cada una de los espacios en memoria.

El sistema operativo mantiene una tabla de segmentos que permite mapear la dirección lógica en física, y cada entrada contiene:

- **Base:** dirección física de comienzo del segmento
- **Limite:** longitud del segmento

También tenemos el **STBR (Segment-Table Base Register)** que apunta a la ubicación de la tabla de segmentos, y el **STLR (Segment-Table Lenght Register)** que contiene la cantidad de segmentos de un programa.
Cuando la CPU desea acceder a una dirección de memoria lógica será necesario hacer la traducción a la dirección física, esta conversión la realiza el **MMU** consultando la tabla de segmentos correspondiente.

Sementacion por demanda

Igual que la paginación por demanda, un proceso no requiere que todos sus segmentos estén en memoria principal para poder ejecutarse. Por ello, el selector del segmento posee un bit de validez para cada segmento, el cual indica si la pagina se encuentra actualmente en memoria. Cuando un proceso direcciona un segmento, el MMU examina esta bit de validez, si el segmento esta en memoria principal el acceso continua sin problemas, en caso de que no esté se genera un trap al SO (falla de segmento) igual como sucede en la paginación por demanda. Las rutinas de administración de memoria deberán ver si hay lugar para alojar el nuevo segmento, si no lo hay se efectúa una compactación. Si luego de la compactación sigue sin haber espacio, se descarga un segmento al área de intercambio y se incorpora el nuevo segmento a memoria principal.

Ventajas de segmentación

- Visible al programador
- Facilita la modularidad, estructuras de datos grandes y da soporte a la compartición y protección
- El programador puede conocer las unidades lógicas de su programa
- Fácil compartir segmentos

Desventajas de la segmentación

- Problemas de fragmentación externa ya que los segmentos son de tamaño variable
- Se complica el manejo de memoria virtual, ya que los discos almacenan la información en bloques de tamaños fijos, mientras los segmentos son de tamaño variable
- Ahorra memoria, pero requiere de mecanismos adicionales da hardware y software
- Dificultad de traer un nuevo segmento a memoria, ya que será necesario encontrar un área de memoria libre ajustada a su tamaño (por su tamaño variable)

Segmentación paginada

Paginación y segmentación son técnicas diferentes, cada una de las cuales busca brindar las ventajas enunciadas anteriormente. Para la segmentación se necesita que estén cargadas en

memoria áreas de tamaños variables. Si se requiere cargar un segmento en memoria que antes estuvo en ella y fue removido a memoria secundaria, se necesita encontrar una región de la memoria lo suficientemente grande para contenerlo, lo cual no es siempre factible. En cambio recargar una página implica sólo encontrar un marco de página disponible.

A nivel de paginación, si quiere referenciar en forma cíclica n paginas, estas deberán ser cargadas una a una, generándose varias interrupciones por fallas de páginas. Bajo segmentación, esta página podría conformar un sólo segmento, ocurriendo una sola interrupción por falla de segmento. No obstante, si bajo segmentación se desea acceder un área muy pequeña dentro de un segmento muy grande, este deberá cargarse completamente en memoria, desperdiciándose memoria. Bajo paginación sólo se cargará la página que contiene los ítems referenciados.

Puede hacerse una combinación de segmentación y paginación para obtener las ventajas de ambas. En lugar de tratar un segmento como una unidad contigua, éste puede dividirse en páginas. Cada segmento puede ser descrito por su propia tabla de páginas.

Los segmentos son usualmente múltiplos de páginas en tamaño, y no es necesario que todas las páginas se encuentren en memoria principal a la vez. Además, las páginas de un mismo segmento, aunque se encuentren contiguas en memoria virtual, no necesitan estarlo en memoria real. Las direcciones tienen tres componentes: (s, p, d), donde la primera indica el número del segmento, la segunda el número de la página dentro del segmento y la tercera el desplazamiento dentro de la página.

Ventajas de segmentación paginada:

El esquema de segmentación paginada tiene todas las ventajas de la segmentación y la paginación:

- Debido a que los espacios de memorias son segmentados, se garantiza la facilidad de implantar la compartición y enlace.
- Como los espacios de memoria son paginados, se simplifican las estrategias de almacenamiento.
- Se elimina el problema de la fragmentación externa y la necesidad de compactación.

Desventajas de la segmentación paginada

- Los tres componentes de la dirección y el proceso de formación de direcciones hace que se incremente el costo de su implantación. El costo es mayor que en el caso de de segmentación pura o paginación pura.
- Se hace necesario mantener un número mayor de tablas en memoria, lo que implica un mayor costo de almacenamiento.

- Sigue existiendo el problema de fragmentación interna de todas -o casi todas- las páginas finales de cada uno de los segmentos. Bajo paginación pura se desperdicia sólo la última página asignada, mientras que bajo segmentación paginada el desperdicio puede ocurrir en todos los segmentos asignados.

b.- Fallos de pagina. Definicion. Como se realiza la detección y son solucionados.

Un fallo de pagina ocurre cuando un proceso intenta usar una dirección que esta en una pagina que no se encuentra en la memoria principal. Bit V=0

- La pagina no se encuentra en su conjunto residente (working set)
- El bit V es controlado por el HW

El HW detecta la situación y genera un trap al SO
El SO pondrá colocar al proceso en estado de "Waiting" mientras se gestiona que la pagina que se necesite se cargue
El SO busca un marco libre (**asumiendo que hay, si no hay que reemplazar pagina**) en la memoria y genera una operación de E/S al disco para copiar en dicho frame la pagina 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 una interrupción su finalización

Cuando la operación de E/S finaliza, se notifica al SO y este:

- Actualiza la tabla de paginas del proceso
    - Coloca el Bit V en 1 en la pagina en cuestión
    - Coloca la dirección base del frame donde se colocó la pagina

El proceso que generó el fallo de pagina vuelve al estado de Ready
Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes generó el fallo de pagina.

Si todos los marcos están ocupados:
En el proceso previo, si no hay marcos disponibles en memoria debemos seleccionar una pagina victima.
El reemplazo optimo seria que la pagina a ser removida no sea referenciada en un futuro próximo, pero como este algoritmo es imposible de implementar ya que el futuro no lo sabemos la concha de tu hermana, la mayoría de las tecnicas de reemplazo predicen el futuro mirando el compartamiento pasado. Ejemplos de estos algoritmos son:

- OPTIMO (quitar de memoria la pagina que no será utilizada en mas tiempo, algoritmo teorico)
- FIFO (primera pagina entrar primera en salir)
- LRU (quita las paginas menos usadas recientemente)
- 2da CHANCE (mejora de fifo pero utilizando bit de presencia como "respaldo")

Reemplazo global:
El fallo de pagina de un proceso puede reemplazar la pagina 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 pagina de un proceso solo puede reemplazar sus propias paginas
El SO puede determinar cual es la tasa de page-faults de cada proceso
No cambia la cantidad de frames asignados
Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos

Además de esto, podemos asignar marcos a los procesos de forma dinámica o fija:

- Asignación dinámica: el numero de marcos para cada proceso varía
- Asignación fija: numero fijo de marcos para cada proceso.
    - Forma equitativa: 100 frames y 5 procesos, 20 para cada uno
    - Forma proporcional: se asigna acorde al tamaño del proceso

c.- Hiperpaginacion (Thrashing): definición, formas de detectarla y tratarla

La hiperpaginacion se da cuando un sistema pasa mas tiempo paginando (cargando y descargando paginas entre memoria principal y área de intercambio) que ejecutando procesos, y como consecuencia hay una baja importante en la performance del sistema.
Si un proceso contara con todos los frames que necesita, no habría thrashing.

Ciclo de thrashing:

1) El SO monitorea el uso de la CPU
2) Si hay baja utilización → aumenta el grado de multiprogramación(por el long term scheduler)
3) Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos
4) Si un proceso necesita mas frames, comienzan los page-faults y robo de frames a otros procesos
5) Por swapping de paginas, baja el uso de del CPU
6) Vuelve a 1

Control de thrashing

Podemos detectar el thrashing, basándonos en el comportamiento del pasado reciente de un programa, para predecir que instrucciones y datos se utilizaran en un futuro próximo.
Podemos aumentar la cantidad de memoria ram (mejor solución a largo plazo)
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.
También existen tecnicas como la estrategia de **Working Set, Modelo de Localidad** y la **Estrategia de PFF (frecuencia de fallo de pagina)**

Modelo de localidad

Mejor conocido como principio de cercanía de referencias.
Las referencias a los datos dentro de un proceso tienden a agruparse.
Para prevenir hiperactividad, un proceso debe tener en memoria sus paginas mas activas.

- Localidad temporal: si una posición de memoria es referenciada, es muy probable que la misma posición vuelva a ser referenciada en un futuro cercano. En este caso es comun almacenar una copia de los datos referenciados en cache para un acceso rápido
- Localidad espacial: si una posición de memoria es referenciada en un momento dado, es probable que las posiciones cercanas a ella sean también referenciadas pronto.
- Localidad secuencial: las direcciones de memoria que se están utilizando suelen ser contiguas, esto se da a que se ejecutan secuencialmente.

Modelo de Working Set
Basado en el modelo de localidad.
Si el numero de marcos asignados a un proceso desciende por debajo del minimo, debemos suspender la ejecución de ese proceso, luego debemos descargar sus paginas restantes liberando los marcos asignados. Esto se da porque cualquier proceso que no cuente con los marcos suficientes provocara fallos de pagina muy frecuentemente. (el proceso requiere una pagina, se la trae por pagefault, necesita otra pagina y como tiene pocos marcos, otro pagefault, asi sucesivamente, a causa de tener sus marcos por debajo de su minimo).

Estrategia de page-fault
PFF → frecuencia de page-fault
PFF alta → se necesitan mas frames
PFF baja → los procesos tienen muchos frames asignados
Establecer limites superior e inferior de las PFF
Excede PFF max → le doy 1 frame mas
Debajo de las PFF minimo → le saco frame
Puede llegar a suspender un proceso si no hay mas frames. Sus frames se reasignan a procesos de alta PFF

3)Sistema de archivos, E/S y Buffer Cache

a.- Describa la estructura del sistema de archivos de UNIX System V ¿Si quisiera **CREAR (**te pueden decir abrir o borrar tambien) un nuevo archivo, por ejemplo /home/usuario/final.txt, que estructuras de las indicadas son utilizadas y modificadas?

Mas información en → http://www.ual.es/~acorral/DSO/Tema_4.pdf

Un sistema de archivos permite realizar una abstracción de los dispositivos físicos de almacenamiento para que sean tratados a nivel lógico.
El sistema de archivos de UNIX se caracteriza por:

- Poseer una estructura jerarquica
- Poder crear o borrar archivos
- Tratamiento consistente de los datos
- Crecimiento dinamico de los archivos
- Tratar a los dispositivos periféricos como archivos

El sistema de archivos de UNIX System V reside en un único disco lógico o partición de disco y se compone de los siguientes elementos:

- Boot block → contiene el código requerido para arrancar el sistema operativo
- Superblock → contiene atributos e información sobre el sistema de archivos, tal como el tamaño de la partición y el tamaño de la tabla de i-nodos
- I-NODO → estructura de datos que contiene la información clave de un archivo
- I-NODE table → tabla que contiene todos los i-nodos
- Datablocks → bloque de datos de los archivos

create()

Tiene la misma funcionalidad que open

Si el archivo no existe, se crea uno nuevo. Si ya existe trunca su contenido.

A) obtener el i-nodo a partir del pathname

B) si el archivo no existía → asignar el i-nodo libre y crear la entrada en el directorio padre si es que tenemos permisos de escritura en el directorio padre (/home/usuario)

Si el archivo ya existía:

- Si el acceso no esta permitido → retorna error
- Si el acceso esta permitido → truncar el archivo si tenemos permisos de escritura sobre el archivo y liberar todos sus bloques de datos

En cualquier caso:

C) asignar una entrada en la tabla de archivos

D) asignar una entrada en la tabla de descriptores de archivos

ABRIR un archivo

open()

Permite el acceso a datos del archivo

A) acceso al i-nodo → traer copia a memoria

B) el kernel busca en el sistema de archivos el archivo pasado como parámetro

C) si el archivo no existe o no se tienen permisos → ERROR

D) obtiene copia del i-nodo en memoria

E) asigna una entrada en la tabla de archivos

- Puntero al i-nodo + desplazamiento = tamaño del archivo

F) si modo es igual a O_TRUNC, entonces liberar los bloques (disco) del archivo y establecer el tamaño del archivo (en inodo) a 0

G) asignar una entrada en la tabla de descriptores de archivos de usuario

ABRIR un archivo "/etc/password" (otra explicación mas textual, revisar si esta bien o no o si es lo que realmente se pide)

Cuando el kernel emplea el análisis del nombre, encuentra que es "/" y pasa a leer el inodo asociado al directorio raíz. Este inodo pasa a ser el inodo del trabajo y el kernel verifica si corresponde a un directorio, y si el proceso tiene permisos para buscar archivos dentro de él suponiendo que los permisos están en regla, dentro del directorio raíz busca el siguiente elemento que toma el kernel "etc". En este inodo refleja que "etc" es otro directorio y por

lo tanto en él podemos buscar el archivo “passwd”, y tras encontrarlo y teniendo permisos para acceder a el, nos habilita una estructura para poder trabajar con ese archivo

LEER un archivo
read()

A) Obtener la entrada en la tabla de archivos a partir del descriptor del archivo
B) Comprobar los permisos
C) Obtener el inodo siguiendo los punteros
D) Hasta el fin de la lectura o error o no queden bytes en el archivo:

- Desplazamiento + nro de bloque de disco
- Calcula el desplazamiento en el bloque para comenzar la E/S
- Calcular el numero de bytes a leer dentro del bloque
- Si el numero de bytes a leer = 0 → fin de la búsqueda
- Si no:
    - Leer el bloque en un buffer (buffer caché)
    - Copiar los datos del buffer + dirección del proceso destino del usuario
    - Actualiza el desplazamiento en el archivo, dirección en el proceso de usuario, nro de bytes que queda por leer
- Actualizar el desplazamiento en la tabla de archivos

BORRAR un archivo (FALTA IMPLEMENTAR)

b.- Relacione entre E/S, sistema de archivos y buffer cache utilizando la SystemCall (read) (explicando los 3 temas haciendo de cuenta que queremos leer un archivo en disco, por eso systemcall read) (se puede utilizar la systemcall read explicada arriba, pero aca están relacionados los 3 temas)
Determinar el dispositivo que almacena los datos.
Traducir el nombre del archivo en la representación del dispositivo
Si el nombre (pathname) es absoluto, la búsqueda del inodo del archivo se iniciara en el directorio raíz. Si el nombre es relativo, la búsqueda se iniciara en el directorio del trabajo actual que tiene asociado el proceso que va a acceder al archivo.
A medida que se van recorriendo los inodos intermedios, se van verificando los permisos para comprobar si el proceso tiene derecho a acceder a los directorios intermedios.
Una vez que encontramos el inodo, conociendo el numero de inodo y el dispositivo lógico asociado, (ya que el sistema operativo trata a los dispositivos de E/S de forma logica ocultando sus detalles a tareas de mas bajo nivel con el hecho de simplificarlas) calculamos en que bloque de disco esta. Una vez que tenemos el bloque de disco, acomodamos el cabezal del disco en la pista y sector correspondientes al archivo que buscábamos.
A todo esto cada dispositivo tiene asignado un buffer (espacios en memoria principal para el almacenamiento temporario de sectores de disco con el objetivo de minimizar los accesos a disco).

Los buffers en memoria principal están ordenados por una lista que se maneja por LRU, donde cuando se necesita cargar un nuevo bloque a memoria, se elije el buffer que hace mas tiempo no es referenciado
Si buffer que busca el proceso esta en el buffer cache, el kernel lo devolverá inmediatamente sin necesidad de realizar una lectura física en disco, y al buffer le asignamos el sector de disco que acabamos de buscar.
Si no está el buffer en el buffer cache, se realiza lo siguiente:

- El kernel realiza una petición de lectura al controlador del disco
- El proceso se duerme por un evento de E/S
- La CPU puede ser otorgada a otro proceso mientras se realiza la E/S (ya que utilizamos la técnica de E/S manejada por interrupciones, en la que se soluciona el problema de espera de la CPU que tenia la técnica de E/S programada, en la que la CPU chequeaba repetidamente el estado del dispositivo ("**Polling**") para ver si había terminado su operación, con interrupciones la CPU puede seguir ejecutando instrucciones mientras se completa la operación de E/S y cuando esta finaliza se interrumpe al procesador para avisarle que terminó y continuar la ejecución de su anterior proceso desde su ultima instrucción)
- El kernel informa al controlador de disco que quiere leer datos
- El controlador de disco trasmite los datos del disco al buffer
- Finalmente, el controlador del disco interrumpe al procesador cuando la E/S se ha completado
- El gestor de interrupciones puede despertar al proceso que dormia o a otro proceso, dependiendo el algoritmo de planificación.
- Ahora el bloque de disco ya esta en el buffer cache para que pueda ser usado

Los buffers están formados por el Header y el lugar donde se almacena el bloque de disco
El header se identifica por Nro de dipositivo y Nro de bloque
Tiene 2 punteros:

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

El header tambien posee un estado, que puede ser:

- Free o disponible
- Busy o no disponible (en uso por algún proceso)
- Kernel leyendo o escribiendo en disco
- Delayed Write (buffers que hayan sido modificados en memoria, pero el bloque original en disco todavía no fue actualizado)

La Free List organiza los buffers disponibles, es decir los buffers donde puede cargarse un nuevo bloque de disco, y se ordena por LRU

Hash queues
Son colas para optimizar la búsqueda de un buffer en particular
Se organizan según su función de hash usando (#dispositivo, #bloque)