Teoria

1. **Indique ventajas y desventajas de utilizar paginación en varios niveles. Explique la importancia de usar una TLB en este tipo de esquemas.**

Con paginación jerárquica podemos paginar la tabla de páginas, de esta forma ya no necesitamos tener la misma contigua en memoria, sino que traemos a memoria física únicamente las páginas de la tabla que necesitamos. De esta forma, sin importar que el tamaño total de las tablas sea mayor a usar una única tabla, lo que realmente cargamos en RAM para tablas de páginas es menor, permitiendo así cargar más páginas de un proceso, o más procesos, es decir, aumentar el nivel de multiprogramación
La mayor desventaja de este esquema es que por cada nivel de indirección necesitamos ir a una nueva tabla de páginas, es decir, un acceso más a memoria, que podría convertirse en un acceso a disco si es que la misma no se encuentra aún en memoria física. Esto hace que el acceso a memoria sea más lento. Por lo tanto, con una TLB evitamos estas indirecciones ya que con acceder a esta caché, ya sabemos en qué frame se encuentra una página.

2. **Escriba un ejemplo en pseudocódigo donde una o más operaciones provoquen thrashing cuando: a) se utiliza asignación fija y sustitución local, b) se utiliza asignación variable y sustitución global.**

Pueden mostrar cualquier ejemplo en el que se vea que un proceso necesite cierta cantidad de páginas simultáneamente cargadas en memoria para realizar una operación y que se le asigne una cantidad menor de frames. No confundir en otros casos en los que se necesiten más páginas pero no simultáneamente, ya que en dicho caso se podría ir reemplazando las páginas sin problema.
Apunta a que un proceso para no entrar en sobrepaginación necesita tener una cantidad de frames suficiente para poder acomodar su localidad, es decir, las páginas que utiliza activamente,
Para el b) es similar, la diferencia radica en que en este caso la cantidad total de páginas requeridas por todos los procesos va a ser menor a la cantidad de frames disponibles para los mismos, por lo que irán robándose frames unos a otros.

3. **Compare las entradas salidas programadas, por interrupciones y por DMA, usando como criterio su duración y los cambios de contexto que provocan.**

Las IO programadas no requieren ningún cambio de contexto ya que es la CPU la que se encarga de realizar todas las peticiones al módulo de IO, de consultar su estado y de copiar los datos a memoria. Por lo tanto, este tipo de IO es la que menos tiempo dura, pero desperdicia mucho tiempo de la CPU, por lo que son recomendables sólo para realizar IOs de corta duración.
Para realizar IOs largas, es mejor usar cualquiera de las otras técnicas (siendo DMA la más eficiente). Entre ambas, DMA tiene menos cambios de contexto ya que este HW lanzará una única interrupción recién al finalizar por completo la IO. Por otro lado, las IOs por interrupciones generarán un número mayor de interrupciones (cada vez que el módulo le avise a la CPU que se llenó el buffer y debe copiar datos a memoria) y por lo tanto más cambios de contexto.

4. **Indique V/F Justifique**
    a. **En un FS de tipo FAT los hardlinks no se pueden realizar sobre directorios.**

F. Los hardlinks no existen en un FS de tipo FAT,,

    b. **En un FS de tipo EXT2 el tiempo de acceso para un archivo suele ser menor que en un EXT por su división en grupos.**

V. Al dividir el volumen en grupos, en el que cada uno tiene bloques de datos y estructuras para administrar dichos bloques; permite que un archivo no tenga sus datos "tan dispersos" ya que se trata de asignar los bloques pertenecientes a un grupo de bloques. De esta forma, los bloques de un archivos estarán "más cerca" físicamente en disco, requiriendo menos reposicionamiento del cabezal, disminuyendo así el tiempo de acceso.

5. **Explique por qué es conveniente que una partición de swap utilice asignación de bloques contigua**

Como lo que tenemos que almacenar en swap son páginas, necesitamos que las operaciones de lectura/escritura sean lo más

eficientes posibles, ya que esto afectará al tiempo de acceso a memoria. De entre todas las técnicas de asignación vistas, la contigua es la que agrega menos overhead.

Práctica

MEMORIA

Luke, para convertirse en jedi, comenzó a aprender un poco sobre memoria , diseñando un sistema con paginación bajo demanda (páginas de 1KiB), asignaciòn de frames fija y sustituciòn local (utilizando el algoritmo clock-modificado para lo mismo). “Usar TLB, tú debes” le dijo un día Yoda, por lo que Luke decidió experimentar con una caché de 20 entradas con únicas columnas pág - marco. Colocó a correr dos procesos P1 (se le asignan 3 frames) y P2 (se le asignan 2 frames) y luego de un tiempo el estado para cada proceso era el siguiente:

P1

frame	pàg	u	m
22	5	1	1
30	10	1	1
35	-		

Se acaba de leer la pág 10 de P1

Nota: la siguiente víctima para P2 es 0

P2

frame	pàg	u	m
10	**0**	0	1
50	100	1	1

TLB

pág	frame
10	30
...	...

Si luego se realizan las siguientes referencias a memoria(lecturas): P2 3111 - P2 501 - P1 500 - P1 2000 - P1 6001 -P1 2001 - P2 522 - P2 3200 - P1 300. Ayude a Luke e indique:

a. Para cada referencia: cantidad de accesos a memoria y a la TLB
b. La dirección física generada por la última referencia
c. ¿El comportamiento de los procesos (el tipo de referencias que generan) cumple el concepto que fundamenta el uso de las TLBs?
d. ¿Cómo podría mejorar la eficiencia los accesos en este caso? (Tip: se pueden considerar cambios en la TLB, en la asignación de frames, en el algoritmo de sustitución, etc)

a) Las páginas son de 1024 bytes, por lo que si dividimos DL / Tam Pág = obtenemos el nro de pág accedido

P2 3111 -> pág 3 al cambiar de proceso se limpia la TLB . Se accede a la TLB para buscar pàg 3, que no encuentra, por lo que luego accede a la TP, se genera un PF, se elige como vìctima la pàg 0 (se descarga a disco). Se accede a memoria para ir a buscar la pàg 3 y para actualizar la TP. Tambièn se actualiza la TLB.

P2 501 -> pàg 0 . Se accede a la TLB para buscar la pàg, luego a la TP, se genera PF y se reemplaza la pàg 3. En este caso no es necesario descargar la pàg ya que no se modificó. Se carga la pág 0 y se actualiza la TLB y la TP.

P1 500 -> pág 0 . Se accede a la TLB (previamente vaciada), a la TP, se genera PF, se carga la pàg 1 en el frame que todavía está vacío (35). Se actualiza la TP y la TLB.

P1 2000 -> pàg 1 .Se accede a la TLB, a la TP, se genera PF. Se elige la pág 0 como víctima (que no es necesario descargar). Se carga la pág 1 , actualizando la TP y la TLB.

P1 6001 -> pág 5 . Se accede a la TLB, a la TP, se actualiza la TLB:

P1 2001 -> pág 1. Se accede a la TLB -> se obtiene el marco

P2 533 -> pág 0 Se accede a la TLB (vacía), a la TP, se actualiza la TLB

P2 3200 -> pág 3 Se accede a la TLB, a la TP., se genera un PF. Se reemplaza la pág 100, la cual hay que descargar. Se carga la pág 3 y se actualiza la TP y la TLB.

P1 300 -> pàg 0 . Se busca en la TLB, en la TP, se genera PF, se elige como víctima a 10, por lo cual hay que descargarla . Luego se carga 0, y se actualizan la TP y la TLB.

b) 300 = pág 0 offset 300 -> frame asignado 30 => DF = 30 * 1024 + 300 = **31020**
c)
Analizando..
P1
tenía cargado en memoria las págs 5-10
y luego referencia : 0 - 1 -5 -1 -0

P2
tenía cargado en memoria
0 - 100
y luego referencia 3- 0 -0 -3

Por lo tanto, vemos que estos procesos no están referenciando siempre a distintas págs , sino que en ese intervalo de tiempo parecerían usar el mismo conjunto de págs. Entonces se podría decir que cumple el principio de localidad (temporal)

d)Lo que vemos es que al vaciar la TLB nunca se llega a usar la entrada que acabamos de cargar. Una solución sería que nuestra caché tenga el campo Proceso/PID de forma que podamos identificar a qué proceso pertenece cada traducción. De esta forma no sería necesario vaciar la caché en cada process switch.
Por otro lado, si se asigna un frame más a cada proceso , se generarían menos PFs y serían necesarias menas operaciones de disco.

FS
El joven anakin un día le prestó su notebook a Palpatine, luego, al listar los archivos de su FS de tipo Unix vio lo siguiente:

/**anakin ls -li** 17666347 drwxr-xr-x 3 askywalker 6 Nov 20 22:45 jedi_course 17666676 -rw-r--r-- 2 askywalker 6 Nov 20 22:44 salvar_padme 17666343 drwxr-xr-x 3 askywalker 6 Nov 20 22:43 sith_course 17667204 lrwxr-xr-x 1 askywalker 6 Nov 20 22:48 the_force -> jedi_course/the_force_tutorial **/jedi_course ls -li** 17666973 -rw-r--r-- 1 askywalker 6 Nov 20 22:45 the_force_tutorial **/sith_course ls -li** 17666676 -rw-r--r-- 2 askywalker 6 Nov 20 22:44 the_dark_side_of_the_force_tutorial	**Contenido Inodo 17666976** Atributos (...) 33 12 199 122 148 - **Contenido Bloque 148** 333 222 999 -

Si se sabe que los bloques son de 4KiB, los inodos tienen 3 pts directos, 2 ind simples y 1 ind doble y las direcciones son de 32 bits:

a. ¿Qué ocurrirá cuando Anakin abra “the_force” y “salvar_padme”? Indique para cada caso qué pasos se seguirán para leer dichos archivos.
b. Indique cuál archivo es de tipo directorio, cuál de tipo regular (si tiene o no algún hardlink) y cuál de tipo softlink. Justifique en cada caso.
c. ¿Qué tamaño tiene asignado el archivo “salvar_padme”? ¿Cuántos accesos a bloques serán requeridos si se le quiere agregar al archivo 3000 bloques de datos?

a- cuando abra “the_force” se irá a buscar el inodo 17667204. El contenido de dicho archivo será “jedi_course/the_force_tutorial”. Luego se buscará ese path hasta llegar al inodo 17666973. Se irá a buscar dicho inodo y se leerán sus bloques de datos.

Cuando se abra “salvar_padme” se irá a buscar el inodo 17666676, se leerán sus bloques de datos. (que son los mismos que de “the_dark_side_of_the_force_tutorial”)
b- directorio: jedi_course -> vemos que el primer bit dentro de los permisos es “d” , lo cual indica que es un directorio. También vemos que después se listan sus entradas.
archivo regular: Por ejemplo “the_dark_side_of_the_force_tutorial”, su primer bit es “-”, por lo que no es ni un directorio ni un softlink. vemos que tiene dos referencias, siendo la otra “salvar_padme”, son hardlinks. (vemos que tienen igual inodo, permisos y timestamps)
softlink: “the_force”. Vemos que su tipo es “s” , y también se ve gráficamente con la flechita -> . Vemos que tiene distinto inodo y propiedades que el archivo al que apunta, “the_force_tutorial”

c- Por lo que vemos, se están usando 5 ptrs del inodo, estos son , los 3 directos y dos ind simples.
Cada bloque de punteros puede tener = 4KiB / 32 bits = 1024 ptrs
Sabemos que se usó el 2do ptr ind simple significa que se usó todo el primer bloque de ptrs de 1er nivel. Sin embargo, no podemos estar seguros de que se haya usado todo el 2do. Si vemos el contenido del bloque 148 (el 2do bloque de ptrs) vemos que sólo se usaron 3 ptrs.
Finalmente , el archivo tiene = 3 (ptrs directos) + 1024 (1er ptr ind) + 3 (2do ptr ind) = **1030 bloques de datos = 4,...MiB**

Ahora se quieren agregar 3000 bloques de datos. Los primeros los vamos a ir referenciando con los ptrs que nos quedaron sin usar del 2do bloque de ptr. (1021 ptrs)

Por lo tanto , me quedan 3000 - 1021 = 1979 que tenemos que referenciar con el ptr doblemente indirecto
Vamos a necesitar el primer bloque de puntero (que apunta a bloques de ptrs) + 2 bloques de punteros de 2do nivel (para direccionar los 1929 restantes, ya que no me alcanza con uno solo).

Finalmente, vamos a escribir = 3000 BD + 1 BP (del ptr de 1er nivel, que ya estaba asignado de antes) + 1 BP (primer nivel, apuntado por el ptr doblemente indirecto) + 2 BP (de segundo nivel) = 3004 Bloques

IO

Un disco de 100 pistas, cuyo tiempo entre pistas es de 1u, es planificado con diferentes algoritmos. El brazo se encuentra en la pista 50 y llegan diferentes pedidos en los instantes 0 y 40: T=0u → 60-70-90-40 | T=40u → 10-20-30

a. Indique cuál sería el orden de atención de los siguientes pedidos, usando F-SCAN
b. Indique cuál sería el orden de atención de los siguientes pedidos, usando C-SCAN
c. Indique cuál sería el orden de atención de los siguientes pedidos, usando C-LOOK

a- FSCAN
Todos los pedidos que llegan en 0 van a una cola de atención, el resto van a la cola de espera
Atiende: 50 | 60 -70 -90 -100 -40 -30 -20 -10

C-SCAN
Atiende. 50 | 60 - 70 - 90 -100 - 0 - 10 -20 -30 -40

C-LOOK
Atiende : 50| 60 -70 -90 - 10 -20 -30-40