Sistemas Operativos
2° Parcial 1C2023 - TM - Resolución

<u>Aclaración</u>: La mayoría de las preguntas o ejercicios no tienen una única solución. Por lo tanto, si una solución particular no es similar a la expuesta aquí, no significa necesariamente que la misma sea incorrecta. Ante cualquier duda, consultar con el/la docente del curso.

Teoría

1-

Espacio en disco: El esquema de permisos Unix ocupa poco espacio. ACL ocupa mucho más espacio (por tener un permiso por cada usuario).

Granularidad: El esquema de permisos Unix tiene poca granularidad. ACL tiene mucha granularidad (permite especificar diferente el permiso de cada usuario).

Copiar de EXT2 a FAT32 implicaría perder los permisos ya que FAT32 no posee un esquema de permisos por archivo. Al revés sería necesario setear algunos permisos por defecto, dado que no existirían en FAT32.

2- Fragmentación externa no se sufre, y fragmentación interna sigue existiendo en segmentación paginada por usar páginas, que representan asignación fija de espacio. En todo caso esta fragmentación es mayor debido a sufrirse en cada segmento.

3

a. Falso, los hardlink no se pueden realizar entre archivos de diferentes volúmenes ya que el mismo es una nueva entrada de directorio apuntando al número de inodo del archivo original, y cada volúmen, aún siendo del mismo tipo de File System, tiene sus propios inodos.
b. Falso, la ventaja que posee paginación jerárquica sobre paginación regular es que requiere de menos espacio para sus tablas de páginas, ya que utiliza una sola tabla indexada por número de marco vs una tabla de páginas por cada proceso en paginación regular.

4- Como máximo se podrían considerar 3 accesos a memoria principal si asumimos que la tabla de segmentos no se carga en registros/cache (1 acceso tabla de segmentos + 1 acceso tabla de páginas + 1 acceso al marco), con paginación jerárquica tendríamos más accesos a tablas de páginas según la cantidad de niveles.

5- No tendría sentido dado que no existe fragmentación externa. Lo único que podría llegar a mejorar es el tiempo de búsqueda de cada bloque dado que los mismos quedan contiguos.

Práctica

1)
Parte a)
Dado que la fragmentación máxima es tamaño de cluster - 1 byte, entonces concluimos que cada cluster es de 4096 bytes.
Considerando que el directorio no fue leído, se deberá considerar una lectura del mismo, el cual ocupa 1 cluster.

Primero es necesario leer el directorio “carpeta1”, el cual comprende 1 acceso a la FAT y 1 cluster.
Luego, de a.txt, se desea leer desde el cluster 10/4096 = 0, hasta el cluster 5000/4096 = 1, por lo tanto son 1 - 0 + 1 = 2 clusters. Para recorrerlos se necesitan 2 accesos a la FAT (memoria) también.
En total:

- 1 lectura FAT (directorio) + 2 lecturas FAT (a.txt) = 3 accesos a la FAT.
- 1 lectura cluster (directorio) + 2 lecturas cluster (a.txt) = 3 accesos a disco.

Para b.txt, no es necesario leer de disco el directorio “carpeta1” nuevamente, dado que ya se encuentra en el buffer de memoria. Luego, se desea leer desde el cluster 59023/4096 = 14, hasta el cluster 140875/4096 = 34, por lo tanto son 34 - 14 + 1 = 21 clusters.
Para recorrerlos, se necesitan 14 accesos a la fat para llegar a la ubicación del cluster 14, y luego 21 accesos más para recorrer los restantes, dando un total de 14 + 21 = 35 accesos a la fat (memoria).

Para cada uno de estos casos podría considerarse válido 1 acceso menos a la FAT dado que el primer cluster ya se conoce desde la entrada de directorio y el último bloque se conoce leyendo la entrada de la FAT del bloque anterior.

Parte b)
Al truncar ambos archivos a 100 bytes, ambos quedan con un solo cluster asignado.
Por lo tanto, se deben marcar (escribir) todos los punteros de la FAT desde el 2do puntero hasta el último de cada archivo, con algún valor que indique que el cluster está libre. No es necesario escribir (“borrar”) en disco los clusters.
Para a.txt, se escribe un solo puntero en FAT (dado que tiene 2 clusters).
Para b.txt, se escriben 34 punteros en FAT (dado que tiene 35 clusters).

Se podrían considerar también 1 puntero más en cada caso dado que la primera entrada contendría “EOF”.

2)
Lo primero es determinar cant de bits para #pág / offset

Si observamos la TLB veremos que se necesitan 16 bits para nro página (4 caracteres hexa) y 8 bits para cada nivel. Esto se corrobora también basándonos en la info de que la dirección 02013333h no produce ninguna interrupción (PF) lo cual indica que está presente en TP y en este caso también en TLB. Pág 1er nivel es: 02 que está presente y pág del 2do nivel: 01 marco 14 está presente.

Vamos a indicar las DL del proceso A

DL	Pág 1er Nivel	Pág 2do nivel	PFs	Accesos TLB	DF
02031010h	02 - presente	03 - presente	No	1 (miss)	111010h
04021111h	04 - presente	02 - presente	No	1 (hit)	151111h
02002424h	02 - presente	00 - PF	Si	2	102424h
01044444h	01 - PF	04 - PF	Si	2	1F4444h

02031010h está en memoria pero no en la TLB por lo cual no hay PF pero se actualiza la TLB (pág 0203, marco 11) reemplazando la entrada con el marco 3A.

04021111h está en la TLB por lo cual no hay PF y no reemplazamos en TP ni TLB.

02002424h no está en TLB, está presente en el primer nivel pero no en el 2do por lo que genera PF, se le asigna el marco 10 (libre) y se reemplaza en TLB teniendo en cuenta que a 0402 la acabamos de referenciar.

01044444h no está en la TLB, no está presente en 1er nivel, genera PF y se carga la tabla de 2do nivel (pág 1) en el marco 1B (libre). Luego asumimos que ninguna de sus páginas está presente por lo que se le asigna el marco 1F (libre) y se actualiza finalmente la TLB.

TPA 1er Nivel

Pág	Marco	Bit P
0	1A	1
1	1B	1
2	1C	1
3	1D	1
4	1E	1
...		

TPA 2do Nivel - Pág 1

Pág	Marco	Bit P
...		
4	1F	1
...		

TPA 2do Nivel - Pág 2

Pág	Marco	Bit P
0	10	1
1	14	1
2	12	1
3	11	1
4	11	0
...		

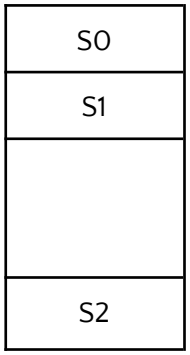
TPA 2do Nivel - Pág 4

Pág	Marco	Bit P
0	13	1
1	15	0
2	15	1
3	13	0
4	16	1
...		

TLB

Pág	Marco	Proceso
0104	1F	A
0203	11	A
0402	15	A
0200	10	A

3)



0611h
0000 01 | 10 0001 0001
-> Para que corresponda al segmento 1 necesito tomar 10 bits como offset, por lo que me quedaría un offset de 529 (512+16+1), al ser la última dirección válida significa que su tamaño es 530. Luego, el segmento 0 tiene el mismo tamaño.

Al tener 10 bits para offset en la dirección, el máximo tamaño posible para un segmento es 2^10 B = 1024 bytes.

Por lo tanto la tabla de segmentos nos queda (en decimal):

a)

Seg	Base	Tamaño
0	0	520
1	520	520
2	3072	1024

b) No tendría sentido compactar en este momento ya que toda la memoria libre está contigua.

c) Teniendo bloques de 1 KiB == 1024 B, si estoy parado en el byte 15000 y quiero leer 1000, tendría que leer desde el byte 15000 al 15999 -> 15000/1024 = 14,6..., 15999/1024 = 15,6... es decir, estoy buscando leer los bloques de datos 14 y 15, los cuales están direccionados a través del bloque de punteros (indirecto simple) ya que tengo solamente 12 punteros directos.
Por lo tanto, necesito realizar **3 accesos a bloques** (1 BP(s) + 2 BD).

d) Tam máx arch =- 256^2 * 1KiB = (2^8)^2 * 2^10 B = 2^16 * 2^10 B = 2^26 B = **64MiB**