Sistemas Operativos
1° Rec 1er Parcial 2C2023 - 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.
    a. Verdadero, La traducción en tiempo de ejecución permite, que luego de un page fault, una página ubicada en memoria virtual pueda asignarse a cualquier marco libre.
    b. Verdadero. La función de la TLB es facilitar la traducción de la dirección lógica a dirección física. Cuando se utiliza paginación jerárquica y ante un fallo de TLB, la traducción de la página puede implicar acceder a porciones de la tabla de páginas que no se encuentran en memoria, generando de esta manera un page fault.

2. El proceso A podría tener una entrada en su tabla de páginas que diga: "página:1, frame:3", y otro proceso podría tener en su tabla de páginas una entrada que diga "página 4, frame: 3". Tiene como ventajas que es una mecánica simple, donde no es necesario agregar estructuras adicionales.

3. En la lista de archivos abiertos por proceso podría tener algunos valores como por ejemplo el nombre del archivo, modo de apertura, el puntero en el archivo, entre otros campos, mientras que en la lista de archivos abiertos global, tenga muy seguramente campos como el nombre del archivo, contador de aperturas, si tiene locks, y los permisos del mismo.

Lista de archivos abiertos global		**Lista de archivos abiertos P1**		**Lista de archivos abiertos P2**	
nombre	Cantidad aperturas	nombre	Modo apertura	nombre	Modo apertura
\usr\lala.txt	2	\usr\lala.txt	R	\usr\lala.txt	R

4. En la protección de archivos mediante ACL se genera por cada archivo una lista en la cual se indica en cada nodo de la misma que usuarios tienen acceso al archivo y que tipo de operaciones puede realizar (lectura, escritura, ejecución), mientras que en un esquema tipo UFS, en cada archivo se definen los permisos para el propietario del archivo, para un grupo de usuarios y para el resto de los usuarios. Una ventaja del esquema ACL sobre el utilizado en UFS, es que es más granular, si por ejemplo quisiera generar un archivo con permisos para algunos usuarios nada más, simplemente los agrego a la lista indicando sus permisos, si quisiera hacer lo mismo en UFS debiera (seguramente) crear un grupo nuevo para incluir en este a los mismos usuarios y asignarle los permisos al mismo.
5. En un filesystem tipo FAT no existe una estructura administrativa dedicada a los

bloques libres, pero estos se marcan en la **TABLA** FAT, donde se indican con ciertos valores el "estado" del bloque (libre,fin, error, siguiente bloque), mientras que en EXT2, existe un bitmap de bloques libres, donde cada posición de este array representa un bloque, indicando con 0 si el bloque se encuentra libre o con 1 caso contrario. Respecto a creación de archivos con bloques libres disponibles, en FAT no tenemos alguna limitante por sus estructuras administrativas (dado que la única estructura administrativa propia del FS es la tabla fat), por otro lado, en EXT2 podríamos tener bloques libres y estar imposibilitados de crear archivos nuevos, esto ocurre si todos los inodos disponibles en el fs se encuentran ocupados (puede verse claramente en el bitmap de inodos).

Práctica

1) Datos: FS: Ext2, bloques de 4KiB, punteros de 8 bytes, Inodos: { 12 PD, 1 IS, 1 ID, 1 IT }

Calculo cuantos punteros entran en un bloque: 4KiB / 8 bytes = 2^12 / 2^3 = 2^9 = 512
Con esta configuración de inodos puedo definirme para mayor comodidad los rangos de bloques de datos que puedo acceder con cada tipo de puntero:

12 PD -> BD #0 al BD #**11**
1 IS -> BD #12 al # (**512** + **11**) -> BD #12 al BD #**523**
1 ID -> BD #524 al # (**512^2** + **523**) -> BD #524 al BD #**262667**
1 IT -> BD #262668 al # (**512^3** + **262667**) -> BD #12 al BD #**134480395**

a) Leer **desde** el byte 0 al 250180 de un archivo:
Byte 0 -> Bloque #0,
Byte 250180 -> (250180 / 4096 = 61,07....) -> Bloque #61
Debo leer 62 bloques de datos, los primeros 12 bloques los leo con PD, por lo que:
62 - 12 = 50 bloques de datos restantes, y dado que con el IS puede llegar a el bloque #61, los restantes los leeré mediante este puntero indirecto simple, por lo tanto los accesos a bloques son:
62 BD + 1 BP (IS) = 63 Bloques

b) Leer el byte 134217738 de un archivo:
Byte 134217738 -> (134217738 / 4096 = 32768,002...) -> Bloque #32768
Este número de bloque está en el rango de indirección doble, por lo tanto:
1 BD + 1 BP (ID) + 1 BP (IS) = 3 Bloques

c) Leer el byte 550831707018 de un archivo:
Byte 550831707018 -> (550831707018 / 4096 = 134480397,22...) -> Bloque #134480397
Este número de bloque es incluso mayor que lo que podemos referenciar con un indirecto triple, por lo cual no es posible acceder a un bloque semejante, para ello sería necesario tener más punteros.

d) Leer el byte 8000 de un archivo, accediendo desde un SL que apunta a ese archivo:
Byte 8000 -> (8000 / 4096 = 1,95...) -> Bloque #1
Por lo cual para ese byte alcanzaría con un PD, pero, cómo accedemos desde un SL, hemos de acceder a los bloques necesarios para leer la ubicación de dicho archivo, si asumimos que la misma ocupa solamente 1 bloque (no necesariamente el bloque entero) este ademas sera el primer bloque del SL, por lo que con un PD alcanza:
1 BD (archivo original) + 1 BD (SL) = 2 BD

2) Resolucion 1) 3 PF, escritura página 1 del proceso 2

MP				P1-E1A5h (Pag 15)			P2-0301h (Pag 0)			P3-10B2h (Pag 1)			P4-5504h (Pag 5)		
Marco	**Proc**	**Pág**	**U/M**	**Proc**	**Pág**	**U/M**	**Proc**	**Pág**	**U/M**	**Proc**	**Pág**	**U/M**	**Proc**	**Pág**	**U/M**
0	P1	2	1/0	P1	2	1/0	P1	2	1/0	P1	2	0/0	P1	2	0/0
1	P2	1	0/1	P2	1	0/1	P2	1	0/1	P3	1	1/0	P3	1	1/0
2	P1	4	0/1	P1	4	0/1	P1	4	0/1	P1	4	0/1	P1	4	0/1
3	P1	B	1/1	P1	B	1/1	P1	B	1/1	P1	B	1/1	P1	B	1/1
4	P4	8	1/1	P4	8	1/1	P4	8	1/1	P4	8	1/1	P4	8	1/1
5	P4	F	0/1	P4	F	0/1	P4	F	0/1	P4	F	0/1	P4	F	0/1
6	P4	0	0/1	P4	0	0/1	P4	0	0/1	P4	0	0/1	P4	0	0/1
7	P3	6	0/1	P3	6	0/1	P3	6	0/1	P3	6	0/1	P3	6	0/1
8	P3	9	0/1à	P3	9	0/1à	P3	9	0/1	P3	9	0/1	P3	9	0/1
9	P2	D	1/1	P2	D	1/1	P2	D	1/1	P2	D	1/1	P2	D	1/1
A	P2	A	1/1	P2	A	1/1	P2	A	1/1	P2	A	1/1	P2	A	1/1
B	P3	C	1/1	P3	C	1/1	P3	C	1/1	P3	C	1/1	P3	C	1/1
C	P3	3	0/1	P3	3	0/1	P3	3	0/1	P3	3	0/1	P3	3	0/1
D	P4	5	0/0	P4	5	0/0	P2	0	1/1	P2	0	1/1	P2	0	1/1
E	P3	7	1/0	P3	7	1/0	P3	7	1/0à	P3	7	0/0	P4	5	1/1
F	P1	E	1/0	P1	E	1/0	P1	E	1/0	P1	E	0/0	P1	E	0/0à
PF				NO			PF			PF			PF		
Escritura Disco				NO			NO			ESC PAG 1 P2			NO		
Dirección Física				F1A5h			D301H			10B2H			E504H		

B) No, ya que estando el puntero en el marco 8 (página 9 del proceso 3), se reemplazaría la misma por no estar en uso pero al estar modificada se escribiría en disco para ser reemplazada.

3)

a) Tamaño máximo teórico de un archivo en FAT es equivalente al FS completo si no contamos sus estructuras administrativas.
Tam_max = 2^28 * 2^10 B = 2^38 B = **256 GIB**.
Un proceso tendrá como máximo un tamaño de 2^16 B = **64 KiB**

b) 6000/1024 = 5,9...
7000/1024 = 6,9....
Se requiere leer los bloques 5 y 6 del archivo, por lo tanto tendremos 2 acceso a bloques de disco + 6 accesos a la FAT para conocer estos bloques.

c)Habiendo 16 segmentos por proceso (4 bits de dirección) quedan 12 bits para direccionar el desplazamiento, por lo que el tamaño máximo de cada uno será de: 2^12 B = 4 KiB. Por lo tanto, no se podría cargar el archivo completo en un segmento.