Sistemas Operativos
2° Parcial 1C2019 - TM - Resolución

Aclaración: 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. La paginación jerárquica permite tablas de páginas más pequeñas, dividiendo la tabla principal en subniveles. De esta forma, no es necesario que todas las tablas estén en memoria principal, ahorrando espacio.
   Respecto a la tabla de páginas invertidas, que es única en todo el sistema y posee una entrada por cada marco, con lo cual su tamaño en memoria principal es fijo. En cambio, cada proceso tiene su propia tabla jerárquica, con lo cual su tamaño ocupado es variable, aunque pequeño. Para acceder a la tabla de páginas invertidas, a la hora de realizar una traducción, es necesario recorrerla completa. Esto puede optimizarse usando una función de hash. La tabla jerárquica simplemente se parte la dirección para acceder a los diferentes niveles.

2. La asignación contigua es rápida, pero tiene dos grandes problemas: los archivos podrían no tener lugar contiguo para crecer y las porciones de espacio que quedan libres podrían no ser suficientemente grandes para nuevos archivos, siendo esto un caso típico de fragmentación externa. La asignación indexada usa bloques del mismo tamaño y un conjunto de punteros por cada archivo, que permiten aprovechar cualquier bloque disponible, solucionando ambos problemas. Sin embargo, estos bloques podrían estar dispersos, con lo cual la estrategia de asignación contigua suele ser más rápida para el acceso secuencial.

3. El problema que se presenta es fragmentación interna dado que más de la mitad de la página no está siendo utilizada. Una opción de mejora sería utilizar páginas más pequeñas, por ejemplo de 32kb, o plantear otro tipo de esquema de memoria, como memoria contigua.

4. a) Falso. El thrashing podría ser porque un proceso en particular no tiene suficientes frames para ejecutar una instrucción, usando asignación fija.
   b) Falso. No es posible crear hardlinks a otros filesystems

5. Los algoritmos de discos buscan mejorar los tiempos entre pistas. Un ejemplo podría ser SSTF, que busca las pistas más cercanas para mejorar el tiempo de espera promedio de los pedidos. NSTEP SCAN privilegia el orden de llegada, porque arma colas de hasta N pedidos, perdiendo un poco de performance pero evitando inanición.

Práctica

1) a) Tam FS (teórico) = 2^32 * 1KB = 2^32 * 2^10 = 2^42 = 4TB
Como el disco es de 1TB, el tamaño real del FS queda en 1TB.

b) Ptrs x bloque = 1KB/4B = 2^10/2^2 = 2^8 = 256
Tam máx arch. = 256^3 * 1KB = (2^8)^3 * 2^10 = 2^34 = 16GB aproximadamente
El archivo no se puede guardar ya que con la configuración actual el inodo no puede direccionar todo el archivo.

c) Se podría incrementar el tamaño de bloques a 2KB o agregarle un puntero indirecto triple extra al inodo.

d) Siendo el archivo de 32GB y cada bloque de 1KB. El archivo ocuparía 32M bloques por lo que se necesitarían modificar 2^25 entradas de la tabla FAT (1 x bloque).

2) N - STEP - SCAN (N = 3)

Calculamos primero la cantidad de cilindros =>
Capacidad disco = Cilindros * cabezales * sectores x pista * tamaño Sector
128MiB = C * 8 * 64 * 2KiB
2^27 = C * 2^3 * 2^6 * 2^11
2^27 = C * 2^20
C = 2^7 = 128 cilindros

T = 0 pedidos: 100 - 60 -> se comienza a atender
50 (subiendo) 60 - 100 -> se recorrieron 50 cilindros = 100 ms

T = 100 ms
Pedidos (80 - 102 - 100) - (100 - 10
Comienzo a atender la primera cola
100 (subiendo) - 100 - 102 - 127(TOPE) - 80 -> se recorrieron (127 - 100 + 127 -80) 74 cilindros = 148 ms

T = 248 ms
Pedidos (100 - 10 - 10)
80 (bajando) 10 - 10 - 0 - 100 -> se recorrieron 180 cilindros = 360 ms

Orden total pedidos = 60 - 100 - 100 - 102 - 80 - 10 - 10- 100
T fin atención = 608ms

b) C-LOOK (para este no importa saber la cant de cilindros
T = 0
Pedidos 100 - 60
50| 60 -> se recorrieron 10 cilindros = 20 ms

T = 20 ms
Pedidos: 100 - 80 - 102
60 | 80 -> se recorrieron 20 cilindros = 40 ms

T = 60 ms
Pedidos: 100 - 102 - 100 -100
80 | 100 -100 -100 -> se recorrieron 20 cilindros = 40 ms

T = 100 ms
Pedidos: 102 - 10
100 | 102 -> se recorrieron 2 cilindros -> 4 ms

T = 104ms
Pedidos: 10 - 10
102 | (caemos a 10)

Orden total atención : 60 - 80 - 100 - 100 -100 - 102 - 10 -10
T fin atención: 104 ms

3) **A -**

i) DL = NRO PAG | OFFSET -> x BITS nro pag , y BITS offset , x + y = 32
Cant págs que podría tener un proceso = 2 ^x

RAM = cant frames * tamaño frames (1024 por dato)
SIn memoria virtual cant máx págs por proceso = cant frames =>

1GiB = 2^30 = 2^10 ^Tam frame
Tam frame = Tam pág = 2^20 => cose utilizan 12 bits para el nro de pág
⇒ Cán máx de págs = 2*12

ii) Fragmentación interna promedio por proceso = media pág = 2^20 /2 = 2^19
La cantidad de procesos máximos que se podría tener en memoria sería uno por frame = 1024
=> fragment int promedio máx = 2^19 * 2^10 = 2^29 = **512 MiB**

Por otro lado, si consideramos la aclaración de que se le dan 4 frames fijos a cada proceso, en ese caso podríamos tener 1024/4 = 256 procesos en memoria => 2^19 * 2^8 = 2*27 = **128 MiB**

B -

i)
DL 00B 12B10h -> pág 11 -> en TP P = 1 -> sólo se activa bit de uso y se actualiza el inst de últ ref
DL 002 22ABCh -> pág 2 -> en TP P = 0 -> PF -> según el dato, la víctima termina siendo la pág 1 que estaba en el frame 7

Vemos cuáles eran las opciones -> las 4 págs con P = 1

Pág	U	M	Últ ref
1	**1**	**1**	**100**
11	1	0	Mayor valor
33	1	0	101
34	1	1	200

Podrían ser:
- FIFO, no está el dato de instante de carga, pero podría ser válido
- Clock, todos tenían como bit de uso en 1 , depende del instante de carga
- LRU, se elige el que tiene el menor valor

No podría ser
- Clock modificado, la pág 1 tenía U y M en 1, mientras que 11 y 33 tenían el M en 0 y hubiesen sido mejores opciones para este algoritmo

ii)
Como la TLB no tiene columna de proceso, al cambiar de proceso hay que limpiarla, sólo tendía las entradas de estas dos últimas referencias

PÁG	FRAME
11	20
2	7