Sistemas Operativos
2° Parcial 2C2023 - TT - 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. Para mover un archivo de ext2 primero he de leerlo, por lo cual en el directorio que lo "contiene", localizo en sus entradas el número inodo correspondiente al archivo, dado que lo moveremos a un fs tipo fat32, todos aquellos atributos del inodo que representen permisos (de usuarios, grupos, etc..) se perderán, dado que en este otro fs no existen dichos atributos, se deberá entonces leer los punteros (y bloques de punteros) del inodo que referencian sus bloques de datos que luego se escribirán en fat32. Para poder escribir dichos bloques de datos, es necesario primero generar una entrada en el directorio destino para el archivo nuevo y localizar un primer cluster libre recorriendo la tabla FAT, se seguirá recorriendo la tabla buscando los siguientes clusters libres para escribir los datos que requiera el archivo.
   Finalmente, se debe de "eliminar" el archivo original del fs ex2, para ello se marca el inodo en el bitmap de inodos como libre y se ha de actualizar el bitmap de bloques libres del grupo al que pertenece ese inodo también como libres (probablemente también se deba actualizar algún valor en el Super bloque y en el Descriptor de bloques dado que se modificó un grupo).

2. Sí, podría. Sería un esquema similar al de segmentación, pero recordando que en este caso el proceso estaría ubicado en su totalidad en una única partición de un tamaño predefinido anteriormente en el sistema. Los procesos podrían manejar direcciones lógicas de forma relativa al comienzo del programa, y las mismas se traducirían a direcciones físicas usando la tabla de particiones de memoria del sistema.

3.

	Secuencial/Contiguo	Enlazado/Encadenado	Indexado
Fragmentación	Fragmentación Interna en el último bloque del archivo Fragmentación externa debido a que los bloques deben estar contiguos.	Fragmentación Interna en el último bloque del archivo.	Fragmentación Interna en el último bloque del archivo.

Crecimiento	Es posible sólo si hay bloques libres después del último bloque.	Es posible siempre que haya bloques libres en el filesystem	Es posible siempre que haya bloques libres en el filesystem
#accesos a bloque	Acceso directo. Se puede calcular por que los bloques están contiguos	Acceso Secuencial. Cada bloque tiene un puntero al siguiente y hay que recorrer los bloques anteriores para llegar a uno en particular.	Acceso directo. Se accede a través de una tabla por archivo indexada.

4.
    a. Verdadero: Un caso podría ser que un proceso necesite cierta cantidad de páginas en memoria para poder ejecutar, pero su conjunto residente no es lo suficientemente grande y por lo tanto, generaría thrashing. Usando asignación variable podría aumentar el tamaño del conjunto residente para reducir la cantidad de page faults.
    b. Falso: Además de las páginas del proceso, porciones de la tabla de páginas del proceso podrían estar en el espacio de intercambio, las cuales al ser necesitadas generan page faults si no se encuentran en memoria principal.

.

5. Al crear un softlink sobre un archivo regular es necesario agregar una nueva entrada de directorio. Considerando que el mismo necesitará un inodo, se asignará uno libre (de la tabla de inodos) marcando el mismo como ocupado en el bitmap de inodos. Además, se necesitará un bloque de datos para alojar el path del archivo apuntando con lo cuál se asignará uno libre y se modificará el bitmap de bloques libres.
   Dichos cambios además impactaran el superbloque y descriptor del grupo correspondiente, ya que la cantidad de bloques e inodos libres totales y del grupo cambiaron.
   El principal beneficio de un softlink por sobre un hardlink es que el mismo puede ser realizado entre diferentes particiones.

Práctica

1. El tamaño de la tabla FAT es de 16384 bloques de 4KiB o sea, 16384 * 4KiB = 64 MiB (2^26 bytes)
   Los punteros de FAT32 son de (32 bits) 4 bytes. Por lo tanto contiene 2^26 bytes / 2^ 2 bytes de punteros. Por lo tanto, contiene 2^ 24 entradas. En total 16.777.216 punteros que apuntan a un bloque de 4 KB.
    a. El tamaño máximo real del File System es cantidad de punteros por tamaño del bloque (16.777.216 * 4.096) = 68.719.476.736 = 64 GiB. Como el tamaño del disco es mayor, puede alojar dicho filesystem en una partición de 64 GB.

El tamaño máximo teórico del File System no está limitado ni por el disco ni por particiones. Es la cantidad máxima de punteros que puede contener la FAT por el tamaño del bloque. En FAT los punteros son de 32 bits pero sólo se usan 28. 2^28 * 2^12 = 2^40 = 1 TiB.

b. EXT tiene punteros de 4 bytes y bloques de 4 KiB. Los bloques de punteros contienen (4096/4) 1024 punteros.
El archivo ocupa 4.326.000 / 4096 bloques de DATOS. Son 1035 bloques.

El inodo está compuesto por 10 punteros directos, 1 puntero indirecto simple y 1 puntero indirecto doble.
10 PD cubre 40 KB (10 bloques)
1 P IS cubre 4096 KB (1024 bloques).
Hasta acá se cubren los primeros 1034 bloques de datos del archivo. Falta un bloque que será el primer bloque de datos que se accede a través de la indirección doble.
Por lo tanto el archivo requiere:
1 bloque de punteros de la indirección simple (**1**).
1 bloque de punteros de la indirección doble y de esta indirección, el primer bloque de punteros (**1 + 1**).
El archivo requiere 1035 bloques de datos + 3 bloques de PUNTEROS. Total 1038 Bloques.

2.

a. Sí, sería beneficioso para dicho sistema contar con direcciones más grandes. Aún cuando el tamaño de la dirección actual le permite direccionar toda la memoria física (con 32 bits es posible direccionar 2^32 = 4GiB), una de las ventajas de utilizar memoria virtual es que los procesos podrían referenciar direcciones más grandes.
En dicho sistema, si se continúa utilizando direcciones de 32 bits, ese aspecto de la memoria virtual no podría ser aprovechado.
b. De los 32 bits totales de la dirección, 12 bits son utilizados para el offset (2^12 = 4KiB), por lo cual los 20 bits restantes son utilizados para el número de página. Esto quiere decir que las tablas de página tendrán 2^20 entradas. Teniendo en cuenta que cada entrada pesa algunos bytes, las tablas de página de cada proceso estarían en el orden de los MiB. Sabiendo que varios procesos coexisten en el sistema en el mismo tiempo, esto representa una cantidad significativa de memoria siendo utilizada solamente para alojar tablas de páginas.
La solución a dicho problema es utilizar paginación jerárquica, donde solo la tabla de primer nivel debe ser mantenida en memoria principal, y las tablas de 2do nivel pueden ser movidas entre memoria y swap y viceversa, a demanda.
c. El bitmap de marcos libres tendrá tanta cantidad de entradas como marcos tenga la memoria. Dado que la memoria principal es de 4GiB y los frames son de 4KiB, el bitmap tendrá 2^20 entradas (2^32/2^12). Sabiendo que con 1 byte

puedo alojar 8 entradas (son bits), el bitmap pesará 2^17 bytes (2^20 / 2^3) = 128KiB.

3. Las páginas tienen como máxima fragmentación interna 2047 bytes, por lo tanto las páginas son de 2 KiB, y al tener el mismo tamaño que los bloques, estos también son de 2 KiB.
   Punteros de 4 bytes => Punteros x Bloque = 2 KiB / 4 bytes = 512 punteros x bloque
   Con la configuración del inodo puedo direccionar los bloques:
   Punteros Directos: #0 - #11
   Ind. Simples (1): #12 - #523 Ind. Simple (2): #524 - #1035
   Ind. Doble: #1036 - #263179
   Ind. Triple: #263180 - #134480907

   Cálculo el bloque de datos a calcular es Byte #12300 / 2048 bytes = 6.00... => BD #6
   Sabemos que la página 0 posee el bloque 0 del archivo, y la página 1 posee el bloque 1 del archivo, y los bloques a ser accedidos son:

   BD #6 - BD #850 - BD #1035 - BD #6 - BD #1036
   Como las páginas se cargan de manera secuencial, cada bloque distinto irá a una página distinta:

# Bloq. Datos	6	850	1035	6	1036
# pagina	2	3	4	2	5

#Frame	#Página	2E	3L	4L	2E	5E
54	0	0	3	3	3	5
55	1	1	1	4	4	4
56	-	2	2	2	2	2
		PF	PF	PF		PF

   Accesos a bloques:
   -BD #6: 1 Bloque de datos

   -BD #850: 1 Bloque de Punteros (IS) + 1 Bloque de datos, no hace falta bajar al archivo el bloque 0 reemplazado dado que había sido de lectura por enunciado.

   -BD #1035: 1 Bloque de Punteros (IS) + 1 Bloque de datos, no hace falta bajar al archivo el bloque 1 reemplazado dado que había sido de lectura por enunciado.

   -BD #6: 0 accesos dado que ya lo tenemos en memoria

   -BD #1036: 1 Bloque de Punteros (ID) + 1 Bloque de Punteros (IS) + 1 Bloque de Datos, no hace falta bajar al archivo el bloque #850 reemplazado dado que había sido de lectura.