Actividades-SO-Temas-3-y-4.pdf

Anónimo

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Actividades SO examen final Temas 3 y 4

2. Si se pierde el primer puntero de la lista de espacio libre, ¿podría el Sistema Operativo reconstruirla? ¿Cómo?
Si se podría recuperar y dependiendo de la implementación del sistema tendrá una forma u otra: Con una tabla FAT bastaría con recorrer todas sus entradas y comprobar que esté vacía, esto aunque no es muy eficiente es mejor que recorrer la memoria entera en busca de bloques libres. Si en cambio tiene un mapa de bits y ese está en memoria principal es muy eficiente ya que bastaría con recorrer este mapa y ver los bloques que tienen el bit a 0.

3. El espacio libre en disco puede ser implementado usando una lista encadenada con agrupación o un mapa de bits. La dirección en disco requiere de bits. Es un disco con B bloques, en que F están libres. ¿En qué condición la lista usa menos espacio que el mapa de bits?
B son los bits que necesita el mapa: un bit por bloque
F*D son los bits que necesita la tabla: DE bits para cada puntero que hay q referenciar, si hay F bloques libres harían falta F*D bits para guardar los punteros.

4. Entre los posibles atributos de un archivo, existe un bit que marca un archivo como temporal y por lo tanto está sujeto a destrucción automática cuando el proceso acaba ¿Cuál es la razón de esto? Después de todo un proceso siempre puede destruir sus archivos, si así lo decide.
Podría ser para facilitar la vida al programador, ya que por un descuido podrían no borrarme esos datos temporales y quedar en memoria lo que podría derivar en espacio inutil en el disco y así se asegura de todo lo que haya en el disco sea información útil

5. Algunos SO proporcionan una llamada al sistema (RENAME) para dar un nombre nuevo a un archivo existente ¿Existe alguna diferencia entre utilizar esta llamada para renombrar un archivo y copiar el archivo a uno nuevo, con el nuevo nombre y destruyendo el antiguo?

A nivel de utilidad de usuario no, ya que se obtiene el mismo archivo con otro nombre, pero nivel del SO requiere crear un nuevo espacio en la tabla del directorio copiar todos los bloques en bloques distintos de memoria y si usa tabla FAT cambiarla con las entradas correspondientes a cada bloque de memoria que se ha tenido que comprobar si estaba vacio, ademas tendria un modo distinto aunq características parecidas como el tamaño. Luego se debería de borrar la entrada del directorio correspondiente actualizando a libres los bloques que antes ocupaba el archivo. Además en caso de que existieran enlaces duros a este archivo se perderán.

6. Un in-nodo de sistema de archivos s5fs de UNIX tiene 10 direcciones de disco para los diez primeros bloques de datos, y tres direcciones más para realizar una indexación a uno, dos y tres niveles. Si cada bloque índice tiene 256 direcciones de bloques de disco ¿cuál es el tamaño del mayor archivo que puede ser manejado, suponiendo que 1 bloque de disco es de 1 Byte?

Lo desglosamos:
-Las 10 primeras entradas que contienen direcciones equivaldrá a 10 KB de acceso directo, esto quiere decir que los archivos de 10KB o menos tienen un acceso directo a memoria y son casi instantáneos.
La entrada 11 es una dirección a un bloque de primer nivel de 256 direcciones lo que quiere decir que en esta entrada caben 256KB.
-La entrada 12 tiene una dirección a un bloque de índices lo que quiere decir que cada entrada hace referencia a 256 direcciones más (lo que soporta un bloque) -> 256(entradas)*256(direcciones de cada bloque de cada entrada) = 65536.
-En la última entrada está la dirección a una tabla de 3 nivel: igual q antes pero con otro nivel mas, lo que equivale a: 256*256*256= 16777216 KB
*Esto da como resultado que un archivo en esta memoria puede tener como máximo 16843018 KB de espacio.

7. Sobre la conversión de direcciones lógicas dentro de un archivo a direcciones físicas de disco. Estamos utilizando la estrategia de indexación a tres niveles para asignar espacio en disco. Tenemos que el tamaño de bloque es igual a 512 bytes, y el tamaño de puntero es de 4 bytes. Se recibe la solicitud por parte de un proceso de usuario de leer el carácter número N de determinado archivo. Suponemos que ya hemos leído la entrada del directorio asociada a este archivo, es decir, tenemos en memoria los datos PRIMER BLOQUE y TAMAÑO. Calcule la sucesión de direcciones de bloque que se leen hasta llegar al bloque de datos que posee el citado carácter.

Teniendo en cuenta que tenemos la dirección del primer bloque de índices tenemos que recurrir a los n bit correspondientes a la dirección dentro del bloque de 1º nivel que obtenemos dentro de los bits de la dirección del dato solicitado y nos desplazamos 2^n bits según nos indique este número, aquí estaría la dirección donde se encuentra el bloque de segundo nivel correspondiente, una vez allí con los siguiente m bits buscamos la dirección del último bloque de índices y con los siguientes p bits obtenemos el desplazamiento dentro de ese bloque para encontrar la dirección en memoria donde se encuentra el dato buscado.

8. ¿Qué organización de archivos elegiría para maximizar la eficiencia en términos de velocidad de acceso, uso del espacio de almacenamiento y facilidad de modificación (añadir/borrar /modificar), cuando los datos son:
a) modificados infrecuentemente, y accedidos frecuentemente de forma aleatoria
La gestión contigua aqui seria la mejor ya que al no ser modificado con frecuencia el problema de que hay que comprimirlo o cambiarlo de sitio por falta de espacio apenas se daría (que es el principal inconveniente de esta forma de organización) y al accederse frecuentemente el método más rápido es que estuviera todo junto, ya que al estar todo junto se evita tiempo de acceso por saltos en memoria.

b) modificados con frecuencia, y accedidos en su totalidad con cierta frecuencia
Aquí el enlazado y el indexado son una buena opción. El enlazado mejor ya que si se accede al archivo en su totalidad basta con acceder al principio del archivo e ir saltando según los punteros. Es mejor que el indexado ya que al aumentar el tamaño del archivo bastaría con buscar un bloque libre rellenarlo y apuntar a ese bloque, en cambio para el indexado es tb fácil a no ser que la tabla se exceda lo que podría dar un problema.

c) modificados frecuentemente y accedidos aleatoriamente y frecuentemente.
Aquí la única buena opción sería el indexado ya que puede acceder a cualquier parte del archivo a través de su tabla de índices a la parte buscada, con el enlazado deberías recorrer las listas hasta encontrar el bloque buscado. No puede ser contiguo (que sería una buena opción) pq al modificarse frecuentemente habría muchos problemas con la adaptación del espacio.

10. ¿Tendría sentido combinar en un mismo sistema de archivos dos métodos de asignación de espacio en disco diferentes? Por ejemplo, el método continuo combinado con el enlazado. Ponga un ejemplo y razonalo.

Si, podría tener sentido, como por ejemplo un mix de contiguo para los archivos más estáticos que normalmente no son modificados y son muy grandes (así evitamos tener que guardar una tabla con punteros a los distintos bloques como bases de datos, además podremos acceder fácilmente a ellos y de forma rápida ya q es solo encontrar el bloque y el desplazamiento dentro de ese bloque.

En cambio para archivos más pequeños o con más modificaciones podemos usar el enlazado o indexado que aunque su acceso es más lento, si son bloques más pequeños no debería de tardar demasiado y en caso de modificaciones basta con buscar un hueco en memoria y con el puntero apuntar a ese bloque o en caso de indexado añadir una entrada mas que referencie a ese bloque.

11. ¿Por qué la FAT en MS-DOS y Windows tiene un tamaño fijo en disco para cada sistema de archivos?

Ya que si fuera dinámico podría consumir mucho tiempo, merece la pena ocupar un poco más de espacio (tampoco es mucho 4 bytes por puntero generalmente) para registrar todos los bloques en los que se divida la memoria (memoria con 100 bloques la FAT tendría 100 entradas, una para cada bloque), además podría fragmentarse.

12. Para comprobar la consistencia de un sistema de archivos de Unix, el comprobador de consistencia (fsck) construye dos listas de contadores (cada contador mantiene información de un bloque de disco), la primera lista registra si el bloque está asignado a algún archivo y la segunda, si está libre. Según se muestra en la siguiente figura:
En uso: 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0
libres: 0 0 0 1 1 1 0 0 0 1 0 1 1 0 1
¿Existen errores? ¿Son serios estos errores? ¿por qué? ¿Qué acciones correctivas sería necesario realizar sobre la información del sistema de archivos?

Se ve claramente una inconsistencia ya que algunos bloques que supuestamente están en uso están marcados también como libres, lo que provocaría un gran problema ya que si se va a asignar un espacio a un archivo que está marcado como “Libre” y en realidad está ocupado lo que se escriba ahi “aplastaria” los datos de otro archivo q en realidad sí está usando. También hay otros bloques que no están en uso pero tampoco marcados como libres por lo que ese espacio estaría perdido, ya que ninguno podría acceder a esos datos.

13. Cuando se abre el archivo /usr/ast/work/f, se necesitan varios accesos a disco. Calcule el número de accesos a disco requeridos (como máximo) bajo la suposición de que el in-nodo raíz ya se encuentra en memoria y que todos los directorios necesitan como máximo 1 bloque para almacenar los datos de sus archivos.

-Habría que acceder al bloque del directorio raíz que contiene la entrada a /usr
-Otro acceso para obtener el inodoro de usr.
-Acceder al bloque del directorio con la entrada a /ast
-Otro para obtener el inodo de ast
-Otro para leer el bloque que contiene la entrada a /work
-Obtener el inodo del archivo work
-Extraer la entrada correspondiente al archivo f
-Conseguir el inodo del archivo f
En total habría que hacer 8 accesos a memoria.

14. Supongamos que un proceso, P1, abre el archivo "datos" en modo lectura/escritura y otro proceso, P2, abre el mismo archivo y con el mismo modo, ya continuación crea un proceso hijo que abre el archivo "/usr/pepe/doc" en modo lectura/escritura. Represente toda la información relevante sobre el estado de las tablas de descriptores de archivos, tabla de archivos y tabla de inodos después de dichas operaciones.

En la tabla de descriptores de archivo cada entrada va a apuntar a una entrada de la tabla de archivos en orden de llegada según haya libres. Y en cada entrada de la tabla de archivos se hace referencia al nodo al que se está apuntando y sus modos de acceso. Y en la tabla de inodos solo cambia el número de referencias a este inodo (cuantos procesos están apuntando a ese archivo)

Cuando p1 y p2 abren el fichero “datos” se crean dos entradas en la tabla de archivos con el mismo inodoro en ambos.
Tras esto el estado de la tabla de archivos de cada hijo es parecida. En p1 se guarda la entrada de la tabla de archivos 1, en el proceso 2 se apuntaría a la entrada 2, y en el hijo se apuntaría a la 3 entrada de la tabla de archivos.

En cada entrada de la tabla de archivos se guarda el inodo correspondiente al archivos p1 y p2 (I1) y p3->hijo de p2(I2), apunta a otro inodo pq abre otro archivo.
Ya en la tabla de inodos, lo único que cambia es el número de referencias que hay a ese archivo

15. Suponiendo una ejecución correcta de las siguientes órdenes en el sistema operativo Unix: /home/jgarcia/prog > ls -i (* lista los archivos y sus números de inodos del directorio prog*) 18020 fich1.c 18071 fich2.c 18001 pract1.c /home/jgarcia/prog > cd ../tmp /home/jgarcia/tmp > ln -s ../prog/pract1.c p1.c (* crea un enlace simbólico *) /home/jgarcia/tmp > ln ../prog/pract1.c p2.c (* crea un enlace absoluto o duro*Represente gráficamente cómo y dónde quedaría reflejada y almacenada toda la información referente a la creación anterior de un enlace simbólico y absoluto ("hard") a un mismo archivo, pract1.c.

Al crear un enlace absoluto ese archivo creado apuntará al mismo inodoro por lo que será literalmente el mismo archivo referenciado desde distintas partes. En la tabla de inodos en el inodoro de ese archivo se incrementa el contador de referencias indicando que se está apuntado desde un enlace duro. En cambio el enlace simbólico sólo contiene la dirección del archivo apuntado.

16. En Unix, ¿qué espacio total (en bytes) se requiere para almacenar la información sobre la localización física de un archivo que ocupa 3 Bytes? Suponga que el tamaño de un bloque lógico es de 1 Kbytes y se utilizan direcciones de 4 bytes. Justifique la solución detalladamente.

17. ¿Por qué en el sistema de archivos FAT está limitado el número de archivos y/o directorios que descienden directamente del directorio raíz y en Unix no lo está?

En otros SO que utilizan la FAT como forma de organización de archivos como Windows la tabla tiene un tamaño predefinido al directorio por lo que el espacio está supeditado a ese tamaño de tabla, entonces cabrían tantos archivos como entradas tenga la tabla fat. En cambio para Unix al utilizar un esquema de inodos permite que crezca dinámicamente.

18. Tenemos un archivo de 4 MBytes, calcule el espacio en disco necesario para almacenar el archivo (incluidos sus metadatos) en un Sistema de Archivos FAT32 y en otros 5fs. Suponga que el tamaño del cluster en FAT32 y el de bloque lógico en s5fs es de 4 KBytes.

Harían falta 4x2**20/(4x1024) bloques → se necesitan 1024 bloques para guardar el archivo por lo que en total (y suponiendo que cada puntero son 4 bytes al ser FAT32) 4+1024 bloques que apuntar = 4096 bytes, 4KB. Esto sería en ambos sistemas lo que se pierde de espacio para guardar las direcciones de los bloques de los archivos, a eso le sumamos el espacio del archivo: En total 4,004MB.