Sistemas Operativos

Práctica 6: Sistemas de archivos

Notas preliminares

- Los ejercicios marcados con el símbolo ★ constituyen un subconjunto mínimo de ejercitación. Sin embargo, aconsejamos fuertemente hacer todos los ejercicios.

Ejercicio 1

Suponer una computadora cuyo disco se accede sin memoria *cache* y un sistema de archivos `FAT`. Además, en este sistema, la `FAT` no queda almacenada en la memoria (recordar que lo normal es que la `FAT` se cargue en memoria). ¿Cuántos accesos al disco son necesarios para llegar hasta el último bloque de un archivo de N bloques?

Ejercicio 2 ★

Se tiene un disco con capacidad de 128 GB, y bloques de 8 KB. Supongamos un sistema de archivos similar a `FAT`, donde la tabla se ubica desde la posición 0, y cada entrada de la tabla es de 24 B.

a) ¿Qué tamaño ocupa la tabla?
b) ¿Cuántos archivos de 10 MB es posible almacenar?
c) Se sabe que un archivo comienza en el bloque 20. Dada la siguiente `FAT`, indicar el tamaño de dicho archivo.

Bloque	0	1	2	3	4	5	6	...	20	21	22	...
Siguiente	EOF	2	23	4	5	0	7	...	21	22	3	...

Ejercicio 3 ★

Un sistema de archivos de UNIX, similar a `ext2`, tiene bloques de tamaño 4 KB y el direccionamiento a bloques de disco (LBA) es de 8 B. A su vez, cada `inodo` cuenta con 5 entradas directas, dos indirectas y una doblemente indirecta.

a) ¿Cuál es el tamaño máximo de archivo que soporta?
b) Si el 50 % del espacio en disco está ocupado por archivos de 2 KB, el 25 % por archivos de 4 KB y el 25 % restante por archivos de 8 KB, ¿qué porcentaje del espacio en disco está siendo desperdiciado? (Considerar sólo el espacio utilizado en los bloques de datos).
c) ¿Cuántos bloques es necesario acceder para procesar completo un archivo de 5 MB?

Ejercicio 4 ★

Suponer que se cuenta con un sistema de archivos basado en inodos como `ext2` y bloques de 4 KB.

a) Si se tiene un archivo de 40 KB, ¿cuántos bloques es necesario leer para procesarlo completamente?
b) ¿Cuántos bloques es necesario leer si el archivo tiene 80 KB?

Ejercicio 5

Considerando un sistema `ext2` (12 entradas directas, 3 indirectas, 1 doble indirecta y 1 triple indirecta) y un sistema basado en `FAT`:

¿Cuántos bloques de disco se deben acceder para leer los siguientes bloques de un archivo?

i) 1, 2, 3, 5, 7, 11, 13, 17, 23.

ii) 1, 2, 3, 4, 5, 6, 10001.

iii) 13, 10000, 1000000.

iv) 14, 15...50.

Asumir que si un bloque se lee dos veces, se va a buscar una sola vez al disco, que tanto la `FAT` como el inodo del archivo correspondiente ya están cargados en memoria, y que se conoce el número de bloque inicial (en `FAT`) y el número de inodo (en `ext2`).

Ejercicio 6

Considerar la siguiente ruta a un archivo: `/home/aprobar.txt`.

a) En un sistema basado en `FAT`, ¿cuántos bloques de disco se deben acceder para llegar a los bloques de datos de este archivo?
b) En un sistema basado en `ext2`, se quiere leer el archivo `/pepe.txt`, que es un enlace simbólico al archivo mencionado arriba. El enlace no está cargado en memoria. ¿Cuántos bloques de disco se deben acceder para llegar a los bloques de datos del archivo referenciado?

Ejercicio 7 ★

Una compañía que fabrica discos rígidos decide, como parte de cierta estrategia comercial, emprender la creación de un nuevo *filesystem*. Durante la fase de diseño preliminar el equipo de ingeniería a cargo del proyecto discute acaloradamente la conveniencia de adoptar un enfoque inspirado en `FAT` o la de uno basado en inodos. Indicar cuál de las dos opciones recomendaría, y por qué, para cada uno de los siguientes requerimientos:

1. Es importante que puedan crearse enlaces simbólicos.
2. Es importante que la cantidad de sectores utilizados para guardar estructuras auxiliares sea acotada, independientemente del tamaño del disco.
3. Es importante que el tamaño máximo de archivo sólo esté limitado por el tamaño del disco.
4. Es importante que la cantidad de memoria principal ocupada por estructuras del *filesystem* en un instante dado sea (a lo sumo) lineal en la cantidad de archivos abiertos en ese momento.

Ejercicio 8

Se tiene un disco rígido de 16 GB de espacio con sectores de 1 KB. Se desea dar formato al disco usando un sistema de archivos de uso específico llamado *HashFS*, basado en `FAT`. La idea es que no existen directorios ni archivos. Dado un path, se calcula el *hash* del nombre y éste indica cuál es el archivo buscado.

En resumen, este sistema de archivo cuentas con dos tablas:

- Una única `FAT` que guarda las entradas correspondientes al próximo bloque, indicando el final de un archivo cuando estos valores coinciden.

- Una única tabla de *hash* que contiene, para cada *hash* posible, el identificador del bloque inicial y el tamaño en bytes del archivo correspondiente a dicho *hash*.

La novedad es que este sistema de archivos permite configurar los siguientes elementos:

- Tamaño del bloque: 2, 4 u 8 sectores.
- Tamaño de identificadores de bloque: 8, 16, 24 o 32 *bits*.
- Tamaño del *hash*: 8, 16, 24 o 32 *bits*.

a) Suponiendo que se configura con 2 sectores por bloque, identificadores de bloque de 24 *bits*, y *hash* de 16 *bits*. ¿Cuál es el tamaño que ocupa la `FAT`? ¿Cuál es el tamaño de la tabla de archivos? ¿Cuál es el espacio que queda en disco para archivos?
b) Sabiendo que se desea maximizar la cantidad de archivos que el sistema soporta y que, además, en promedio los archivos tendrán un tamaño de 1 KB, ¿cuál sería la configuración óptima del sistema de archivos? Justificar.
c) ¿Cómo lo configuraría si el promedio de tamaño de archivos es de 16 MB? Justificar.

Ejercicio 9

Linux permite pasar un descriptor de archivo de un proceso a otro (es decir, el valor de retorno de la syscall `open()`). Suponga que un proceso abre el directorio `/home` que está sobre una partición de `ext2` y le envía el descriptor del archivo `home` a otro proceso con los permisos necesarios para poder leerlo.

Suponiendo que tiene la función auxiliar `Ext2FSInode load_inode(int inode_number)` que dado un número de inodo devuelve la estructura del inodo, escriba el pseudocódigo que le permita obtener el nombre del directorio (`home`) a partir del descriptor de archivo recibido.

Ejercicio 10 ★

Se tiene un disco formateado con `FAT` para el cual se quiere poder devolver el contenido de un archivo a partir de su ruta absoluta. Para ello se debe implementar la función:

```
datos = cargar_archivo(directorios[])
```

donde `directorios` es la lista de los nombres de los directorios de la ruta (ordenados, incluyendo el nombre del archivo, y sin incluir al directorio raíz). Es decir, si el archivo a abrir es `\Documentos\Users\foto.png` entonces `directorios = ['Documentos', 'Users', 'foto.png']`.

Para ello se cuenta con las siguientes funciones auxiliares:

- `FAT_entry(block_address)`: devuelve la entrada de la tabla `FAT` de la posición `block_address`.
- `raw_data = read_blocks(block_address1,block_address2,...)`: lee del disco todos los bloques indicados por parámetro, en orden.
- `parse_directory_entries(raw_data)`: devuelve una lista de `struct_entrada_directorio`, donde cada elemento representa los subdirectorios del directorio pasado en `raw_data`.
- `raw_data = root_table()`: devuelve los datos de la tabla de directorios de `root`.

Se pide:

a) Enumerar tres campos que debe tener la estructura `struct_entrada_directorio` según este tipo de *filesystem*.
b) Escribir el pseudo-código de la función `cargar_archivo`.

Ejercicio 11 ★

Considere un sistema de archivos `ext2`. Se pide:

a) Implementar una función `find_file_less_size(char * dir, int min_bytes, char * arch_nombre)`, que tome el `path` de un directorio, una cantidad de bytes, y un nombre, y devuelva una lista con todos los archivos que tengan un tamaño menor a `min_bytes`, cuyo nombre sea `arch_nombre`, a partir de `dir` (inclusive). La búsqueda deberá también considerar los subdirectorios del directorio dado (excepto '.' y '..').

b) Diseñar la lista que retornará la función para que contenga: la información de cada tipo de archivo encontrado (regular, binario, bloques, directorio, etc), su última fecha de modificación, su tamaño, y su propietario.

Considere que cuenta con el tamaño del bloque en la variable `BLOCK_SIZE`.

Se cuenta con las siguientes estructuras para representar inodos y entradas de directorio:

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
struct Ext2FSInode {
    unsigned short mode; // info sobre el tipo de archivo y los permisos
    unsigned short uid; // id de usuario
    unsigned int size; //  en bytes
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime; // fecha ultima modificacion
    unsigned int dtime;
    unsigned short gid; // id de grupo
    unsigned short links_count; // cantidad de enlaces al archivo
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
};
```

Se cuenta también con las siguientes funciones:

```
char * nombre_propietario(unsigned short id_usuario)
// dado el id de un usuario, devuelve el nombre del usuario.
struct Ext2FSInode * Ext2FS::inode_for_path(const char * path)
// dado un path, devuelve su inodo
void Ext2FS::read_block(unsigned int block_address, unsigned char * buffer)
// dada una direccion de bloque y un buffer, carga el bloque indicado en el // buffer
unsigned int Ext2FS::get_block_address(struct Ext2FSInode * inode, unsigned int block_number)
// dados un inodo y un numero de bloque, recorre el inodo buscando la
// direccion del bloque de datos indicado
struct Ext2FSInode * Ext2FS::load_inode(unsigned int inode_number)
// dado un numero de inodo, busca el inodo en el grupo y lo devuelve
```

c) ¿Qué problemas podrían ocurrir al hacer la búsqueda del item anterior si consideramos también los enlaces simbólicos?