Departamento de Arquitectura

Sistemas Operativos
Práctico 9

Curso 2024

Objetivos

- Familiarizarse con las funciones de manejo de sistemas de archivo de un sistema operativo.

Ejercicio 1 (basico)

(a) Considere un archivo, actualmente conteniendo 100 bloques. Indicar cuántas operaciones de entrada/salida son necesarias cuando estamos trabajando con las siguientes estrategias de ubicación: contigua, enlazada e indexada, si un bloque:

1. Es agregado al comienzo.
2. Es agregado en el medio.
3. Es agregado al final.
4. Es borrado del comienzo.
5. Es borrado del medio.
6. Es borrado del final.

(b) ¿Cuales estrategias de asignación presentadas en la parte anterior presentan fragmentación de disco interna y cuáles externa?

Ejercicio 2 (medio) Un sistema operativo administra sus archivos en disco utilizando el método de asignación indexada. Para esto, se dispone de las siguientes estructuras:

```
type block = array [0..511] of byte; // 512 bytes

type dir_entry = Record
        name : array [1..12] of char;  // 12 * 8 bits
        type : (file,dir);             // 1 bit
        used : boolean;                // 1 bit
        inode_num : int;           // 16 bits
        perms : array [1..14] of bit;  // 14 bits
    End;

type inode = Record
        inode_num : int;                // 16 bits
        used      : boolean;                // 1 bit
        data      : array [1..5] of int;// 5 * 16 bits
        tope      : 0..5;                   // 3 bits
        type      : (file, dir);            // 1 bit
        size      : int;                // 16 bits
        reserved  : array[1..11] of bit;    // 11 bits
    End;

type inode_table = array [0..max_inode_on_disk] of inode;
type disk = array [0..max_blocks_on_disk] of block;
Var
  TI   : inode_table;
  D    : disk;
```

A su vez, se sabe que el directorio raíz es el inodo número 0, que la tabla de inodos y el disco son globales, y que cada bloque de datos de los directorios tiene 32 entradas de tipo dir_entry.

1. Implementar una función que retorne la cantidad de bytes utilizados por los archivos (type == file) del sistema de archivos.
2. Implementar una función que busque un archivo (file o dir) dentro de un directorio (no búsqueda recursiva).

   Procedure searchFile (archivo: array [1..12] of char; inodo : int; var nro_inodo: int; Var ok: boolean);

   Donde $archivo$ es el nombre del archivo a buscar, $inodo$ es el número de inodo del directorio donde buscar el archivo, nro_inodo es para retornar el número de inodo buscado y ok es para retornar si la operación se concretó con éxito o no.

   Asumir que se dispone de una función que lee del disco el bloque pasado como parámetro: **readBlock(d: disk; block_num: 0..max_blocks_on_disk; var buff : block);**
3. Implementar una función que dado un camino absoluto (ej.: /home/sistoper/archivo.txt) retorne el número de inodo correspondiente.

   Procedure getInode(cam: array of char; var nro_inodo: int; var ok: boolean);

   Donde cam es el camino absoluto, nro_inodo es el número de inodo del archivo referenciado y ok es para devolver si la operación se concretó con éxito o no.

Ejercicio 3 (avanzado) Se desea implementar un sistema de archivos siguiendo un modelo de asignación en forma de lista. El sistema debe soportar un estructura jerárquica de directorios en forma de árbol. Suponiendo que el tamaño de cada bloque de disco es de 1 KiB, que se tiene una cantidad `CANT_BLOQUES` de bloques disponibles y que se cuenta con los siguientes tipos definidos:

```
type sector = array [0..1023] of byte;
type disk = array [0..(CANT_BLOQUES-1)] of sector;
type fat = array [0..(CANT_BLOQUES-1)] of integer;

type entrada_dir = Record
  ...
end; // 32 bytes
```

(a) Complete la definición de `entrada_dir` sabiendo que su estructura ocupa 32 bytes de memoria y defina todo lo necesario para trabajar con el sistema planteado.

Dadas las estructuras definidas en la parte (a) se pide responder:

(b) ¿Cuál es el tamaño máximo de un archivo?

(c) ¿Cuál es la cantidad máxima de archivos soportados?

(d) Mencione una ventaja y una desventaja de aumentar el tamaño del bloque de disco (suponiendo que la capacidad total del disco se mantiene constante).

(e) Defina e implemente una función que elimine un archivo dada su ruta absoluta.
Suponga que para su implementación cuenta con tres funciones auxiliares: una para leer un bloque desde el disco, una similar para escribir un bloque y una para partir la ruta absoluta del archivo. Debe definir estas funciones auxiliares pero no es necesario que las implemente.

Ejercicio 4 (avanzado) Un sistema de archivos utiliza una estrategia indexada multinivel de dos niveles y un mapa de bits para administrar el espacio libre del disco. En la estructura indexada se dispone de 7 bloques de primer nivel y 1 bloque de indirección simple. A continuación se presentan las estructuras de datos utilizadas por este sistema de archivos.

```
const MAX_BLOQUES = 65536;
const MAX_INODOS = 8192;
```

```
type bloque = array [0..1023] of byte;  // 1024 bytes
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;
type entrada_dir = Record
    usado : boolean;                    // 1 bit
    nombre : array [0..123] of char;    // 124 bytes
    es_dir : boolean;                   // 1 bit
    inodo_num : int16;                  // 2 bytes
    permisos : array [0..13] of bit;    // 14 bits
End; // 128 bytes

type inodo = Record
    usado : boolean;                    // 1 bit
    inodo_num : int16;                  // 2 bytes
    es_dir : boolean;                   // 1 bit
    tamaño : int32;                     // 4 bytes
    directo : array [0..6] of int16;    // 14 bytes
    directo_tope : int16;               // 2 bytes
    indirecto : int16;                  // 2 bytes
    indirecto_tope : int16;             // 2 bytes
    reservado : array [0..45] of bit;   // 46 bits
End; // 32 bytes

type inodos_tabla = array [0..MAX_INODOS-1] of inodo;
type disco = array [0..MAX_BLOQUES-1] of bloque;

var IT : inodos_tabla;
    MB : mapa_bits;
    D : disco;
```

Se sabe que:

- las variables `IT`, `MB`, y `D` son globales.
- el inodo número 0 es el directorio raíz.
- las entradas de los directorios son almacenadas utilizando un array de entradas.
- en el mapa de bits, los bloques libres son marcados con valor 1 (true).

Se dispone de los siguientes procedimientos:

- `Procedure leerBloque(d : disco; bloque_num : 0..MAX_BLOQUES-1; var buffer : array [0..1023] of byte; var ok : boolean);`
  Lee desde el disco `d` el bloque con índice `bloque_num` en la variable `buffer`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.
- `Procedure escribirBloque(d : disco; block_num : 0..MAX_BLOQUES-1; buffer : array [0..1023] of bytes; var ok : boolean);`
  Escribe en el bloque con índice `bloque_num` del disco `d` la información que se encuentra en la variable `buffer`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.
- `Procedure obtenerNombreDirectorio(camino : array of char; var dir : array of char);`
  Retorna en el parámetro `dir` el nombre del directorio que contiene el archivo referenciado en el parámetro `camino`.
  Por ejemplo: `obtenerNombreDirectorio('/home/sistoper/a.txt') = '/home/sistoper'`

- `Procedure obtenerNombreBase(camino : array of char; var base : array of char);`
  Retorna en el parámetro `base` el nombre del archivo o directorio referenciado en el parámetro `camino`.
  Por ejemplo: `obtenerNombreBase('/home/sistoper/a.txt') = 'a.txt'`

Se pide:

(a) Indique el tamaño máximo que puede tener un archivo utilizando el sistema de archivos planteado. Justifique.

(b) Implemente una función que encuentre y retorne un bloque libre en el disco. La función a implementar tiene la siguiente firma:
`Procedure buscarBloqueLibre(var bloque : 0..MAX_BLOQUES-1; var ok : boolean);`
El parámetro `bloque` es el número de bloque libre encontrado. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

(c) Implemente una función que retorne en la variable `inodo_num` el número de inodo correspondiente al camino absoluto `camino`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito. La función a implementar tiene la siguiente firma:
`Procedure obtenerInodo(camino : array of char; var inodo_num : integer; var ok : boolean)`

(d) Implemente una funcion que dado un inodo perteneciente a un directorio, retorne la cantidad de archivos y directorios que se encuentran dentro de ese inodo.
```
Procedure obtenerCantidad(inodo_num : integer; var ok : boolean,
      var cantidad_archivos: integer, var cantidad_directorios: integer);
```
Nota: La funcion **NO** debe ser implementada de forma recursiva.

Ejercicio 5 (avanzado) Se tiene un sistema de archivos híbrido. Por un lado, los directorios son almacenados usando una estrategia indexada simple con soporte para enlaces duros (hard links). Por otro lado, los archivos son almacenados usando una estructura FAT. Considere la siguiente estructura de datos:

```
1  const MAX_BLOQUES = 65536;
2  const MAX_INODOS = 8192;
3  type entrada_dir = Record
4    usado : boolean;                // 1 bit
5    nombre : array [0..24] of char; // 25 bytes
6    tipo : (file, dir);             // 1 bit
7    inodo_num : int;                // 2 bytes
8    fat_inicio : int;               // 2 bytes
9    tamaño : int;                   // 2 bytes
10   reservado : array [0..5] of bit; // 6 bits
11 End; // 32 bytes
12
13 type inodo = Record
14   usado : boolean;                // 1 bit
15   inodo_num : int;                // 2 bytes
16   datos : array [0..7] of int;    // 16 bytes
17   tope : int;                     // 2 bytes
18   referencias : int;              // 2 bytes
19   reservado : array [0..6] of bit; // 7 bits
20 End; // 23 bytes
21
22 type bloque = array [0..1023] of byte; // 1024 bytes
23 type mapa_bits = array [0..MAX_BLOQUES-1] of bit;
24 type inodo_tabla = array [0..MAX_INODOS-1] of inodo;
25 type fat_tabla = array [0..MAX_BLOQUES-1] of int;
```

```
26
27   var
28     FAT: fat_table;
29     IT : inodos_tabla;
30     MB : mapa_bits;
```

Notas generales:

- Las variables FAT, IT, y MB son globales.
- En la estructura entrada_dir el campo inodo_num es utilizado únicamente cuando el tipo es *dir*. El atributo fat_inicio y tamaño son utilizados únicamente cuando el tipo es *file*.
- El directorio raíz es el inodo con índice 0 (cero).
- En FAT el valor -1 indica el fin de archivo.
- El campo referencias es utilizado para contar la cantidad de enlaces.
- En MB el valor 0 (cero) indica un bloque libre y 1 indica un bloque ocupado.

Se dispone de los siguientes procedimientos:

- `Procedure leerBlq(blq_num: int, Var buff: bloque) : boolean`
  Lee de disco el bloque blq_num, pasado como parámetro, y carga el contenido leído en el parámetro de salida buff. Retorna el éxito de la ejecución de la operación.
- `Procedure escrBlq(blq_num : int, buff : bloque) : boolean`
  Escribe en el bloque blq_num, pasado como parámetro, la información que se encuentra en el parametro buff. Retorna el éxito de la ejecución de la operación.
- `Procedure parteCamino(camino: array of char, var base: array of char, var resto: array of char);`
  Retorna la primera parte de la ruta en el parámetro base, y las restantes partes en resto. Por ejemplo: `parteCamino('/users/so/a.txt', 'users', '/so/a.txt')`
  `parteCamino('/a.txt', 'a.txt', '')`
  `parteCamino('/', '', '')`

Se pide:

(a) Dada la estructura del sistema de archivos planteado, mencione y describa brevemente tres problemas de correctitud que pueden darse.

(b) Implementar una función que busque un directorio o archivo:
`Function search(cam: array of char): entrada_dir;`
Donde `cam` es la ruta completa al elemento del sistema de archivos buscado. Retorna una entrada de directorio que corresponde al elemento buscado y `null` en caso que la operación no se ejecute con éxito.

(c) Implementar una función que reemplace el contenido a un archivo:
`Function zeros(camArchivo: array of char):boolean;`
Donde `camArchivo` es el camino de un archivo cuyo contenido debe ser sobreescrito totalmente con ceros. Retorna `true` si la operación fue ejecutada con éxito. Esta función no crea archivos.

(d) Implementar una función que mueva un archivo con la siguiente definición:
`Function move(camOrigen: array of char, nomOrigen: array of char, camDestino: array of char, nomDestino: array of char): boolean;`
Donde `camOrigen` es la ruta completa al directorio donde se encuentra el archivo a mover, `nomOrigen` es el nombre del archivo a mover en el directorio origen, `camDestino` es la ruta completa al directorio destino y `nomDestino` es el nombre para el archivo en el destino. Retorna un booleano indicando si la operación fue realizada con éxito. Esta operación no mueve directorios ni crea directorios.