Sistemas Operativos
Práctico 9

Curso 2024

Ejercicio 1 Un sistema operativo administra sus archivos en disco utilizando el método de asignación indexada. Para esto, se dispone de las siguientes estructuras:

```
1  type block = array [0..511] of byte; // 512 bytes
2
3  type dir_entry = Record
4      name : array [0..11] of char;  // 12 * 8 bits
5      type : (file,dir);             // 1 bit
6      used : boolean;                // 1 bit
7      inode_num : int;               // 16 bits
8      perms : array [0..13] of bit;  // 14 bits
9  End;
10
11 type inode = Record
12     inode_num : int;                // 16 bits
13     used      : boolean;            // 1 bit
14     data      : array [0..4] of int;// 5 * 16 bits
15     tope      : 0..5;               // 3 bits
16     type      : (file, dir);        // 1 bit
17     size      : int;                // 16 bits
18     reserved  : array[0..10] of bit;// 11 bits
19 End;
20
21 type inode_table = array [0..max_inode_on_disk] of inode;
22 type disk = array [0..max_blocks_on_disk] of block;
23 Var
24   TI   : inode_table;
25   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.

(a) Implementar una función que retorne la cantidad de bytes utilizados por los archivos (type == file) del sistema de archivos.

(b) Implementar una función que busque un elemento (file o dir) dentro de un directorio (no búsqueda recursiva).

Procedure searchFile (elem: array [0..11] of char; inodo : int; var nro_inodo: int; Var ok: boolean);

Donde $elem$ es el elemento a buscar (file o dir), $inodo$ 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);

(c) 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.

Solución:

Consideración general: Una de las metas de un sistema operativo es administrar los recursos de hardware de forma eficiente. Por esta razón, siempre se debe hacer enfasis en implementaciones eficientes. Se debe evitar recorrer bucles innecesariamente, minimizar las lecturas/escrituras de disco, evitar implementaciones recursivas, etc.

Parte (a) El contenido del disco se encuentra accesible a través del array de bloques `D`. Estos bloques tienen el contenido binario de archivos y directorios. Sin embargo, no nos son útiles para implementar la función solicitada. La razón es que accediendo directamente a estos bloques no seríamos capaces de distinguir archivos de directorios. Lo que es más, ni siquiera podríamos distinguir bloques usados de libres porque entre las estructuras dadas por la letra no contamos con un mapa de bits para auxiliarnos.

Es necesario acceder a la estructura de inodos `TI` para buscar los inodos de tipo file. Tenemos dos alternativas para acceder esta estructura: (1) recorrer el árbol de directorios a partir de la raíz, i.e. `TI[0]`, o (2) recorrer de forma plana el array (desde `TI[0]` hasta `TI[max_inode_on_disk]`). La recorrida (1) es definitivamente más eficiente, y lo es aún más cuantos menos archivos y directorios existan en el sistema. Con esta recorrida nuestro algoritmo examinará tantos inodos como la cantidad de archivos y directorios se encuentren creados en el sistema. La recorrida (2) es más ineficiente porque requiere examinar siempre `max_inode_on_disk + 1` inodos, independientemente de cuantos archivos y directorios existan en el sistema. Sin embargo la recorrida (2) es más fácil de implementar que la recorrida (1). En este caso (y luego de la discusión planteada) decidimos implementar una recorrida de tipo (2).

Por último nos queda preguntarnos que se espera cuando se pide que la función implementada retorne la *cantidad de bytes utilizados por los archivos*. La cantidad de bytes utilizados por un archivo es: (1) la cantidad de bytes que requiere este archivo para almacenar su contenido, o (2) la cantidad de bytes que este archivo ocupa realmente en el disco? (recordemos que los discos sufren de fragmentación interna por ser un dispositivos orientados a bloques). Si la respuesta es (1), entonces debemos sumar los tamaños reportados en el campo `size` de los archivos. En cambio, si la respuesta es (2), deberíamos sumar la cantidad de bloques asignados a los archivos y multiplicarlo por el tamaño de cada bloque (512 bytes). En este caso (y luego de discutir nuevamente) decidimos que se trata de (1).

En base a estas decisiones, nuestra implementación de la función solicitada es la siguiente:

```
1  Procedure BytesUsadosPorArchivos(Var tamaño : Integer);
2  Var i : Integer;
3  Begin
4    tamaño = 0;
5    For i = 0 to max_inode_on_disk do
6    Begin
7      If ((TI[i].used) and (TI[i].type == file)) then
8        tamaño += T[i].size;
9      End if;
10   End for;
11 End;
```

- Línea 7: Es muy importante verificar `used=true` para no sumar el tamaño de archivos que ya fueron borrados.

Parte (b) Para esta segunda parte no podemos simplemente buscar un elemento (file o dir) de nombre `elem` iterando entre los inodos de `TI` porque en el sistema de archivos pueden existir

muchos elementos con el mismo nombre, cada uno en un directorio diferente. Nuestra implementación debe encontrar un elemento en particular, el que se encuentra dentro del directorio dado por el inodo `inodo`. Este elemento es único porque el nombre de cada elemento debe ser único en el directorio que lo contiene.

Para recorrer el contenido de un directorio es necesario recorrer sus bloques. En cada bloque asignado a un directorio se encuentra un array de `dir_entry`. Estos `dir_entry` dan nombre y apuntan al inodo de cada uno de los archivos y sub-directorios contenidos en cada directorio. En este caso particular, como la estructura `dir_entry` ocupa 16 bytes y cada bloque tiene una capacidad de 512 bytes (dado por la letra), cada bloque asignado a un directorio contendrá un array de `dir_entry` con $512/16 = 32$ entradas.

```
1  Procedure searchFile(elem: Array [1..12] of char; inodo: Integer;
2                       Var nro_inodo: Integer; Var ok: Boolean);
3  Var
4    size, block, entry : Integer;
5    buff : Array [0..31] of dir_entry;
6  Begin
7    If (ok = (TI[inodo].type == dir and elem <> "") then
8      block = 0; ok = false;
9      While (not ok) and (block < TI[inodo].tope) do
10     Begin
11       readBlock(D, TI[inodo].data[block], buff);
12       entry = 0;
13       While (not ok) and (entry < 32) do
14       Begin
15         If (ok = (buff[entry].used and
16            buff[entry].name == elem)) then
17
18            nro_inodo = buff[entry].inode_num;
19            ok = true;
20         End if;
21         entry++;
22       End while
23       block++;
24     End while;
25   End if;
26 End;
```

- Línea 7: Siempre se deben verificar los parámetros de entrada. En este caso, el `inodo` pasado por parámetro debe ser un directorio y el nombre del elemento buscado no debe ser vacío.
- Línea 9: Este bucle itera con variable `block` entre todos los bloques asignados al directorio. El identificador de cada bloque asignado se encuentra almacenado en el array `data` y la cantidad de bloques asignados está determinado por el campo `tope`.
- Línea 11: Se lee desde del disco `D` el contenido del bloque apuntado por la posición `block` del array `data` y se almacena en un array de `dir_entry` de tamaño 32. Como ya discutimos, cada bloque de un directorio contiene 32 elementos `dir_entry`.
- Línea 13: En cada bloque se itera con la variable `entry` entre sus `dir_entry`.

- Líneas 15-17: En caso de que el `dir_entry` actual esté en uso y su nombre asignado coincida con el elemento buscado, entonces se encontró el elemento buscado y se guarda su número de inodo.
- Si no verificamos `used=true` podríamos encontrar una referencia a un archivo o directorio que fue borrado y que tenía nombre igual al que buscamos.

Parte (c) Para esta última parte asumiremos que existen las funciones `basename` y `dirname` que, dado un camino, retornan el nombre del archivo (o sub-directorio) y el directorio que los contiene, respectivamente. Por ejemplo:

- `basename('/home/sistoper/archivo.txt')` retorna `'archivo.txt'`
- `dirname('/home/sistoper/archivo.txt')` retorna `'/home/sistoper'`.
- `basename('/home/sistoper')` retorna `'sistoper'`
- `dirname('/home/sistoper')` retorna `'/home'`
- `basename('/home')` retorna `'home'`
- `dirname('/home')` retorna `'/'`.

La implementación de estas funciones es trivial y en general se indica explicitamente que se pueden asumir implementadas aunque en este caso la letra no lo hace explicito. (**Nota:** Siempre consultar a su docente de confianza antes de asumir cualquier supuesto)

Si usamos la función implementada en la parte anterior es casi inmediata y natural una solución recursiva. Una **solución recursiva** podría ser como se muestra a continuación:

```
1  Procedure getInode(cam : array of char; Var nro_inodo : Integer;
2                     Var ok : Boolean);
3  Var
4    base : array of char;
5    dir  : array of char;
6    dir_inodo: Integer;
7  Begin
8    If (cam == "/") then
9      ok = true;
10     nro_inodo = 0;
11   Else
12     base = basename(cam);
13     dir = dirname(cam);
14     getInode(dir,dir_inodo,ok);
15     If (ok) then
16       searchFile(base,dir_inodo,nro_inodo,ok);
17     End if;
18   End if;
19 End;
```

Si bien esta solución es correcta, no es del todo aceptable debido a su ineficiencia. Una implementación recursiva *nunca* es buena idea desde el punto de vista de la eficiencia. Por esta razón lo adecuado es realizar una implementación iterativa. Si asumimos la existencia del TAD `Pila` (previa consulta con nuestro docente de confianza), una solución iterativa podría ser como la que se muestra a continucación:

```
1  Procedure getInode(cam : array of char; Var nro_inodo : Integer;
2                     Var ok : Boolean);
3  Var
4    base : array of char;
5    pila : Pila;
6    nro_inodo : Integer;
7  Begin
8    empty(pila);
9    While (cam <> "/") do
10     base := basename(cam);
11     push(pila, base);
12     cam  := dirname(cam);
13   End while;
14
15   ok = true;
16   nro_inodo = 0;
17   While (not isEmpty(pila) and ok) do
18     base = pop(pila);
19     searchFile(base,nro_inodo,nro_inodo,ok);
20   End while;
21 End;
```