**EJERCICIOS DE FINAL – FILE SYSTEM**

**¿Qué es el VFS, cuáles son sus componentes y cómo se relacionan?**

El VFS (virtual File System) es el subsistema del kernel que implementa las interfaces que tienen que ver con los archivos y sistema de archivos provistos a los programas corriendo en modo usuario. Todos los sistemas de archivos deben basarse en VFS para:

- Coexistir

- Inter-operar

Esto habilita a los programas a utilizar las system calls de UNIX para leer y escribir en diferentes sistemas de archivos y diferentes medios. Ejemplo open(), read() y write() funcionan sin que estas necesiten tener en cuenta el hardware subyacente.

**Diagrama

Descripción generada automáticamente**

(comando copy se comunica con VFS para realizar la copia del disco removible al HDD)

El VFS proporciona la capa de abstracción de archivos a los programas en user mode. Esta capa trabaja mediante la definición de interfaces básicas y de estructuras que cualquier sistema de archivos soporta.

**Diagrama

Descripción generada automáticamente**

Las componentes del VFS son:

* **Super bloque**, que representa un sistema de archivos. Existe en disco y en memoria. En disco provee información al kernel de la estructura del file system. En memoria provee la información necesaria y los estados para manipular al file system montado.
* **i-node**, que representa un determinado archivo dentro del disco. Un inode puede referenciar a un archivo, un directorio o un link simbólico a otro objeto. El inode consiste de data y operaciones que describen sus contenidos y las operaciones que pueden realizarse en él (ej. open, read, write).
* **dentry**, que representa una entrada de directorio, que es un componente simple de un path. El file system tiene un dentry root, el único que no va a tener padre. Si un archivo tiene el path "/home/user/name", se van a crear cuatro dentries: "/", "/home", "/user", "/name". Vive en memoria, no en disco.
* **file**, que representa a un archivo asociado a un determinado proceso.

Diagrama

Descripción generada automáticamente

Los procesos abren objetos de tipo file. Para acceder al archivo realmente guardado en disco, tienen que entrar en la dentry cache, que lo que hace es darte la ruta que termina en un objeto inodo (el dentry es un archivo que relaciona el inodo con el nombre del archivo). Este objeto inodo es parte del superbloque.

Observación, un directorio es tratado como un archivo normal, no hay un objeto específico para los directorios. En UNIX, son archivos normales que listan los archivos contenidos en ellos.

Luego existe un conjunto de operaciones:

* super\_operations, métodos que aplica el kernel sobre un determinado sistema de archivos, por ejemplo write\_inode() o sync\_fs().
* inode\_operations, métodos que aplica el kernel sobre un archivo determinado, por ejemplo create() o link().
* dentry\_operations, métodos que se aplican directamente por el kernel a un determinado directory entry, como por ejemplo, d\_compare() y d\_delete().
* filte\_operations(), métodos que el kernel aplica directamente sobre un archivo abierto por un proceso, read() y write() por ejemplo.

Diagrama

Descripción generada automáticamente

No sé si está del todo bien, pero por lo que entendí en los i-node se guarda la metadata sobre los datos que están en disco, el superbloque tiene la información de todo el file system (cantidad de inodes, cantidad de bloques, donde comienzan las tablas, etc). El dentry es info sobre un directorio, y un file es el archivo en cuestión asociado a un proceso, que este tendrá un dentry, que el dentry tendrá un i-node asociado al archivo.

**¿Qué es un hardlink, softlink, un volumen y un mount point?**

* **Hardlink**: Es el mapeo entre el nombre del archivo y el archivo en sí, los hardlinks se refieren al mismo archivo a través de un mismo inodo, por lo tanto, cada enlace es una copia exacta tanto de datos, como permisos, propietario, etc.
* **Softlink**: El softlink a diferencia del hardlink crea un inodo completamente nuevo, sería como un acceso directo de windows ya que apunta al mismo archivo, pero con otro inodo. A diferencia del hardlink este enlace también se puede dar con directorios. Y además si se elimina el enlace no se eliminará el auténtico.
* **Volumen**: Es una abstracción que corresponde a un disco lógico, en el caso más es un disco correspondiente a un disco físico, podría ser un pendrive también. En síntesis, es una colección de recursos físicos de almacenamiento.
* **Mount Point**: Es un punto en el cual el root de un volumen se engancha dentro de la estructura de un file system ya existente.

**Describa el API del sistema de archivos. Diferencia entre las system calls y library calls, ponga un ejemplo.**

Las system calls de archivos se pueden dividir en dos partes:

* las que operan sobre los archivos:
  + open(): convierte el nombre de un archivo en una entrada de la tabla de descriptores de archivos, y devuelve dicho valor.
  + creat(): equivale a llamar a open() con los flags O\_CREAT|O\_WRONLY|O\_TRUNC.
  + close(): cierra un file descriptor.
  + read(): se utiliza para hacer intentos de lecturas hasta un número dado de bytes de un archivo.
  + write(): escribe hasta una determinada cantidad (count) de bytes.
  + lseek(): reposisiona el despazamiento (offset) de un archivo abierto.
  + dup(): crea una copia del file descriptor del archivo.
  + link(): crea un nuevo nombre para un archivo.
  + unlink(): elimina un nombre de un archivo del sistema de archivos.
* las que operan sobre los *metadatos* de los archivos
  + stat(): devuelven información sobre un archivo.
  + access(): chequea si un proceso tiene o no los permisos para utilizar el archivo.
  + chmod(): cambian los bits de modos de acceso.
  + chown(): cambian el id del propietario del archivo y el grupo de un archivo.

Una system call es implementada en el kernel space, en cambio una library call es implementada en el user space. Por ejemplo, en C existe la library call fopen que internamente llama a la system call open, pero devuelve un objeto FILE.

**Describa la organización general de un file system de 64 bloques de 5 kbytes, 56 de los cuales son bloques de datos.**

La visión del sistema de archivos debe ser la de una partición de 64 bloques (de 0 a 63) de un tamaño de 64 \* 4 KB bloques.

Diagrama

Descripción generada automáticamente

Desde el bit 8 al 63 es ocupado por datos

Texto

Descripción generada automáticamente con confianza media

Dentro de los primeros 8 bits debe guardarse la información sobre cada uno de los archivos (metadata). Es importante porque mantiene información de qué bloque de datos pertenece a un determinado archivo, el tamaño del archivo, etc. Esta información se guarda en una tabla llamada inode table (array de inodos).

Imagen que contiene Texto

Descripción generada automáticamente

Suponiendo que los inodos ocupan 256 bytes , un bloque de 4KB puede guardar 16 inodos por ende nuestro sistema de archivo tendrá como máximo 80 inodos. Esto representa también la cantidad máxima de archivos que podrá contener nuestro sistema de archivos.

En el bit 1 se va a a guardar el inode bitmap y en el bit 2 el data bitmap. En estos bitmaps se mapea 0 si un objeto está libre y 1 si el objeto está ocupado.

En el bit 0 se guarda el super bloque, este contiene la información de todo el file system, incluyendo: cantidad inodos, cantidad de bloques, dónde comienza la tabla de inodos, dónde comienzan los bitmaps.

Tabla

Descripción generada automáticamente con confianza baja

**¿ Qué es un inodo ? ¿ Qué es un superbloque ? ¿ Qué es un bitmap?**

* **Super bloque**: representa a un sistema de archivos. Existe en disco y en memoria. En disco provee información al kernel de la estructura del file system. En memoria provee la información necesaria y los estados para manipular al file system montado.
* **Inodo**: representa a un determinado archivo dentro del disco. Un inode puede referenciar a un archivo, un directorio o un link simbólico a otro objeto. El inode consiste de data y operaciones que describen sus contenidos y las operaciones que pueden realizarse en él (ej. open, read, write). Un inodo simplemente es referido por un número llamado inumber que sería lo que hemos llamado el nombre subyacente en el disco de un archivo. Dado un inumber se puede saber directamente en que parte del disco se encuentra el inodo correspondiente.
* **Bitmap**: es una estructura bastante sencilla en la que se mapea 0 si un objeto está libre y 1 si el objeto está ocupado.

**Suponiendo que BigFS es una variante de FSS, el clásico sistema de archivos de unix, el cual posee 12 referencias a i-nodos directas, 1 indirecta, 1 doble indirecta, una triple indirecta y una cuádruple indirecta. Asumiendo bloques de 4kb, 8 byte por puntero a bloques. Cuál es el máximo tamaño de archivo que se soporta?**

Bloques de 4 kb, 8 bytes por puntero a bloques ⇒ 2^12/2^3 = n = 512 bloques

12 directas: referencia 12 bloques

1 indirecta: referencia n bloques

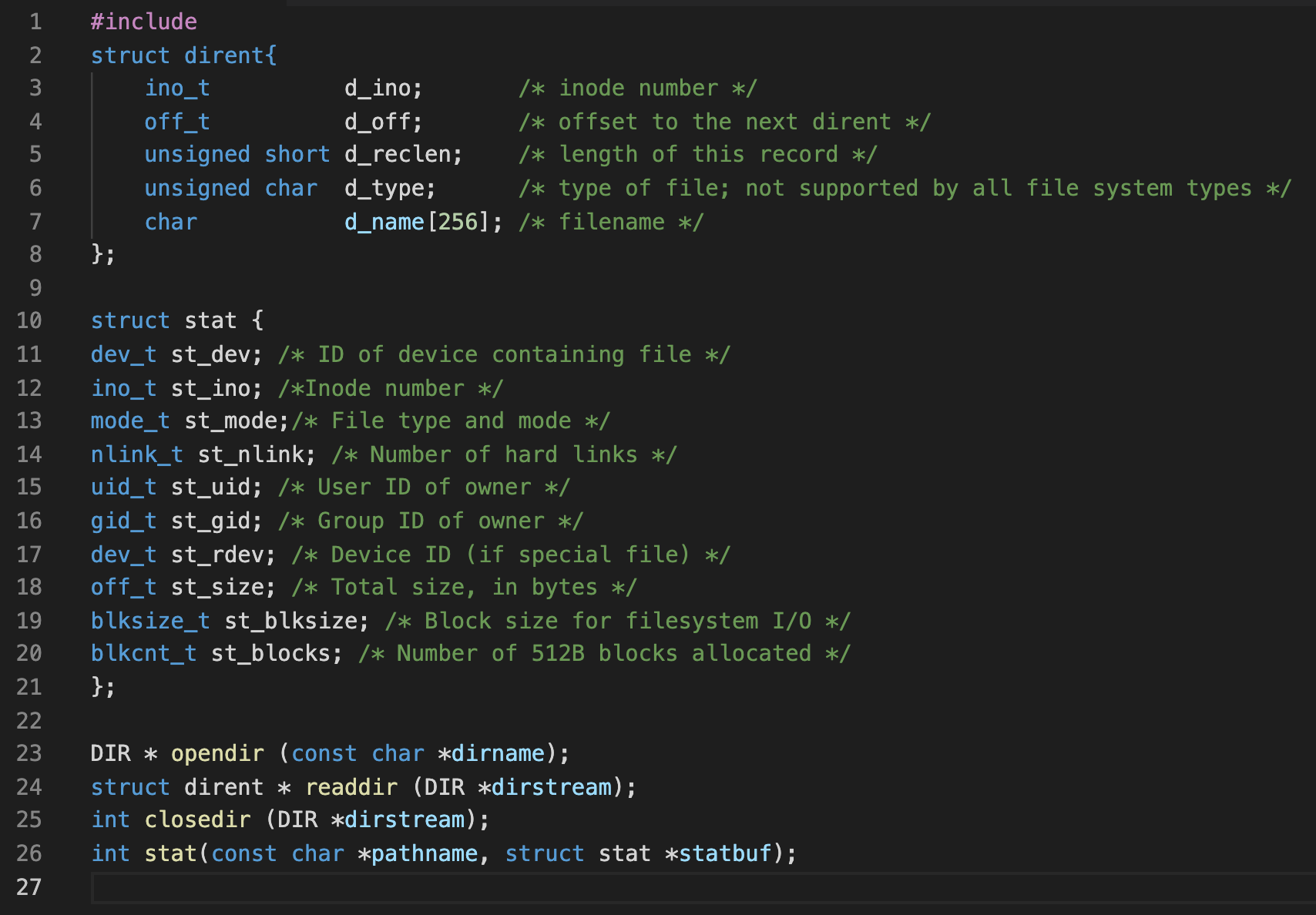
1 doble indirecta: referencia n^2 bloques

1 triple indirecta: referencia n^3 bloques

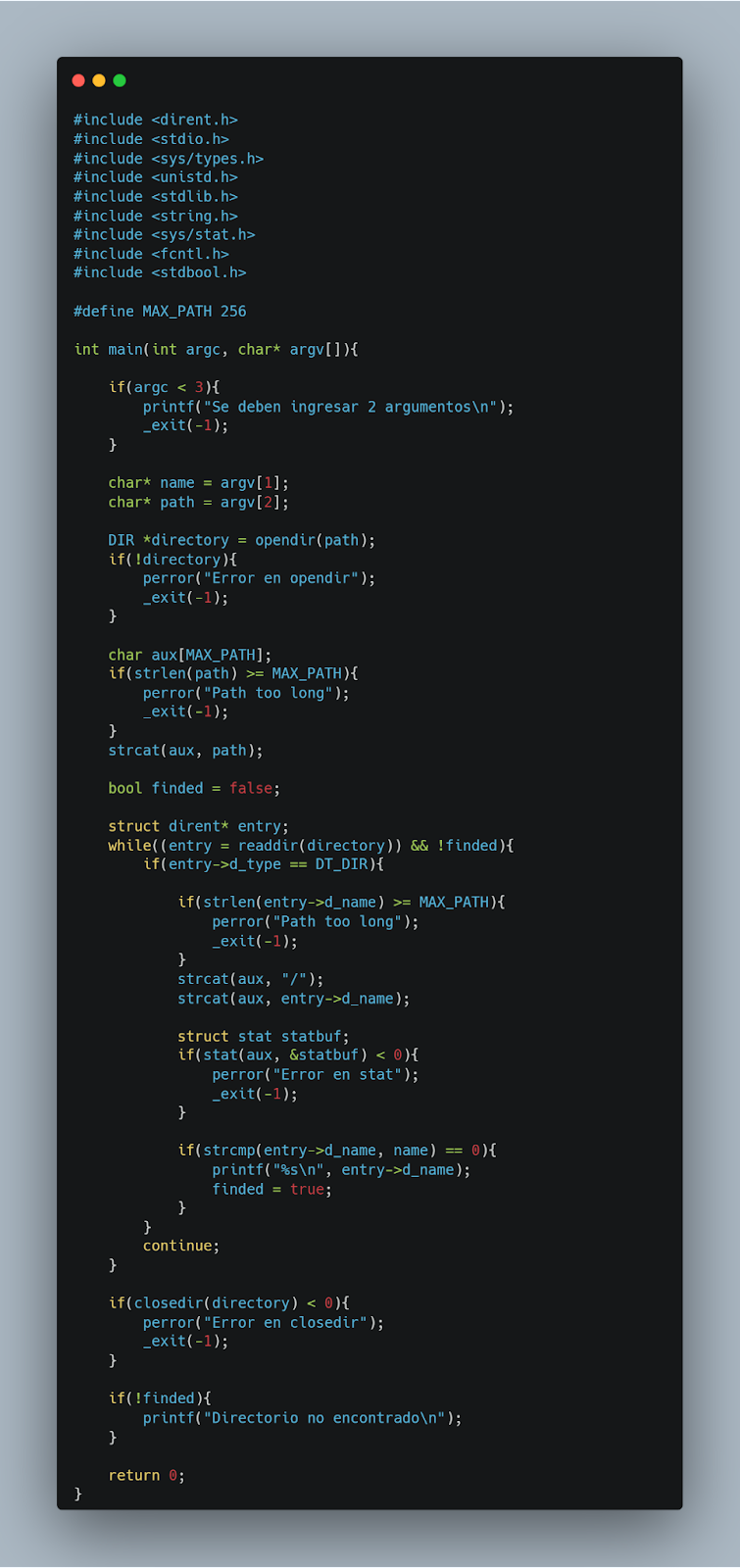
1 cuádruple indirecta: referencia n^4 bloques ⇒ Máximo tamaño de archivo = (12 + 512 + 512^2 + 512^3 + 512^4) \* 2^12 = 282 TB

[tamaño total de archivo = tamaño bloque \* cantidad bloques]

**Construya un comando con las siguientes syscalls:**



**Crear un comando linux que dado un nombre y un path recorra el mismo buscando si existe un directorio con ese nombre y lo muestre**.



**Sea un disco que posee 128 bloques de 4 kb y un sistema operativo cuyos i-nodos son de 128 bytes. Defina la estructura completa del sistema de archivos unix-like. Justificar cada elección.**

Para definir una estructura completa del sistema de archivos unix-like necesitamos asignar espacio a un super bloque, un bitmap de datos, un bitmap de inodos, una región de datos y una tabla de inodos.

Mínimamente se puede asignar un bloque a cada estructura, por lo que si bien los bitmaps van a ocupan mucho menos espacio necesariamente le tenemos que asignar 1 bloque.

Estructura:

- Bloque 0 -> Super bloque

- Bloque 1 -> Bitmap de inodos

- Bloque 2 -> Bitmap de datos

Para definir una estructura completa del sistema de archivos unix-like necesitamos asignar espacio a un super bloque, un bitmap de datos, un bitmap de inodos, una región de datos y una tabla de inodos.

Mínimamente se puede asignar un bloque a cada estructura, por lo que si bien los bitmaps van a ocupan mucho menos espacio necesariamente le tenemos que asignar 1 bloque.

Estructura:

- Bloque 0 -> Super bloque

- Bloque 1 -> Bitmap de inodos

- Bloque 2 -> Bitmap de datos

Faltan definir los bloques a asignar a la tabla de inodos y a la región de los datos.

Cada bloque ocupa 4 KB (4096 bytes) por lo que si los inodos ocupan 256 bytes, se disponen de 4096/256=32 inodos por bloque. Por otro lado, la cantidad máxima de inodos que se pueden tener en esta estructura de 128 bloques es de 128 inodos. Pues cada archivo ocupa mínimamente 1 bloque y está asociado a un inodo. La cantidad de inodos disponibles puede ser inferior a 128 (cantidad total de bloques) en el caso de que existan archivos grandes en el sistema que ocupen más de un bloque.

Quedan asignar 128-3=125 bloques de los cuales la mayor cantidad va a estar asociada a la región de datos y se debe asignar la mínima cantidad de bloques tal que se almacenen todos los inodos disponibles. Si asignamos arbitrariamente 10 bloques a la tabla de inodos, podemos almacenar 10\*32=320 inodos y nos quedarían disponibles 125-10=115 bloques para asignar a la región de datos. Sin embargo, necesitamos 320-115=205 inodos por lo que estamos desperdiciando 205/32=6 bloques enteros asociados a inodos ya que no se van a ocupar. Tenemos, entonces, que disminuir la cantidad de bloques asignados a inodos ya que esta estructura no resulta óptima. Si asignamos 4 bloques a inodos podemos almacenar hasta 4\*32=128 inodos. Quedarían asignados 125-4=121 bloques a la región de datos por lo que necesitamos mínimamente 121 inodos. 128-121=7 < 32 por lo que no estamos desperdiciando bloques.

Finalmente, podemos asignar 4 bloques a la tabla de inodos y el resto de los 128-3-4=121 bloques a la región de datos.

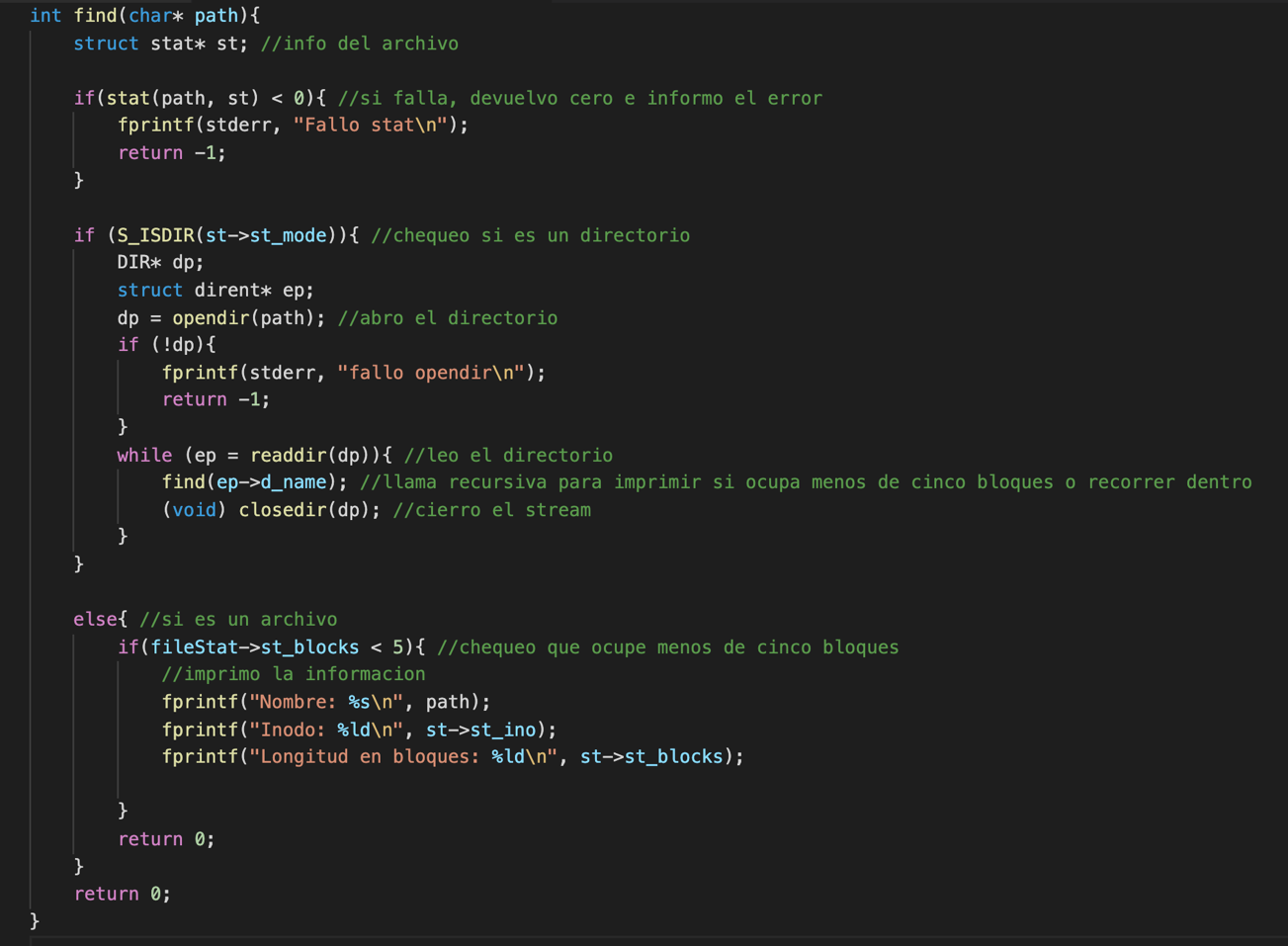
- Bloques 3,4,..,6 -> inodos

- Bloques 7,11,..,127 -> datos

**Construya un comando con las siguientes syscalls**

### 

### Crear un comando linux que dado un directorio busque en profundidad todos los archivos que ocupen menos de 5 bloques de datos y muestre nombre, inodo y longitud en bloque.



**Sea un disco que posee 256 bloques de 4kb y un sistema operativo cuyos i-nodos son de 512 bytes. Defina la estructura completa del sistema de archivos unix-like . Justificar cada elección.**

Los inodos ocupan 512 bytes, con lo cual en cada bloque entran 4096/512 = 8 inodos. Necesitamos tener, al menos, 256 inodos para cubrir todos los bloques del disco. Por lo tanto, necesitariamos 256/8 = 32 bloques para inodos.

El primer bloque se utiliza para el superbloque. El segundo y tercero para el inode-bitmap y data-bitmap respectivamente. Estos bitmaps indican

cuales bloques e inodos estan utilizados o libres.

Luego vienen los 32 bloques de inodos y el resto de data.

En resumen, tenemos que:

Bloque 0: Superbloque.

Bloque 1: inode-bitmap.

Bloque 2: data-bitmap.

Bloques 3-34: inode blocks.

Bloques 35-255: data blocks.

**Sea un disco que posee 512 bloques de 8kb y un sistema operativo cuyos i-nodos son de 256 bytes. Defina la estructura completa del sistema de archivos unix-like Justificar cada elección.**

Para definir una estructura completa del sistema de archivos unix-like necesitamos asignar espacio a un super bloque, un bitmap de datos, un bitmap de inodos, una región de datos y una tabla de inodos.

Mínimamente se puede asignar un bloque a cada estructura, por lo que si bien los bitmaps van a ocupar mucho menos espacio necesariamente le tenemos que asignar 1 bloque.

Estructura:

- Bloque 0 -> Super bloque

- Bloque 1 -> Bitmap de inodos

- Bloque 2 -> Bitmap de datos

Quedan asignar 512-3=509 bloques a la región de datos y a la tabla de inodos.

Cada bloque ocupa 8 KB (8192 bytes) por lo que si los inodos ocupan 256 bytes, se disponen de 8192/256=32 inodos por bloque.

Por otro lado, la cantidad máxima de inodos que se pueden tener en esta estructura de 512 bloques es de 512 inodos. Pues cada archivo ocupa mínimamente 1 bloque y está asociado a un inodo. La cantidad de inodos disponibles puede ser inferior a 512 (cantidad total de bloques) en el caso de que existan archivos grandes en el sistema que ocupen más de un bloque.

Si asignamos arbitrariamente 16 bloques a la tabla de inodos, podemos almacenar 16\*32=512 inodos y nos quedarían disponibles 509-16=493 bloques para asignar a la región de datos por lo que necesitamos mínimamente 493 inodos. 512-493=19 < 32 por lo que no estamos desperdiciando bloques.

Finalmente, podemos asignar 16 bloques a la tabla de inodos y el resto de los 512-3-16=493 bloques a la región de datos.

- Bloques 3,4,..,18 -> tabla de inodos

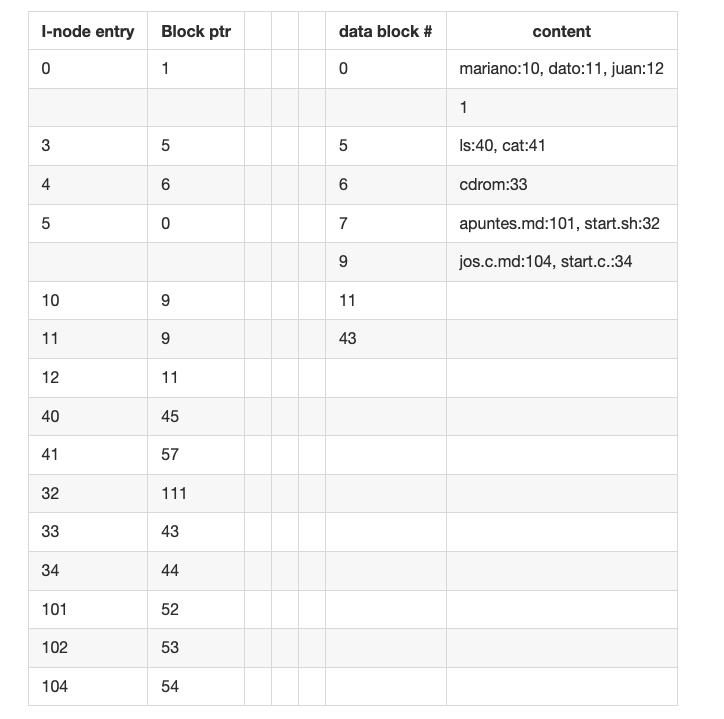
- Bloques 19,12,..,511 -> región de datos

**Se tiene un file system basado en i-nodos con la siguientes características:**

* **Los bloques son de 1 kiB (1024 bytes) [2^10]**
* **El tamaño de un i-nodo es de 64 bytes [2^6]**
* **La distribución de los bloques es:**
* **El bloque 0 es el boot\_block**
* **El bloque 1 es el superblock**
* **El bloque 2 es el i-node bitmap**
* **El bloque 3 es el block-bitmap**
* **Hay 126 bloques dedicados a la i-node table**
* **Hay 128 bloques dedicados a datos**

**Dada la siguiente información de la tabla de i-nodos y el contenido de los bloques de datos, indicar:**

1. **¿Qué se mostraría en pantalla o que equivale ejecutar ls /bin , ls /home/juan, ls /home/mariano? Indicar la secuencia de operaciones (lecturas de bloque blkrd indicando la numeración relativa a la sección de i-nodos o datos; y la numeración dentro del sistema entero), que se realizan para acceder al archivo /home/dato/start.sh. Indicar para cada bloque leído qué información contiene y qué parte resulta relevante.**
2. **¿Hay algún archivo que tenga más de una referencia (hard link)? ¿Qué syscall o comando unix usaría para borrar este tipo de archivos?**



- ls /bin:

ls cat

- ls /home/juan:

(\*Vacio\*)

- ls /home/mariano:

jos.c.md start.c

Cada bloque de inodos contiene 1024/64 = 16 inodos. Para describir en que bloque se encuentran, utilizare la notacion [A|B], siendo A el bloque relativo a los bloques de inodos y B el absoluto a todos los bloques,teniendo en cuenta el sistema entero.

Para intentar acceder a /home/dato/start.sh, se comenzara en el inodo 0 [0|4], que corresponde a la raiz.

Aqui vemos que es un directorio, con lo cual sabemos que los datos que contenga seran otros archivos.

Ademas, tenemos los permisos, con los cuales podemos ver si podemos acceder al mismo (es decir, si tiene el permiso "search", que es el equivalente a "execute" para los archivos). El inodo contiene

un puntero al bloque de datos 1 (abs 131).

Si vamos a ese bloque, obtenemos una lista de archivos con sus correspondientes inodos. Buscamos el que

necesitamos ("home") y vamos al inodo que indica.

En el inodo 5 [0|4], leemos nuevamente que estamos en un directorio, y chequeamos si tenemos los permisos

correspondientes. En caso de ser asi, proseguimos al bloque de datos indicado por el puntero, el 0 (abs 130).

Ahi nos encontramos nuevamente con una lista donde aparecen "mariano", "dato" y "juan". Nos interesa "dato",

asi que vamos al inodo indicado en el mismo, el 11 [0|4].

Una vez mas, vemos que es un directorio y verificamos permisos. El puntero apunta al bloque de datos 9 (abs 139).

Si vamos al mismo, encontramos una lista de archivos con nombres "jos.c.md" y "start.c".

Ninguno es el archivo que necesitamos, con lo cual arrojariamos un error indicando que el archivo

solicitado no existe.

Ademas de los permisos y del tipo de archivo, en los inodos tenemos otros datos como el user ID y group ID,

fechas de creacion, ultimo acceso y modificacion, tamanio (en bytes y bloques) que ocupa el archivo,

hard links que contiene, entre otros.

2. El bloque de datos 11 es el unico que cuenta con dos referencias. Este bloque corresponde

a los directorios /home/mariano y /home/dato. Debido a como se organizan los inodos, no hay forma de

saber cual es un link al otro.

Para borrar un archivo se puede utilizar la syscall unlink, la cual le resta una unidad a la cantidad

de links que tiene ese archivo. Cuando el contador llega a 0, es eliminado del sistema de archivos.

-------------------------------

(Mi Respuesta – Bien)



**Se tiene un file system basado en i-nodos con la siguientes características:**

* **Los bloques son de 1 kiB (1024 bytes) [2^10]**
* **El tamaño de un i-nodo es de 64 bytes [2^6]**
* **La distribución de los bloques es:**
* **El bloque 0 es el boot\_block**
* **El bloque 1 es el superblock**
* **El bloque 2 es el i-node bitmap**
* **El bloque 3 es el block-bitmap**
* **Hay 126 bloques dedicados a la i-node table**
* **Hay 128 bloques dedicados a datos**

**Dada la siguiente información de la tabla de i-nodos y el contenido de los bloques de datos, indicar:**

**¿Qué se mostraría en pantalla o que equivale ejecutar ls /bin , ls /home/juan, ls /home/mariano? Indicar la secuencia de operaciones (lecturas de bloque blkrd indicando la numeración relativa a la sección de i-nodos o datos; y la numeración dentro del sistema entero), que se realizan para acceder al archivo /home/dato/start.sh. Indicar para cada bloque leído qué información contiene y qué parte resulta relevante.**

**¿Explique solo los componentes de este sistema de archivos que se relacionan con el VFS, de un ejemplo de cada componente en este caso? ¿Que Funciones del Api de archivos debe utilizar para hacer el ejercicio?**

| **I-node entry** | **Block ptr** |  |  |  | **data block #** | **content** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 |  |  |  | 0 | mariano:4, dato:11, juan:12 |
|  |  |  |  |  | 1 | home:5, mnt:11, bin:3 |
| 3 | 5 |  |  |  | 5 | ls:40, cat:41, vim: |
| 4 | 11 |  |  |  | 6 | cdrom:33 |
| 5 | 0 |  |  |  | 7 | apuntes.md:101, start.sh:32 |
|  |  |  |  |  | 9 | [jos.c.md:104](http://jos.c.md:104/), start.c.:34 |
| 10 | 6 |  |  |  | 11 | examen.md:39, apuntes.txt:103 |
| 11 | 9 |  |  |  | 43 |  |
| 12 | 7 |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 39 | 44 |  |  |  |  |  |
| 40 | 45 |  |  |  |  |  |
| 41 | 57 |  |  |  |  |  |
| 32 | 111 |  |  |  |  |  |
| 33 | 43 |  |  |  |  |  |
| 34 | 44 |  |  |  |  |  |
| 101 | 52 |  |  |  |  |  |
| 102 | 53 |  |  |  |  |  |
| 103 | 99 |  |  |  |  |  |
| 104 | 54 |  |  |  |  |  |
|  |  |  |  |  |  |  |

Nota: la primer columna de la tabla de i-nodos refiere al número de i-nodo, mientras que la primer columna de la tabla de bloques hace referencia a la numeración de bloques relativa al los bloques de datos. Ayuda: recordar que el i-nodo 0 es siempre el directorio raíz.

El comportamiento o pasos a seguir para ver que se imprime en cualquiera de los casos pedidos, es el siguiente:

Primero se arranca por la raiz '/' que tal y como se aclara, siempre es el inodo 0.  
A partir de aca, buscamos dicho numero en la tabla, y asosciado tendra un puntero a un bloque de data.  
Procedemos a ver donde apunta ese puntero y vemos que se encuentra en el bloque de datos, tendremos el numerode inodo siguiente en nuestro path (para el primer caso seria 'home' por ejemplo) y repetimos nuevamente.  
Buscamos el inodo obtenido, su puntero a bloque asosciado, etc. asi hasta el final  
  
Ejecutar ls /home/mariano  
  
inodo 0  
bloque 1 -> home -> inodo 5  
bloque 0 -> mariano -> inodo 4  
bloque 11  
                                             
Se imprime:  
examen.md apuntes.txt

Ejecutar ls /home/juan  
  
inodo 0  
bloque 1 -> home -> inodo 5  
bloque 0 -> juan -> inodo 12  
bloque 7  
                                          
Se imprime:  
apuntes.md start.sh  
  
Ejecutar ls /bin  
  
inodo 0  
bloque 1 -> bin -> inodo 3  
bloque 5  
                     
Se imprime:  
ls cat vim  
  
  
2.  
En este caso tan simplificado podemos identificar varios componentes que se relacionan con el VFS.  
Primero que nada mencionaria al datablock, que es el lugar donde se almacena, como bien su nombre loindica, los DATOS. En esta simplificacion, un datablock podria ser cualquiera de los que vemos en la  
columna 'content', por ejemplo:  
      
    mariano:4,dato:11,juan:12  
  
El otro que tenemos es el inodo. El inodo en la VFS suele guardar la metadata, tamano de archivo,  que datablock pertenece a un determinado archivo, etc. Pero en este sistema vemos que solo se encarga de lo ultimo. Un ejemplo seria:  
  
    inode 0 | blockPtr 1  
      
Las diferencias con un VSF de verdad es que NO tenemos ningun bitmap   
que indique si un inodo o datablock esta ocupado. Ni tampoco un superbloque  
con informacion general del sistema como la cantidad de inodos, etc.  
  
Se utilizaran las siguientes syscalls para poder realizar el ejercicio:

opendir():Para ir 'abriendo' los directorios para poder trabajar con ellos  
readdir():Para leer la informacion de los directorios, y asi poder saber que hay dentro  de ellos.  
closedir() --> Cerrar el directorio una vez terminamos de usarlo

**Sea un disco que posee 128 bloques de 4kb y un sistema operativo cuyos i-nodos son de 256 bytes. Defina la estructura completa del sistema de archivos unix-like . Justificar cada elección.**

Utilizo VSFS como vimos en el apunte.  
  
Al contar con inodos de 256 bytes (2^8) y bloques de 4kb (2^12), entonces cada bloque de i-nodos puede guardar 16 i-nodos.  
  
Como tenemos 128 bloques, y cada bloque de inodos referencia a 16 de estos,  
necesitaremos 8 bloques de i-nodos.  
  
El bloque 0 va a ser el Super Bloque.  
  
El bloque 1 sera el bitmap de i-nodos, donde sabemos que 1 alcanza ya que  
tiene 2^15 bits en un bloque de 4kb, y esto sobra.  
  
El bloque 2 va a ser el bitmap de data, y de forma analoga, tiene espacio mas que suficiente.  
  
Los bloques 3 hasta el 10, seran la tabla de inodos, con los 8 bloques de 16 inodos cada uno.  
  
Los bloques restandes desde el 11 hasta el 127 son bloques de datos.  
  
No utilizamos boot block en esta implementación.