

AYUDANTÍA P1

Germán Leandro Contreras Sagredo

Ricardo Esteban Schilling Broussaingaray

IIC2333 [2018-2] - Sistemas Operativos y Redes

INTRODUCCIÓN

Los objetivos de esta ayudantía son:

1. Comprender la estructura de nuestro sistema de archivos.
2. Aclarar el funcionamiento esperado de la API `czfs`.
3. Resolver las dudas que surjan durante la explicación de lo pedido.

SISTEMA DE ARCHIVOS

- El disco es de tamaño 128MB con bloques de 2KB cada uno.
- Este tiene 65536 bloques, ordenados de manera secuencial.
- Un puntero a un bloque no es más que un `int` de 4 bytes, correspondiente al número de bloque.
- Cuando un bloque es asignado a una función específica, este se asigna **completamente**.
- El primer bloque **siempre** será nuestro directorio **root**.

El bloque de directorio corresponde a un directorio en nuestro sistema de archivos.

Cada entrada del directorio, es decir, cada archivo o subdirectorio dentro de este está representado como una secuencia de 16 bytes.

- Un byte para indicar si la entrada es inválida (0x01), válida y correspondiente a directorio (0x02) o válida y correspondiente a archivo (0x04). Cualquier otro valor del primer byte también representará una entrada inválida, no obstante, no se debería dar.
- 11 bytes que representan el nombre de la entrada en **ASCII**.
- 4 bytes que representan el puntero al **bloque índice** del archivo.

- Corresponden a los siguientes cuatro bloques del disco.
- Su función es indicar los bloques que están ocupados y los que están libres.
- Habrá un bit igual a 1 si el bloque correspondiente está ocupado y 0 en caso contrario.

Ejemplo

El **byte** número 123 (contando desde 0) del primer bloque de directorio es **0xA3** (10100011_2), por lo que inmediatamente sabemos que los bloques 984, 986, 990 y 991 están ocupados.

- Los primeros 5 bloques del disco **siempre** estarán ocupados.
- Los bloques de bitmap **siempre** deben reflejar el estado actual del disco.

- Es el primer bloque de un archivo.
- Posee 8 bytes de metadata. 4 bytes para el tamaño del archivo y 4 bytes para un timestamp.
- También tiene espacio para 509 punteros distintos, cada uno de 4 bytes, estos apuntan **exclusivamente** a bloques de datos.
- Al final de estos bloques, se reservan 4 bytes para un puntero a bloque de direccionamiento indirecto, de ser necesario.
- El orden de los bloques en el bloque índice dicta el orden de los bloques de datos del archivo.

- Similar al bloque índice, solo que **no** posee bytes para metadata o puntero final, por lo que este bloque tiene espacio para guardar 512 punteros a bloques de datos.
- Ya que no existe otro puntero a bloque indirecto, el **tamaño máximo** de un archivo es de $(509 + 512) * 2 = 2042\text{KB}$

- Utiliza la totalidad de su espacio para guardar los archivos.
- No pueden ser subasignados, es decir, cuando uno asigna un bloque de datos a un archivo, este se asigna en su totalidad y no pueden haber dos o más archivos compartiendo el mismo bloque.

CZFS API

- Cristian ruZ File System.
- La API debe estar implementada en un archivo `cz_API.c` con la interfaz llamada `cz_API.h`.
- Debe también incluirse un `main.c` donde se utilicen **todas** las funciones de su librería.
- Son libres también de subir sus propios archivos para agregarlos al disco.
- Dentro de su API, deben definir un **struct** llamado `czFILE`, el que representa un archivo abierto. Este es similar al **struct FILE** de la librería `stdio.h`. Son libres en cuanto a los datos que posee esta estructura.
- La API utiliza únicamente rutas **absolutas**.

- `void cz_mount(char* diskname)` Esta función se encarga de **montar** el disco, dejando como **variable global** la ruta al archivo binario correspondiente. **Siempre** es la primera función que corre en su archivo `main`.
- `void cz_bitmap()` Imprime el bitmap del disco previamente montado, imprimiendo un 1 por bloque ocupado y 0 por bloque libre, además de la cantidad de bloques ocupados y finalmente la cantidad de bloques libres.
- `int cz_exists(char* path)` Retorna 1 si `path` existe en el disco y 0 si no.
- `void cz_ls(char* path)` Similar al comando `ls` de Unix, imprime los contenidos de `path`. De haber un error, se debe explicar cuál es el problema.

- `czFILE* cz_open(char* path, char mode)` Abre un archivo y retorna un puntero a la instancia de `czFILE` que lo representa. De haber un error, se debe retornar `NULL`. El modo puede ser `'r'` para leer archivos existentes o `'w'` para escribir nuevos archivos.
- `int cz_read(czFILE* file_desc, void* buffer, int nbytes)` Lee los `siguientes` `nbytes` del archivo descrito por `file_desc` y lo guarda en un `buffer`. La función debe retornar la cantidad de bytes leídos o -1 si hubo algún error.
- `int cz_write(czFILE* file_desc, void* buffer, int nbytes)` Escribe los `nbytes` que se encuentren en el `buffer` al archivo descrito por `file_desc`. La función debe retornar la cantidad de bytes escritos o -1 si hubo algún error.

FUNCIONES DE MANEJO DE ARCHIVOS

- `int cz_close(czFILE* file_desc)` Función que cierra un archivo abierto previamente. Cuando un archivo es cerrado, este debe estar actualizado en el disco. Si hay un error debe retornar algo distinto de 0.
- `int cz_mv(char* orig, char* dest)` Función que mueve un archivo de una ruta a otra, notar que esta función **no modifica** los datos del archivo mismo en el disco. Retorna 0 si no hay errores.
- `int cz_cp(char* orig, char* dest)` Función que copia un archivo de una ruta a otra, notar que esta función **no** crea un soft link, sino que una copia que puede ser modificada independientemente a la copia original. Retorna 0 si no hay errores.
- `int cz_rm(char* path)` Función que elimina un archivo. Es muy importante que se liberen **todos** los bloques ocupados por el archivo.

- `int cz_mkdir(char *foldername)` Crea un directorio vacío en la ruta indicada. Debe preocuparse que la ruta sea válida. Retona un int distinto de cero frente a un error.
- `int cz_mvdir(char *foldername, char *dest)` Mueve un directorio de una ruta a otra. Ambas rutas deben ser válidas en el sentido que la primera debe ser un directorio y la segunda no debe existir en el disco. Retona un int distinto de cero frente a un error.
- `int cz_cpdir(char* foldername, char* dest)` Copia un directorio junto a **todos sus contenidos** recursivamente de un path a otro. Retona un int distinto de cero frente a un error.
- `int cz_rmdir(char* path)` Elimina un directorio junto a todos sus contenidos recursivamente. Retorna un int distinto de cero frente a un error.

FIN
