Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2333 - Sistemas Operativos y Redes

Proyecto 01

2º semestre 2020 - Profesor C. Ruz

Felipe Barría - Juan Pablo González - Irina Salazar - Lucas Zalaquett

---

OldSchool FileSystem Documentation

Funciones adicionales

- **print_ls()**: función que no recibe argumentos y retorna vacío. Imprime las carpetas que están en el directorio. Inicia un ciclo de 0 a 63, en el que lee en index y en array los 32 bytes correspondientes a cada subloque del bloque de direccionamiento (3 bytes a indice y bloque y 29 bytes del nombre de la carpeta/archivo). Si el índice es válido, imprime el nombre. Finalmente, posiciona el lector del disco en el inicio.
- **is_valid(unsigned char* bits)**: retorna los dos bits correspondientes del argumento que recibe.
- **get_folder_path(char* path, unsigned char new_path[29])**: recibe como argumentos la carpeta buscada path y un arreglo de bytes. Retorna la posicion de numero de bytes donde esta la carpeta buscada.
- **block_number(unsigned char* bits)**: retorna el número del bloque de los 3 bytes indice que estan en el subloque del bloque del directorio.
- **strip_path(char* path, unsigned char new_path[29], int i)**: borra todos los slash path (de la ruta). Recibe como argumentos el path del que borrar los slash, un arreglo de bytes donde se escribe el nuevo path y un int que dice la cantidad de slashes que tiene el path.
- **strip_new_path(unsigned char new_path[29])**: retorna un int que representa lo que hay que eliminar dentro del nuevo path que recibe.
- **print_bits(unsigned char val)**: imprime los bits de un byte que recibe como argumento.
- **bits_in_char(unsigned char val)**: retorna cantidad de bits 1 que hay en un byte que recibe como argumento.

- **update_bitmap()**: retorna el numero de bloques disponibles y actualiza bitmap.
- **int_to_bytes(unsigned char index[3], int block_number)**: transforma un int en un byte.
- **int_from_byte(unsigned char byte)**: retorna la posicion cero donde hay un byte.
- **update_byte(unsigned char byte, int pos_zero)**: suma un 1 donde esta la posicion zero ingresada en el byte.
- **update_remove_bitmap(int index)**: recibe un numero del bloque y actualiza el bitmap sacando ese número para que este disponible.
- **obtain_new_buffer(unsigned char byte, int pos_zero)**: recibe un byte y donde está la posición zero se actualiza con un 0 en el byte.
- **rm_recursive(int mem_dir)**: elimina un archivo de forma recursiva.
- **rm_file_mem_dir(int mem_dir)**: va a la direccion puntero del archivo y elimina todo lo que esta dentro.
- **os_desmontar()**: cierra el archivo de la faviable global file.
- **ret_pos(char* path)**: retorna la posición en bytes en donde se ubica la ruta que recibe.

Funciones de biblioteca

- **os_mount(char* diskname)**: función para montar el disco. Establece como variable global la ruta local donde se encuentra el archivo .bin correspondiente al disco. La variable global es de tipo FILE, lleva de nombre file y es un puntero. Está inicializada en el archivo os_API.h. Abre el archivo usando la función fopen en modo lectura y escritura de bytes.
- **os_bitmap(unsigned num, bool hex)**: la función imprime el estado actual del bloque de bitmap correspondiente a num entre 1 y 64, ya sea en binario (si hex es false) o en hexadecimal (si hex es true). Si num = 0 se debe imprimir el bitmap completo, imprimiendo ademñas una línea con la cantidad de bloques ocupados y una segunda línea con la cantidad de bloques libres. Para que funcione se crea primero una variable buffer (unsigned char*) y se pide espacio en memoria. Luego el código se separa en dos secciones: si num = 0 o cualquier otro caso. En el caso de que num = 0 se inicia un ciclo para recorrer cada uno de los bloques usando las funciones fseek (para mover el lector del archivo a la posicion number indicada) y fread (para leer un bloque entero en el buffer). Si num es mayor a 0 y menor a 65, se llega a la sección querida usando fseek nuevamente, y luego se lee con el uso de fread y el buffer.
- **os_exists**: función que revisa si un archivo existe. Si efectivamente existe retorna 1, y retorna 0 en caso contrario. La función primero verifica si es que el path en su posición 0 es nulo. Si es así, retorna 1, ya que se refiere al directorio. En caso contrario, empieza un ciclo que recorrerá los bloques hasta encontrar si el archivo existe o no. Se guardan los 32 bytes correspondientes a cada subloque del bloque de direccionamiento (3 bytes a indice y bloque y 29 bytes del nombre de la carpeta/archivo). Se chequea que el indice sea válido con la función is_valid(). Si es valido, y hay un slash en la ruta, debo acceder a un nivel más de profundidad de carpeta y se usa la función strip_path() y obtengo el nuevo path, luego verifico si en la nueva ruta está el nombre de lo que estoy buscando. Si no es así, modifica el numero del disco, cambia de lugar el lector y hace una llamada recursiva. En caso de que no haya un slash significa que no debo acceder a más niveles y sólo queda chequear ese nivel.
- **os_ls**: función que recibe un path e imprime en pantalla los nombres de todos los archivos contenidos en la carpeta indicada por el path. Para hacerlo inicia un ciclo de 0 a 63 en los que se guardan los 32 bytes correspondientes a cada subloque del bloque de direccionamiento (3 bytes a indice y bloque y 29 bytes del nombre de la carpeta/archivo), se verifica con la función is_valid que los dos bits de index sean mayor a cero, se obtiene la cantidad de caracterer que tiene path, y si su largo es 1 y solo contiene un slash, o es nulo, imprime todo el contenido de la carpeta directorio. En caso contrario, usa la funcion strip_path para sacar los paréntesis, y si en el nuevo path esta el nombre de lo que estoy buscandi, cambia el numero del disco, reposiciona el lector de disco y hace una llamada recursiva. Finalmente, imprime una linea de simbolos ïgualz reposiciona el lector del disco al inicio.

- **os_open**: -
- **os_write**: -
- **os_read**: -
- **os_mkdir**: Funcion que crea una carpeta en el directorio indicado. En primer lugar, revisa si el directorio indicado no hay una carpeta con el mismo nombre. Para eso se utiliza la funcion os_exists. Luego, revisa que el directorio justamente anreior, es decir, sin la carpeta a crear, efectivamente existe. Al validar ambos items, llama a la funcion recursiva os_mkdir_recursive y va, va a ejecutar n veces, siendo n-1 la cantidad de carpetas contenidas en el directorio. Cada ejecucion revisa la carpeta y ve si hay un elemento que corresponde al elemento siguiente de la recursion, y revisa que dicho elemento no sea un archivo. Ahi se llamara nuevamente la funcion. Cuando se llega a la carpeta n-1, al revisar los elementos de dicha carpeta revisara tambien que exista un espacio disponible para escribir. Si no existe, y si ya existe la carpeta a escribir, o si la direccion anterior al directorio no existe imprimira dicha razon respectivamente, retornando 0. Si cumple con todas las restricciones anteriores retornara 1.
- **os_rmdir**: Funcion que se encarga de eliminar la carpeta enrutada en el path. En primer lugar la funcion revisa que dicha direccion exista. Luego, revisa que el ultimo elemento del path sea efectivamente una carpeta, y no un archivo. Si es un archivo, imrpime el error correspondiente, y sin cambios en el .bin. Si se cumplen ambas condiciones, se procede a eliminar esa carpeta. Si recursive es false, en primer lugar revisa que cada entrada en el bloque de directorio de dicha carpeta este vacia. Sino retornara un print con el error correspondiente. Si recursive es true, para cada elemento dentro de la carpeta se elimiara de forma recursiva (en especial si hay carpetas dentro), y solo recien se procede a eliminar la carpeta original.
- **os_rm**: Funcion encargada de eliminar el archivo enrutada en path. Primero revisa que la ruta indicada existe, y si el ultimo elemento de la ruta es un archivo o no. Al corroborar estas dos restricciones, formatea de la carpeta los 32 bytes correspondientes al indice y nombre del archivo correspondiente. Ademas, llama a la funcion rm_file_mem_dir para liberar los bloques correspondientes en memoria si no hay otros hardlinks asociados al archivo.