Funciones **filesystemv2.c**

**uint32\_t abrir\_archivo(char \*nombreArchivo)**

* Abre el archivo cuyo nombre (sin extensión) se pasa como parámetro. Si el archivo .fcb existe la función retorna el tamaño del archivo. Caso contrario indica que el archivo no existe

**int crear\_archivo(char \*nombreArchivo)**

* Crea el archivo fcb, añadiendo al final de **nombreArchivo** la extensión **fcb**. Guarda en el archivo los valores iniciales requeridos.
* Retorna (1) debido (y como se indica en el enunciado) a que siempre es posible crear un archivo.
  1. **NOMBRE\_DE\_ARCHIVO=archivo (nombre que tendrá el archivo a crear)**
  2. **TAMANIO\_ARCHIVO=0 (Inicialmente tiene valor 0 asignado porque se desconoce su contenido)**
  3. **BLOQUE\_INICIAL=9999 (No definido, debido a que inicialmente no está en la tabla FAT)**

**int truncar\_archivo(char \*nombreArchivo,uint32\_t ui32\_longMen\_datos)**

* 1. **Caso A: (Ampliación del tamaño de archivo) –**
     1. Recibe como parámetros:
        1. El nombre del archivo (sin extensión)
        2. El tamaño o longitud en bytes de los datos en memoria que se deben almacenar
     2. Deberá acceder al **archivo.fcb** y evaluar si el archivo ya tienen una entrada FAT **inicial definida o no**
     3. Caso 2: **no tiene una entrada inicial definida (En este caso el archivo es nuevo)**
        1. Calcula la cantidad de bloques que necesita en FAT
        2. Calcula la cantidad de bloques FAT que se encuentran disponibles
        3. Evalúa si la cantidad de bloque que están disponibles son suficientes
           1. **Si hay bloques suficientes**, busca en la tabla FAT bloques disponibles creando la lista enlazada.

Actualiza el archivo archivo.fcb con el **tamaño del archivo** y el **bloque inicial**

**Finaliza operación**

* + - * 1. **En caso de no haber bloques suficientes** informa que no es posible crear el archivo por falta de espacio en disco.

**int leer\_archivo(char \*nombreArchivo)**

* Recibe como parámetro el nombre del archivo (sin extensión) a leer los bloques que componen sus datos desde el archivo bloques.dat.
* Implementa un bucle que a partir de la “entrada inicial” a la tabla FAT que obtiene del archivo .fcb correspondiente, y un algoritmo que realiza el recorrido por la lista enlazada que define la ubicación de los bloques en el archivo bloques.dat, concatena los bloques en una cadena que posteriormente imprime por pantalla el contenido
* Retorna (1)

**int escribir\_archivo(char \*nombreArchivo,char \*documentoArchivo)**

* Recibe como parámetros:
  + El nombre del archivo (sin extensión) que servirá para abrir el archivo .fcb del mismo y obtener los datos como: tamaño de archivo y entrada\_inicial de la tabla FAT
  + (temporal) – Cadena con el texto del documento leído desde archivo

1. Obtiene del archivo .fcb del **nombreArchivo** el tamaño del archivo y la entrada inicial a la tabla FAT
2. Solicita a memoria el envío de “buffer” de datos para almacenar del mismo tamaño que TAM\_BLOQUE que se encuentre establecido.

Funciones de la librería **util\_filesystem.h**

**int reiniciar\_fat(FILE \*fat,uint32\_t MAX\_ENTRADAS\_FAT**

* Recibe puntero de archivo a archivo de la tabla FAT y la cantidad de entradas máximas (bloques) de la tabla FAT.
* Operación: Borra y reiniciar el contenido del archivo tabla FAT. Su uso es principalmente destinado en realización de pruebas funcionales.
* Retorna siempre (1)

**uint32\_t tamanio\_Archivo\_fcb(char \*nombreArchivo)**

* Recibe el nombre del archivo (sin extensión alguna)
* Retorna el tamaño en bytes del archivo

**uint32\_t bloqueInicial\_Archivo\_fcb(char \*nombreArchivo)**

* Recibe el nombre del archivo (sin extensión alguna)
* Retorna la dirección de la entrada inicial asignada en la tabla FAT

**uint32\_t cantBloques\_FAT\_necesitados(uint32\_t ui32\_longMen\_datos,uint32\_t tamanioBloque)**

* Recibe la cantidad o longitud de bytes en memoria que se deben almacenar en un archivo y el tamaño del bloque de almacenamiento.
* Retorna la cantidad de bloques que se requieren para almacenar los datos de memoria.

**uint32\_t cantidadBloques\_FAT\_libres(FILE \*fat, uint32\_t MAX\_ENTRADAS\_FAT)**

* Recibe el puntero al archivo de la tabla FAT y la cantidad de entradas totales en la tabla FAT
* Retorna la cantidad entradas libres que están disponibles

**uint32\_t buscarBloqueFatDisponible(FILE \*fat,uint32\_t MAX\_ENTRADAS\_FAT)**

* Recibe puntero de archivo a archivo de la tabla FAT y la cantidad de entradas máximas (bloques) de la tabla FAT.
* Retorna la entrada en la tabla FAT del primer bloque libre que encuentra

**int asignarBloquesFAT(FILE \*fat,uint32\_t cant\_entradas\_FAT\_requeridas,uint32\_t MAX\_ENTRADAS\_FAT)**

* Recibe la cantidad de entradas que necesita de la tabla FAT para almacenar los datos del archivo y la cantidad de entradas máximas que tiene disponible la FAT.
* Retorna la entrada en la tabla FAT de la dirección inicial de la lista enlazada

**int actualizar\_Archivo\_fcb(char \*nombreArchivo,uint32\_t ui32\_longMen\_datos,uint32\_t ui32\_entrada\_inicial)**

* Recibe el nombre del archivo (sin extensión alguna), la cantidad de bytes del archivo y la entrada inicial en la tabla FAT
* Retorna (1)

**uint32\_t siguiente\_entrada\_tabla\_FAT(FILE \*fat,uint32\_t ui32\_entrada\_FAT)**

* Recibe el puntero al archivo de la tabla FAT y una entrada base desde la cual se comenzará la exploración
* Retorna la dirección a la siguiente entrada, de acuerdo a la lista enlazada.

**void mostrar\_tabla\_FAT(FILE \*fat, uint32\_t MAX\_ENTRADAS\_FAT**)

* Recibe puntero de archivo a archivo de la tabla FAT y la cantidad de entradas máximas (bloques) de la tabla FAT.
* Retorna la entrada en la tabla FAT del primer bloque libre que encuentra

Funciones de la librería **util\_bloques.h**

**char \*cargar\_buffer\_escritura(char \*buffer\_escritura, uint32\_t ui32\_numero\_bloque, char \*documentoArchivo, uint32\_t TAM\_BLOQUE)**

* Dada una cadena de datos **documentoArchivo** y **ui32\_numero\_bloque** carga en el buffer **buffer\_escritura** los datos de la cadena **documentoArchivo** del bloque indexado por **ui32\_numero\_bloque**
* Recibe como parámetros:
  + Un buffer (memoria) en la que se retornará el contenido de datos a almacenar
  + El número de bloque de “tamaño bloque” de datos que se devolverá en el buffer
  + El buffer (con todos los datos) desde donde se obtiene el bloque de datos que se asigna al buffer
  + El tamaño del buffer
* Retorna el buffer con los datos – Se tiene en cuenta la fragmentación internar

**int escribir\_en\_archivo\_bloques(FILE \*filesystem,uint32\_t ui32\_entrada\_FAT,char \*buffer\_escritura,uint32\_t tam\_bloque)**

* Escribe en el archivo **bloques.dat** **UN** bloque de datos de **tam\_bloque** que proviene de la memoria
* Recibe como parámetros:
  + **filesystem**, puntero al archivo bloques.dat para que pueda direccionar los accesos.
  + **ui32\_entrada\_FAT**, entrada de la tabla FAT asociada con posición del bloques dentro del archivo bloques.dat. Téngase en cuenta que la dirección física de acceso al bloque se calcula como:

**ui32\_bloque\_escritura=(64\*tam\_bloque)+(ui32\_entrada\_FAT\*1024)**

* + **buffer\_escritura**, puntero a la cadena que contiene los datos que se almacenarán en el bloque direccionado en el archivo bloques.dat
  + **tam\_bloque**, tamaño del bloque que puede ser configurable
* Retorna siempre (1)

**char \*lectura\_de\_archivo\_bloques(FILE \*filesystem,uint32\_t ui32\_entrada\_FAT,char \*buffer\_lectura,uint32\_t tam\_bloque)**

* Lee desde el archivo **bloques.dat** **UN** bloque de datos de **tam\_bloque** para enviarlo a la memoria
* Recibe como parámetros:
  + **filesystem**, puntero al archivo bloques.dat para que pueda direccionar los accesos.
  + **ui32\_entrada\_FAT**, entrada de la tabla FAT asociada con posición del bloques dentro del archivo bloques.dat. Téngase en cuenta que la dirección física de acceso al bloque se calcula como:

**ui32\_bloque\_escritura=(64\*tam\_bloque)+(ui32\_entrada\_FAT\*1024)**

* + **buffer\_lectura**, puntero a la cadena que en donde se volcarán los datos que se leerán desde el bloque direccionado en el archivo bloques.dat
  + **tam\_bloque**, tamaño del bloque que puede ser configurable
* Retorna, puntero a **buffer\_lectura**