Documentación API grupo DFLL

Mediante la siguiente documentación se describirá el funcionamiento de las principales funciones disponibles en la API entregada y su manejo de errores.

1) void **cr_mount**(char* *diskname*): Función para montar el disco. Se le entrega la ruta local donde se encuentra el disco .bin a montar. En el funcionamiento de la API, la ruta en cuestión se entrega mediante un argumento en consola. Primero se hace malloc de un array de bloques cargados para poder saber más adelante en la API los bloques que están cargados. Después se llama a una función auxiliar disco_init(), la cual en base al nombre entregado, abre y carga los bloques de directorio y de bitmap de cada partición en el disco y los añade a un array de bloques que posee el disco. Además esta función retorna el disco, cosa que se pueda ocupar el disco como variable más adelante en el programa. También dentro de la función disco_init, al inicizalizar cada bloque, se llama a una función bloque_init(), la cual a cada bloque le hace malloc de su array de bytes y array de bites y retorna el bloque. Así se genera un disco, que tiene un array de bloques, y cada bloque tiene un array de bites y un array de bytes.

Supuesto:Los discos a montar disponibles (los que estan en la carpeta discos) son llamados: “simdiskformat.bin” o “simdiskfilled.bin”.

Se imprime error cuando el nombre del disco a abrir ingresado en consola no esta en la carpeta discos. En dicho caso el programa termina.
De igual modo, de modo mas generico, si el archivo no existe disco_init genera error y termina el programa.

2) void **cr_bitmap**(unsigned *disk*, bool *hex*): Función para imprimir el bitmap en el formato entregado por el argumento *hex*, correspondiente a la partición entregada por el argumento *disk* $\in \{1,2,3,4\}$. Además imprime la cantidad de bloques ocupados y la cantidad de bloques libres en base a la partición entregada. El estado del bitmap se imprime en binario sí *hex* es *false* y en hexadecimal en caso de que binario sea *true*. En caso de que *disk* sea 0, se imprimirá el bitmap completo. El funcionamiento de la función corresponde a imprimir todos los bits del bloque correspondiente al bitmap (bloque directorio + 1) de la partición correspondiente en el formato correspondiente (hexadecimal o binario).

Se imprime error cuando el parametro disk ingresado no es un entero entre 0 y 1.

3) int **cr_exists**(unsigned *disk*, char* *filename*): Función para ver si el archivo filename existe en la partición disk. Retorna 1 en caso de que el archivo exista y 0 en caso de que no exista. La función funciona dirigiéndose a la partición correspondiente y guardando auxiliarmente el nombre de todos los archivos del bloque directorio, los cuales están entre el byte 4 y byte 32 de la secuencia correspondiente y comparándolos con el *filename* entregado. En caso de comparar 2 nombres iguales, la función retorna 1, en caso contrario retorna 0.

No se imprime error cuando el parametr disk ingresado no es un entero entre 1 y 4 porque quien usa cr_exist ya arroja ese error. Lo mismo ocurre con el nombre del archivo ingresado.

4) void **cr_ls**(unsigned disk): Función para imprimir todos los archivos de la partición disk. El funcionamiento viene dado por guardar en una variable auxiliar el nombre de los archivos, los cuales se encuentran entre el byte 4 y el byte 32 de la secuencia correspondiente en el bloque directorio. Al tener todos los bytes en el nombre auxiliar, imprime el nombre y continua con la siguiente secuencia del bloque directorio. Todo lo anterior en la partición correspondiente.

Se imprime error cuando el parametr disk ingresado no es un entero entre 1 y 4.

5) crFile* **cr_open**(unsigned *disk*, char* *filename*, char *mode*): Función para abrir el archivo filename de la partición disk en modo mode. En caso de que el *mode* sea “r”, se busca un archivo en la partición correspondiente y retorna el archivo representado por el *filename*. En caso de que el *mode* sea “w” se revisa que el archivo apuntado por el filename no exista en la ruta correspondiente al filename y retorna un nuevo crFile* que lo representa. El funcionamiento de la función viene dado por ir a la partición correspondiente y obtener la información del archivo correspondiente mediante la comparación de nombres en el bloque directorio. Al ya tener la información correspondiente revisa el bit de validez, para saber si el archivo ha sido eliminado o no. Después se obtiene la dirección del bloque índice mediante la información anterior y se cargan los bloques apuntados por el bloque índice en caso de que *mode* sea “r”. Por último genera una instancia del archivo con sus diferentes atributos obtenidos por la infomación del bloque directorio (tamaño, referencias, bloque índice, además de otra información como la partición). En el caso de que *mode* sea “w”, se asigna el primer bloque índice disponible al archivo y a la vez se van asignadno los bloques de información, según hayan bloques libres. El criterio de asignación es asignar el primer bloque libre que se encuentre.

Se imprime error si el modo no es “r” o “w”.
Se imprime error si la particion a leer o escribir no es un entero entre 1 y 4.
Se imprime error si el nombre del archivo no existe y se trata leer.
En dichos 3 casos el programa termina.
Se imprime error si el archivo ya existe y se trata de escribir. En este caso el programa continua.

6) int **cr_read**(crFile* *file_desc*, void* *buffer*, int *nbytes*): Función que lee los siguientes *nbytes* desde el archivo entregado por *file_desc* y los guarda en la dirección *buffer*. Retorna la cantidad de bytes efectivamente leídos. Para el funcionamiento de esta función, la instancia del archivo, tiene una posición de lectura, para que el programa sepa donde partir leyendo. En base a lo anterior, se calcula la cantidad de bytes restantes que quedan por leer (el cual está determinado por el tamaño del archivo o *nbytes*). Para los bloques correspondientes a ser leídos se recorren y se revisan si fueron anteriormente cargados. En caso de no haber sido cargados se llama a una función auxiliar cargar_bloque la cual hace malloc de un array de 2^{13}correspondiente a todos los bytes del bloque y asigna el bloque al array de bloques correspondiente al disco.

7) int **cr_write**(crFile* *file_desc*, void* *buffer*, int *nbytes*): Función para escribir en el archivo *file_desc* los *nbytes* indicados por la dirección *buffer*. Retorna cantidad de bytes efectivamente escritos. La función va a la partición correspondiente y a medida que queden bloques disponibles (que quede memoria disponible), por cada bloque le asigna la información apuntada correspondiente por *buffer*, a la vez que se carga el bloque, se le asigna su array de bites y de bytes con la información correspondiente y el bloque en sí es asignado al array de bloques del disco.

Se imprime error cuando no queda memoria

8) int **cr_close**(crFile* *file_desc*): Cierra el archivo *file_desc*. Esta función respalda en el disco.bin la información del archivo *file_desc* antes de cerrarlo. Para lograr lo anterior por cada bloque cargado correspondiente al archivo, ocupa una función auxiliar respaldar_a_bin (o respaldar a bin_bits si son bits, la cual cambia bits a bytes y despues respalda), a la cual se le entrega un número de bloque y mediante esa información escribe en la posición correspondiente del disco.bin los bytes correspondientes. Los ecribe en la ruta del disco, la cual es una variable global.

9) int **cr_rm**(unsigned *disk*, char* *filename*): Elimina el archive indicado por *filename* de la partición indicada por *disk*. Esta función abre el archivo mediante la función cr_open, después encuentra la secuencia correspondiente al archivo en el bloque directorio, para cambiar el bit de validez de esa secuencia de directorio. Después le resta 1 a la cantidad de hardlinks del archivo y en caso de que la cantidad sea menor o igual a 0, cambia el bitmap de los bloques de datos a partir de punteros indirectos, bloques de datos a partir de punteros directos y del bloque índice.

10) int **cr_hardlinks**(unsigned *disk*, char* *orig, char* dest*): Crea un hardlink del archivo referenciado por *orig* de la partición *disk* en la nueva ruta *dest*. La función abre el archivo dado por *orig*. La función en primer lugar encuentra la partición correspondiente y obtiene la información de la secuencia correspondiente del bloque de directorio, mediante el parámetro *orig*. La función también se encarga de cambiar los hardlinks de las secuencias correspondientes y de la instancia abierta del archivo. Por último se escribe en la primera secuencia disponible del bloque directorio con la información del nombre dada por *dest*, el nuevo número de hardlinks el primer bit como 1, ya que es una entrada válida.

Se imprime error cuando la particion ingresada no es un entero entre 1 y 4.
Se imprime error cuando no queda espacio en directoria para agregar el hardlink.
Se imprime error cuando no se ha encontrado el archivo descrito por orig.

11) int **cr_softlinks**(unsigned *disk_orig,* unsigned *disk_dest, char* orig, char* dest)*: Crea un softlink del archivo referenciado por *orig* de la partición *disk_orig* en la referencia *dest* de la partición *disk_dest*. La función encuentra el bloque de diccionario donde tiene que escribir una nueva secuencia, mediante los parámetros de destino y hace lo mismo con los parámetros de origen para obtener la información a escribir. En el bloque de diccionario destino se escribe en la primera secuencia disponible, con el primer bit de

validez válido, con los siguientes 23 bits correspondientes al bloque apuntado por el archivo original y los 29 bytes restantes correspondientes a la ruta del archivo en base al softlink creado.

Se imprime error cuando la particion ingresada tanto en origen como destino no es un entero entre 1 y 4.
Se imprime error cuando no queda espacio en directoria para agregar el softlink
Se imprime error cuando no se ha encontrado el archivo descrito por orig.

12) int **cr_unload**(unsigned *disk*, char* *orig, char* dest*): Función que se encarga de copiar en la ruta *dest* del computador del usuario de la API DFLL el archivo referenciado por *orig* o la partición completa referenciada por *disk* en caso de que orig sea NULL. En caso de que el valor de *disk* sea 0, se copiará el disco completo en la ruta *dest*. La función en primer lugar encuentra la partición correspondiente y en caso de que *disk* sea 0, se llama a sí misma recursivamente, para hacer la función sobre todas las particiones. Después se hace llamado a una función auxiliar cr_read_unload, solo con los bloques de datos que no han sido leídos anteriormente. La función se preocupa de recorrer los bloques de datos apuntados por direccionamiento directo y direccionamiento indirecto y en cada bloque recorrido lo carga en caso de que no haya sido cargado, mediante la función cargar_bloque explicada anteriormente, para después escribir en la ruta *dest* la información de cada bloque.

Se imprime error cuando la particion ingresada no es un entero entre 0 y 4.
Se imprime error cuando el archivo infresado de origen no existe en el computador.

13) int **cr_load**(unsigned *disk*, char* *orig*): Función que copia todo lo posible de un archivo o los contenidos de una carpeta en la ruta referenciada por *orig* a la partición *disk*. En caso que la carpeta tenga subcarpetas, solo se copiarán los archivos de la carpeta original y no las subcarpetas y sus archivos correspondientes. La función revisa si el archivo existe, después crea un archivo, mediante la función cr_open en con *mode* “w”, para después ir leyendo el archivo *orig* y a la vez que se va leyendo, mediante un buffer escribe en el nuevo archivo creado mediante la función cr_write().

Se imprime error cuando el archivo ingresado de origen no existe en el computador.