INSTITUTO POLITÉCNICO NACIONAL.

ESCUELA SUPERIOR DE CÓMPUTO.

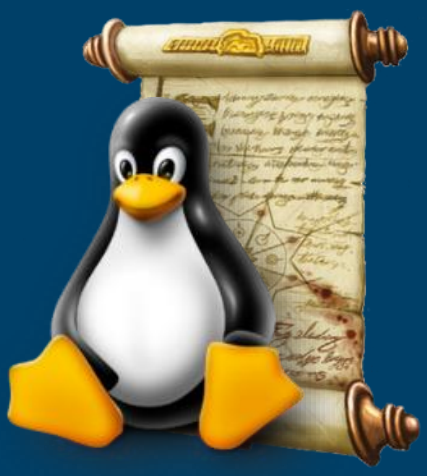
SISTEMAS OPERATIVOS.

PRÁCTICA 8

Llamadas al sistema para manejo de archivos y directorios.

Integrantes del equipo:

- Chavarría Vázquez Luis Enrique.
- Juárez Espinoza Ulises.
- Machorro Vences Ricardo Alberto.
- Pastrana Torres Víctor Norberto.



2CM6 **Unidad 5**

Índice de contenido.

Índice de figuras.

Índice de tablas

Glosario de términos

Directorio

Un directorio es una ubicación para almacenar archivos en una computadora. Es una estructura de catalogación del sistema de archivos que contiene referencias a otros archivos o directorios. Las carpetas y los archivos están organizados en una estructura jerárquica , lo que significa que están organizados de una manera que se asemeja a un árbol. Por ejemplo, un directorio contenido dentro de otro directorio se llama subdirectorio. Los términos padre e hijo se utilizan a menudo para referirse a directorios y subdirectorios, respectivamente. [1]

Rutas de directorio

Para acceder a un archivo dentro de un directorio, es posible que sea necesario especificar los nombres de todos los directorios anteriores. Para hacer esto, será necesario especificar una ruta . Una ruta especifica una ubicación única siguiendo la jerarquía del árbol de directorios expresada en una cadena de caracteres en la que los componentes de la ruta representan cada directorio. El carácter delimitador suele ser una barra diagonal o dos puntos. [2]

Tipos de directorios

Directorio de un solo nivel

Todos los archivos están contenidos en el mismo directorio, el cual es fácil de soportar y entender, sin embargo, cuando aumenta el número de archivos o cuando hay más de un usuario un directorio de un solo nivel tiene limitaciones considerables. Debido a que todos los archivos están en el mismo directorio deben tener nombres únicos. La principal desventaja de un directorio de un solo nivel es la confusión de los nombres de archivos creados por usuarios diferentes. La solución estándar consiste en crear un directorio distinto para cada usuario. [3]

Cabe destacar que en la siguiente ilustración se muestra de forma contundente un diagrama que ilustra a la perfección como son los directorios de un solo nivel de manera gráfica. (ilustración 1).

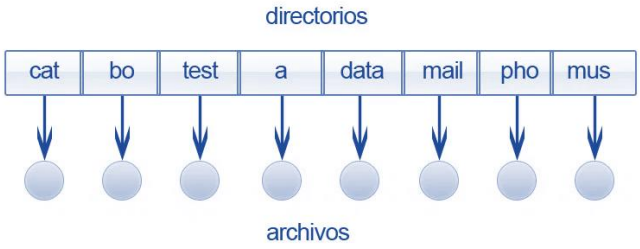


Ilustración 1directorio de un solo nivel.

Directorio de dos niveles

Cada usuario tiene su propio directorio de archivos de usuario (user file directory, UFD) cada UFD tiene una estructura similar, pero lista solo los archivos de un usuario. Cuando comienza un trabajo de usuario o se conecta un usuario, se hace una búsqueda en el directorio de archivos maestro (master file directory, MFD). El MFD esta indexado por el nombre de usuario o el número de cuenta y cada entrada apunta al UFD para dicho usuario. [4]

En el siguiente diagrama podemos apreciar del mismo modo como es que esta estructurado un directorio de dos niveles, con lo cual dentro de la ilustración 2 se puede ver de forma mucho más simple lo ya expuesto en el párrafo anterior.

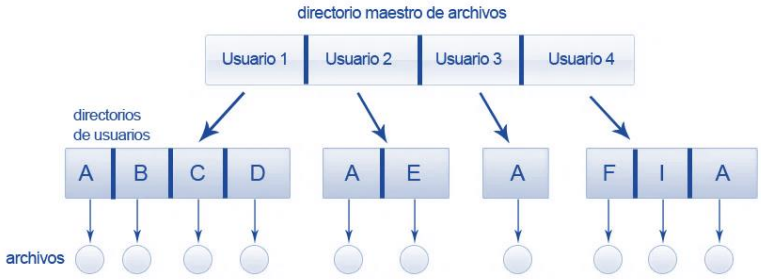


Ilustración 2 directorio de dos niveles.

Directorio con estructura de árbol

Se puede visualizar como un árbol de dos niveles, la generalización natural consiste en extender la estructura del directorio y un árbol de altura arbitraria. Esta generalización permite a los usuarios crear sus propios subdirectorios y organizar sus archivos con base en esto. El sistema MS-DOS, por ejemplo, está estructurado como un árbol. Con un sistema de directorios con estructura de árbol, los usuarios pueden tener acceso a los archivos de otros usuarios, además de sus propios archivos. [5]

Vale la pena destacar que en el siguiente diagrama (ilustración 3), podemos ver de forma satisfactoria como es que se integra un directorio con estructura de árbol.

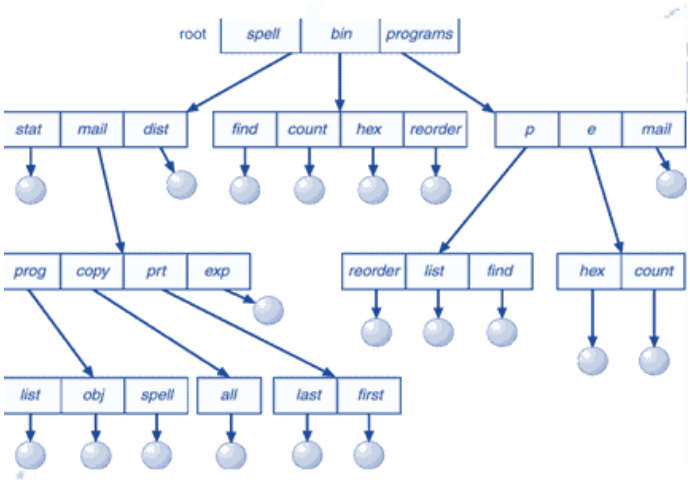


Ilustración 3 diagrama de un directorio de árbol.

Directorio con estructura acíclica

Una estructura de árbol prohíbe el compartimiento de archivos o directorios. Una gráfica acíclica (grafica sin ciclos) permite que los directorios tengan subdirectorios y archivos compartidos. El mismo archivo o subdirectorio puede estar en dos directorios diferentes. Una gráfica a cíclica es una generalización natural del esquema de directorios con estructura de árbol. Cuando varias personas están trabajando como equipo, todos los archivos que se van a compartir pueden colocarse juntos en un directorio. Cada uno de los directorios de archivos de usuario de todos los miembros del equipo

contiene este directorio de archivos compartidos como un subdirectorio. Una estructura de directorios de grafica acíclica es más flexible que una estructura sencilla de árbol, pero también es más compleja [6]. Si bien ya hemos explicado en que consiste este tipo de directorio también es necesario visualizar de forma concreta como es que este tipo de directorios son integrados de forma gráfica justo como se muestra en la ilustración de la parte inferior (Ilustración 4).

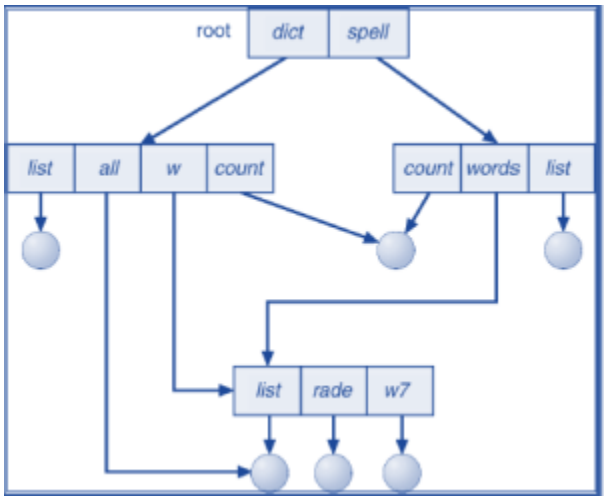


Ilustración 4 diagrama de un directorio con estructura acíclica.

Sistema de gestión de archivos

También denominado simplemente como sistema de archivos o sistema de archivos. Tenemos que el sistema que utiliza un sistema operativo o programa para organizar y realizar un seguimiento de los archivos. Por ejemplo, un sistema de archivos jerárquico es uno que usa directorios para organizar archivos en una estructura de árbol. [7]

Fichero

Un fichero es tradicionalmente un lugar donde se guardan fichas o papeles. En el caso de la informática, nos referimos a un fichero o archivo como un lugar (naturalmente virtual) donde se guarda información. [8]

Llamada al sistema

Una llamada al sistema o system call es un método utilizado por los programas de aplicación para comunicarse con el núcleo del sistema. En los sistemas operativos modernos, esto es necesario cuando una aplicación o proceso de usuario necesita transmitir a o leer información del hardware, de otros procesos o del propio núcleo [9]. De este modo, la llamada es el punto de enlace entre el modo de usuario y el modo de núcleo, los dos modos cruciales de privilegio y seguridad para el procesamiento de las instrucciones de la CPU en los sistemas informáticos.

Contenido (Investigación)

Aspectos fundamentales.

Las llamadas al sistema están estrechamente ligadas al modelo moderno de sistema operativo con modo usuario y modo núcleo, que se introdujo en respuesta al creciente número de procesos que se ejecutan simultáneamente en la memoria principal (RAM) de los ordenadores. De este modo, cada proceso individual tiene sus propios datos con derechos de acceso especiales y, solo si se distribuyen los recursos correctamente, el sistema y los programas pueden ejecutarse según lo esperado. [10]

El modo núcleo de mayor privilegio es aquí la instancia de control decisiva, porque, como mencionamos, en este no solo se ejecutan todos los servicios y procesos del sistema en sí, sino también las acciones críticas del sistema de los programas de aplicación que en el modo usuario están bloqueadas. Para esto, es necesario que el programa realice una llamada al sistema, que en la mayoría de los casos es simplemente una cuestión de acceso a la potencia de cálculo (CPU) o a las estructuras de memoria (memoria de trabajo y espacio de disco duro).

Lista de funciones detallada.

access

La función access () debe verificar el archivo nombrado por el nombre de ruta apuntado por el argumento de ruta para la accesibilidad de acuerdo con el patrón de bits contenido en el modo, utilizando el ID de usuario real en lugar del ID de usuario efectivo y el ID de grupo real en lugar del ID de grupo efectivo. Si se verifica cualquier permiso de acceso, cada uno se debe verificar individualmente. [11] Si el proceso tiene los privilegios adecuados, una implementación puede indicar éxito para X_OK incluso si no se establece ninguno de los bits de permiso de ejecución del archivo.

En la siguiente imagen podemos apreciar la sintaxis que access tiene, con lo cual nos queda mucho más claro como es que podemos trabajar con la misma, lo descrito como se mencionó puede ser apreciado en la ilustración 5.

```
#include <unistd.h>

int access(const char *path, int amode);
```

Ilustración 5 sintaxis para Access.

chdir

El comando chdir es una función del sistema (llamada al sistema) que se utiliza para cambiar el directorio de trabajo actual. [12] En algunos sistemas, este comando se usa como un alias para el comando de consola cd . chdir cambia el directorio de trabajo actual del proceso de llamada al directorio especificado en la ruta. Ahora bien, lo ya descrito es complementado con lo que se puede ver en la ilustración número 6 (ilustración 6) en donde se ve la sintaxis base para poder aplicarlo en nuestro código.

```
int chdir (const char * ruta);
```

Ilustración 6 sintaxis para chdir.

chmod

Cambia el modo del archivo o directorio especificado en ruta. El argumento de modo se crea con uno de los siguientes símbolos definidos en el archivo de encabezado sys / stat.h. [9] Si bien lo descrito anteriormente muestra de forma cabal el funcionamiento básico creemos que también es pertinente poder mostrar dentro de nuestro trabajo la sintaxis por medio de la ilustración 7.

```
int chmod(const char *pathname, mode_t mode);
```

Ilustración 7 sintaxis básica para chmod.

chsize

La función chsize() cambia el tamaño del archivo asociado con los campos del descriptor de archivo para que tenga exactamente el tamaño de bytes de longitud. La función trunca el archivo o lo rellena con un número apropiado de bytes [13]. Si chsize() falla, por ejemplo, porque no hay suficiente espacio, el tamaño del archivo permanece sin cambios. Una vez que tenemos lo anterior en mente podemos apreciar en la imagen de la parte inferior (Ilustración 8) como es que la funcionalidad debe ser utilizada dentro de la sintaxis del lenguaje de programación de manera general.

```
int chsize (int fildes , long size );
```

Ilustración 8 sintaxis básica para chsize.

Se requiere aclarar que el atributo de fildes es en términos generales el descriptor de archivo de un archivo de disco abierto con acceso de escritura. La parte de size es directamente la nueva longitud del archivo, en bytes. Si el tamaño es menor que el tamaño inicial del archivo, se liberan todos los bloques de disco asignados entre el tamaño y el tamaño del archivo inicial. Si el tamaño es negativo, el archivo se trunca a una longitud cero.

closedir

La función closedir () cerrará el flujo de directorio al que se refiere el argumento dirp . Al regresar, el valor de dirp ya no puede apuntar a un objeto accesible del tipo DIR . Si se utiliza un descriptor de archivo para implementar el tipo DIR , ese descriptor de archivo se cerrará. El valor que nos devolverá una vez completado con éxito, closedir () devolverá o retornará el valor de 0; de lo contrario, se devolverá -1 y errno se establecerá para indicar el error. [14] Ya una vez aclarado lo anterior es pertinente mostrar en código de forma concreta la sintaxis básica para poder emplear la funcionalidad anteriormente descrita, lo cual puede ser observado en la ilustración número 9 de la parte inferior.

```
#include <dirent.h>
int closedir(DIR *dirp);
```

Ilustración 9 sintaxis básica para closedir.

_creat

La llamada a la función: es equivalente a la llamada: creat(pathname,mode). Por lo tanto, se crea el archivo nombrado por ruta , a menos que ya exista. A continuación, el archivo se abre solo para escritura y se trunca a una longitud cero. Consulte open (): abre un archivo para obtener más información [15]. El argumento de mode especifica los bits de permiso de archivo que se utilizarán para crear el archivo, con lo cual ya que hemos explicado lo anterior podemos ver de forma muy concreta la parte de la sintaxis para el uso de esta funcionalidad en nuestro sistema la cual evidentemente variará dependiendo de la aplicación que queramos darle a la funcionalidad misma dentro de nuestros proyectos, con lo cual lo dicho puede verse en la parte de abajo en la ilustración número 10.

```
open(pathname, O_CREAT|O_WRONLY|O_TRUNC, mode);
```

Ilustración 10 sintaxis básica para _creat.

createttemp

Esta función genera un nombre de archivo único y crea un archivo con ese nombre. Creattemp normalmente crea el archivo temporal en el directorio / tmp en máquinas UNIX y en \ cbase \ tmp en máquinas MS-DOS. Si se define la variable de entorno TMPDIR, esto nombra el directorio para el archivo temporal. El nombre del archivo consta de un prefijo opcional y un sufijo único generado internamente [16]. El prefijo puede ser NULL o apuntar a una cadena terminada en nulo de hasta dos caracteres para asegurar que los nombres generados sean válidos en máquinas MS-DOS, siendo lo anteriormente descrito una descripción fundamental del funcionamiento de la funcionalidad y que puede verse en su forma de ser aplicada en la imagen de la parte inferior a este párrafo en la ilustración 11.

```
creattemp (prefix, result, pmode, oflags) char *prefix; char *result; int pmode; int oflags;
```

Ilustración 11 sintaxis básica para createttemp.

dup y dup2

Las llamadas al sistema dup proporcionan un modo para duplicar un descriptor de archivos, presentando dos o más descriptores diferentes que acceden al mismo archivo. Se pueden usar para leer y escribir en diferentes partes del archivo [17]. La llamada al sistema dup duplica un descriptor de archivo, fildes, enviando un nuevo descriptor. La llamada al sistema dup2 copia eficazmente un descriptor de archivo a otro especificando qué descriptor usar para la copia, lo ya dicho puede reafirmarse con la siguiente ilustración en la cual podemos ver la sintaxis básica para poder utilizar la funcionalidad (ilustración 12).

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
```

Ilustración 12 sintaxis básica para dup y dup2.

eof y feof

EOF (End Of File)es un parámetro booleano útil para facilitar el cierre de bucles de extracción de datos desde archivo. En C,EOF es una constante de tipo entero (normalmente -1) que es el retorno

que envían distintas funciones de extracción de información desde archivos al llegar a un final de archivo y no existir más datos. También se puede "simular" EOF mediante una entrada de teclado, normalmente CTRL+Z y enter ó CTRL+D y enter según el sistema operativo que empleemos. Por parte de feof el valor que tendremos de retorno será cuando la creación de la corriente asociada con el final del identificador de archivo, la función devuelve un valor distinto de cero, de lo contrario, devuelve cero. [18] En las siguientes imágenes proponemos algunos ejemplos para poder entender mejor la sintaxis de cada uno de ellos.

A continuación, podemos ver como en la ilustración 13 y en la ilustración 14 la forma de trabajar con feof y en la parte inferior a la primera ilustración se aprecia un ejemplo de precisamente como utilizar eof de manera concreta, aunque de manera posterior nosotros hemos planteado un ejemplo con código para ver como es que se utilizan de forma práctica.

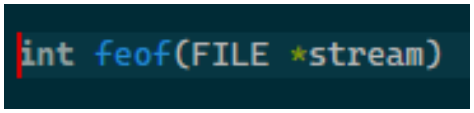


Ilustración 13 sintaxis básica para feof.

Aquí presentamos el ejemplo previamente mencionado en el párrafo anterior, pero recalcamos que en esta imagen solo se muestra un ejemplo mínimo en el que tenemos la aplicación de eof (ilustración 14), pero de manera posterior se mostrará un ejemplo propio un poco más complejo para entender el uso práctico.

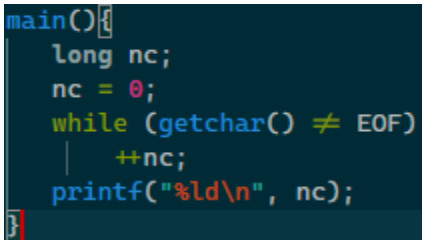


Ilustración 14 ejemplo de sintaxis para eof.

Si se pretende escribir un programa que lea dos cadenas de caracteres que pueden contener saltos de linea y otros caracteres especiales. Por lo tanto, podemos usar EOF (Ctrl-Z o Ctrl-D) para finalizar la cadena. Debemos tener en cuenta que después de recibir un EOF del terminal, no recibirá ningún dato adicional . Por lo tanto, debemos definir que cada variable se ingrese en una línea separada y que los usuarios presionen Intro en lugar de EOF. En la siguiente imagen podemos ver lo anteriormente descrito y sirve del mismo modo como un ejemplo claro de como aplicarlo. Cabe destacar que lo descrito anteriormente se puede ver en el ejemplo que nosotros hemos planteado en la ilustración 15.

```c
#include <stdio.h>
#include <string.h>

int main(void){
    int x = 0;
    int c;
    char a[100];
    char b[100];
    printf("Enter a: ");
    while((c = getchar()) != EOF){
        a[x] = c;
        x++;
    }
    a[x] = '\0';
    x = 0;

    printf("\nEnter b: ");
    while((c = getchar()) != EOF){
        b[x] = c;
        x++;
    }
    b[x] = '\0';
    printf("\n\nResults:\na: %s\n", a);
    printf("b: %s\n", b);
    return(0);
}
```

Ilustración 15 Ejemplo de aplicación con eof.

fclose

Cierra un fichero. Si hay datos en buffer pendientes de volcar, se vuelcan antes de cerrar. Un fichero abierto con "fopen" se cerrará también automáticamente al terminar el programa. Si se intenta cerra un fichero no abierto, se obtendrá una "violación de segmento" [19]. Ahora bien, en la imagen que aparece a continuación en la parte de abajo (ilustración 16) el parámetro que precisamente fclose debe recibir para el correcto empleo del mismo.

```c
fclose(fichero);
```

Ilustración 16 sintaxis para fclose.

_fdopen

Asocia un flujo a un descriptor de archivo obtenido previamente. El descriptor de archivo se ha obtenido mediante una llamada a open() , dup() , create() o pipe() . Estas funciones abren el archivo, pero no proporcionan un puntero de FILE . El modo del descriptor tiene que coincidir con el modo del archivo basado acorde con lo que se halló en la fuente [20]. En la ilustración número 17 podemos ver los parametros que recibe fdopen.

```c
FILE * fdopen(int descriptor, char *modo)
```

Ilustración 17 sintaxis para fdopen.

ferror

Esta función examina el indicador de error del flujo proporcionado, y devuelve un valor verdadero si este indicador está activado. El indicador de error sólo se puede poner a cero mediante clearerr(), todo acorde con lo hallado en [21]. Si bien ya que hemos aclarado el funcionamiento general presentamos en la ilustración 18 los parametros que recibe ferror.

```
int ferror(FILE *flujo)
```

Ilustración 18 sintaxis para ferror.

fflush

Esta función desencadena la escritura del contenido del búfer asociado al flujo. No se modifica el estado de apertura. Si se proporciona un flujo NULL, se fuerza el volcado del búfer de todos los flujos que estén abiertos en modo de salida basado en lo encontrado en [22]. Con lo anterior en mente exponemos en la siguiente imagen de la parte inferior (ilustración 19) los parametros que recibe fflush.

```
int fflush(FILE *flujo)
```

Ilustración 19 sintaxis para fflush.

getc

Esto es una macro equivalente a fgetc(), que lee del flujo proporcionado un carácter, si existe; basado en la información hallada en la fuente de [23]. Ahora bien, es conveniente mostrar los parámetros que deberemos considerar mientras utilizamos esta funcionalidad dentro de nuestra implementaciones de código, lo cual puede ser apreciado de manera clara en la ilustración que aparece a continuación la cual esta numerada como ilustración 20.

```
int getc(FILE *flujo)
```

Ilustración 20 sintaxis para getc.

fgetpos

Esta función equivale a ftell() . La función fgetpos() almacena en la variable señalada por el valor del contador de archivo en ese momento, esto es, indica en qué posición del archivo se va a leer o escribir el próximo carácter acorde con lo que se pudo encontrar en la fuente [24]. Posterior a esta explicación podemos apreciar como en la ilustración 21.

```
int fgetpos(FILE *flujo, fpos_t *pos)
```

Ilustración 21 sintaxis para fgetpos.

fgets

Esta función permite leer una línea del teclado, con la limitación de leer un máximo de tamaño-1 caracteres. Se añade el carácter '\0' al final de la cadena. Esta es, posiblemente, la función más adecuada para leer información de teclado, por cuanto permite garantizar la cantidad de información

leída. Si el número de bytes leídos es menor que el espacio reservado en la cadena de destino, no podrá producirse un error por desbordamiento de ésta, todo lo mencionado puede apreciarse en la fuente [25]. Ahora que ya está claro podemos ver en la ilustración 22 los parametros que fgets recibe y parte de la sintaxis que se debe seguir.

```
char * fgets(char *cad, int tamap, FILE *flujo);
```

Ilustración 22 sintaxis para fgets.

findclose

La función FindClose cierra el identificador de búsqueda especificado. Las funciones FindFirstFile y FindNextFile utilizan el identificador de búsqueda para localizar archivos con nombres que coinciden con un nombre dado. En la siguiente imagen podemos ver como utilizamos el findclose para usarlo como cierre de las funciones con las que contamos, siendo la información presentada fundamentada en su totalidad en lo encontrado en la fuente [26].

En el siguiente ejemplo que presentamos, se muestra como después de usar algun tipo de funcionalidad que requiera emplear el prefijo de find, podemos hacer una implementación de cierre con findclose, la cual nos permitirá hacer un cierre efectivo de la búsqueda y con ello no solo ser más eficiente en la tarea sino que del mismo modo podremos cumplir de manera cabal con el propósito que este tiene, lo cual se puede apreciar con claridad en la ilustración 23.

```
int FindClose_ex1(){
    string strPath = "c:\\*.*";
    FindClose_dir(strPath);
    return 1;
}
void FindClose_dir(string strFile){
    WIN32_FIND_DATAA    find;
    HANDLE    hFile;

    int    nCount = 0;

    if((hFile=FindFirstFile(strFile,&find)) ≠ INVALID_HANDLE_VALUE){
        out_str(find.cFileName);
        nCount++;
        while(FindNextFile(hFile,&find)){
            out_str(find.cFileName);
            nCount++;
        }
        printf("%d files found\n",nCount);
        FindClose(hFile);        // cerramos
    }
    else
        printf("%s: file(s) not found\n",strFile);
}
```

Ilustración 23 sintaxis para findclose y ejemplo de utilización.

findfirst y findnext

Comienza la búsqueda de ficheros definidos por atributos o comodines en un directorio de disco, del mismo modo findnext continua con la operación de findfirst. Si tiene éxito, _findfirst devuelve un identificador de búsqueda exclusivo que identifica el archivo o grupo de archivos que coinciden con la especificación de archivo , que se puede utilizar en una llamada posterior a _findnext o _findclose acorde con lo indagado en la fuente [27]. Ya que hemos entendido para que sirven, podemos ver de forma mucho más clara como es que estas reciben diversos parametros que permiten su correcto

funcionamiento, lo cual evidentemente se puede apreciar en la ilustración de la parte inferior numerada con el identificador de ilustración 24.

```
intptr_t _findfirst(
    const char *filespec,
    struct _finddata_t *fileinfo
);

int _findnext(
    intptr_t handle,
    struct _finddata_t *fileinfo
);
```

Ilustración 24 sintaxis para findfirst y findnext simplificada.

_fileno

Obtiene el descriptor de archivo asociado con una secuencia también cabe aclarar que _fileno devuelve el descriptor del archivo. No hay retorno de error. El resultado no está definido si la secuencia no especifica un archivo abierto. Si la secuencia es NULL, _fileno invoca el controlador de parámetros no válido, como se describe en validación de parámetros, lo mencionado puede ser hallado en [28]. Habiendo considerado esta introducción a _fileno, podemos apreciar los parametros que deberá recibir la funcionalidad del lenguaje de manera clara en la siguiente imagen (ilustración 25).

```
int _fileno(
    FILE *stream
);
```

Ilustración 25 sintaxis para _fileno.

fnsplit

Fnsplit divide un nombre de ruta completo en sus componentes, fnsplit toma el nombre de la ruta completa de un archivo (ruta) como una cadena en la forma X: \ DIR \ SUBDIR \ NAME.EXT y divide la ruta en sus cuatro componentes. Luego almacena esos componentes en las cadenas a las que apunta la unidad, directorio, nombre y ext. Se deben pasar los cinco componentes, pero cualquiera de ellos puede ser nulo, lo que significa que el componente correspondiente se analizará pero no se almacenará. Si algún componente de la ruta es nulo, ese componente corresponde a una cadena vacía no NULL, con lo ya dicho podemos remontarnos a la fuente encontrada en [29].

En la siguiente imagen podemos ver los parametros y un poco más de la sintaxis para la implementación de fnsplit, toda la información descrita es aplicable al ejemplo que tenemos en la parte inferior en la ilustración 26.

```
int fnsplit(const char *path, char *drive, char *dir, char *name, char *ext);
```

Ilustración 26 sintaxis para fnsplit.

fopen

Esta función abre el archivo cuyo nombre (y, opcionalmente, ruta) señala ruta , y le asocia un flujo. El parámetro de modo indica si se desea abrir el archivo para leer, escribir o leer y escribir; también se puede indicar nuestra intención de añadir información al archivo, así como otras combinaciones de opciones lo mencionado puede encontrase en la cita siguiente [30]. En la imagen que procede a esta explicación podemos apreciar de forma cabal como es que fopen recibe múltiples parametros los cuales nos serán de utilidad para la implementación de esta funcionalidad justo como se puede apreciar en la siguiente imagen con el identificador de ilustración 27.

```
FILE * fopen(char *path, char *mode)
```

Ilustración 27 sintaxis para fopen.

fprintf

Esta función permite escribir en un archivo de texto, efectuando las conversiones a formato alfanumérico que se le indiquen, lo mencionado esta fundamentado en lo hallado en la cita siguiente [31]. Una vez aclarado el punto anterior se puede destacar del mismo modo que tenemos en la siguiente imagen un poco de la sintaxis base para la recepción de los parametros con lo que esto queda más que claro en la siguiente ilustración número 28 de la parte inferior.

```
int fprintf(FILE *flujo, const char *format, ...)
```

Ilustración 28 sintaxis para fprintf.

fputc

La macro putc() es equivalente a esta función, aunque puede evaluar varias veces el flujo proporcionado; por tanto, para indicar el flujo no deben emplearse expresiones con efectos secundarios. Finalmente, si se emplea fputc() con stdin como destino, el resultado es equivalente a una llamada a putchar(), se aclara que ya mencionado fue encontrado en la cita que aparece a continuación [32]. En la parte posterior es digno de destacar que podemos hallar una imagen con los parametros que fputc recibe, con lo cual podemos darnos una mejor idea de cómo es que su implementación debe realizarse de manera satisfactoria con lo cual puede resultar mucho más fácil visualizarlo en términos de código al momento de implementarlo en alguna de nuestras soluciones de código, lo ya dicho puede apreciarse con claridad en la ilustración 29.

```
int fputc(int c, FILE *flujo)
```

Ilustración 29 sintaxis para fputc.

fputs

Envía al flujo indicado la cadena cuyo puntero se da como primer argumento. Se añade un signo de nueva linea al final de la cadena [33]. La explicación anterior en definitiva no quedaría completa sin mostrar que parametros recibe debido a que el uso de los mismos es de hecho de suma importancia para su implementación satisfactoria dentro de nuestro codigo con lo cual debemos ser capaces de entender a la perfección como aplicarlo, por lo cual en la parte inferior disponemos de precisamente una imagen que nos ayuda a entender mejor como es que se incrusta en el codigo fputs y del mismo modo justo como se ha mencionado, los parametros que este recibe; lo dicho es apreciable en la ilustración número 30.

```
int fputs(const char *cad, FILE *flujo)
```

Ilustración 30 sintaxis para fputs.

fread

Esta función lee del flujo señalado por flujo num_elementos bloques de tamaño bytes de longitud, y los escribe en memoria a partir de la dirección dada por ptro . Como resultado, la función proporciona el número de bloques leídos. Si se produce un error, el resultado es un número de bloques menor que el esperado o cero. El final del archivo se detecta mediante una llamada a feof(). Esta función, al no hacer cambio de formato alguno, es la que permite leer de disco con mayor rapidez, acorde con lo que ya hemos mencionado del mismo modo podemos encontrar lo anterior en la siguiente referencia [34]. Ya que tenemos lo anterior en mente en la parte inferior podemos apreciar de forma contundente una imagen (ilustración 31) que nos muestra los parametros que recibe.

```
size_t fread(void *ptro, size_t tamaño, size_t num_elementos, FILE *flujo)
```

Ilustración 31 sintaxis para fread.

freopen

Esta función sirve para reutilizar flujos creados anteriormente. Recibe como argumento el nombre (y, opcionalmente, la ruta) de un archivo, el modo de acceso (lectura o escritura) deseado para abrir el archivo, y el puntero de un flujo ya existente. El flujo, si existía anteriormente, se cierra, y se cambia el archivo asociado. Esta función suele utilizarse para cambiar el archivo asociado a los flujos de texto estándar (stdin , stdout y stderr), se puede encontrar lo dicho en la referencia que aparece a continuación [35]. En la parte inferior se aprecian los múltiples parametros que deben ser recibidos con en freopen con lo cual queda mucho más claro como es que este se implementa, esto mencionado puede verse con claridad en la ilustración número 32.

```
FILE * freopen(char *path, char *mode, FILE *flujo)
```

Ilustración 32 sintaxis para freopen.

fscanf

Esta función es análoga a scanf() , con una diferencia: en lugar de leer de stdin , lee del flujo proporcionado como primer argumento. Los demás parámetros tienen comportamiento idéntico al descrito en scanf(), lo que hemos mencionado puede ser hallado en la referencia siguiente [36]. En la siguiente imagen de manera muy concreta podemos ver como se encuentran los parametros que se reciben con el uso de esta funcionalidad (ilustración 33).

```
int fscanf(FILE *stream, const char *format, ...)
```

Ilustración 33 sintaxis para fscanf.

fseek

Esta función permite especificar un valor para el contador de archivo correspondiente al flujo que se proporciona como primer parámetro. El segundo argumento es el número de bites que se desea

desplazar el contador. El tercer argumento especifica el origen del desplazamiento especificado, y puede tomar los valores que se especifican a continuación.

- SEEK_SET Desde el comienzo del archivo
- SEEK_CUR Desde la posición actual del contador de archivo
- SEEK_END Desde el final del archivo

De esta manera, la función fseek() se utilizará para situar el puntero de archivo donde convenga para leer o escribir [37]. Ya aclarado lo anterior en la siguiente tabla podemos ver de manera un poco más clara como es que estas llamadas se llevan a cabo y como es que estas pueden ser empleadas con distintos propósitos en el desarrollo de nuestras soluciones de software, lo ya mencionado puede apreciarse en la tabla número 1.

Tabla 1 para la explicación de las múltiples llamadas para fseek.

Llamada a la acción.	Explicación de la llamada.
`fseek(fp, 0L, SEEK_SET);`	Situar el contador de archivo en el primer byte (equivale a rewind()).
`fseek(fp, 0L, SEEK_END);`	Situar el contador de archivo en el primer byte no utilizado, esto es, al final del archivo.
`fseek(fp, (indice_elemento - 1)*longitud_elemento, SEEK_SET);`	Si se utilizan archivos formados por elementos de un cierto tipo cuyo tamaño en bytes es longitud_elemento , entonces esta expresión permite situar el contador de archivo en el elemento i-ésimo del mismo.

Como puede observarse, fseek() es clave para el manejo de archivos de acceso directo. La función, al igual que todas las de este tipo, proporciona el valor 0 si todo va bien y el valor -1 si se produce algún error, lo dicho puede ser hallado en la siguiente referencia [38].

fsetpos

Equivale a fseek() con un valor de desde donde igual a SEEK_SET, en la siguiente imagen podemos apreciar la sintaxis y del mismo podemos apreciar lo dicho en la siguiente referencia [39]. Procedemos en la siguiente ilustración a poder mostrar los parametros que recibe la funcionalidad fsetpos y todo ellos dentro de la ilustración número 34.

```
int fsetpos(FILE *stream, const fpos_t *pos)
```

Ilustración 34 sintaxis para fsetpos.

fstat

La llamada al sistema fstat envía información de estado sobre el archivo asociado con un descriptor de archivo abierto. La información se escribe en una estructura, buf, cuya dirección se transmite a modo de parámetro, en la referencia presentada a continuación podemos hallar lo ya mencionado

[40]. En términos de lo que respecta a fstat podemos apreciar como en la imagen de la parte inferior se muestran los parametros que la funcionalidad recibe lo cual es totalmente apreciable en la ilustración 35.

```
#include <unistd.h>
#include <sys/stat.h>
#include <sys/type.h>
int fstat(int filedes, struct stat *buf);
```

Ilustración 35 sintaxis para fstat.

ftell

Proporciona el valor del contador de posición de archivo correspondiente al flujo proporcionado, es equivalente a fgetpos(), lo ya mencionado puede ser hallado en la cita [41]. En la siguiente ilustración se aprecia con detalle un poco más de la sintaxis que podemos encontrar con ftell lo cual se aprecia de manera un poco más concreta para poder implementarlo en alguna solución de software que nosotros queramos proporcionar a nuestros clientes (ilustración 36).

```
long ftell(FILE *flujo)
```

Ilustración 36 sintaxis para ftell.

fullpath

La función _fullpath expande el nombre de la ruta relativa en relPath a su ruta completa o absoluta y almacena este nombre en absPath . Si absPath es NULL , malloc se usa para asignar un búfer de longitud suficiente para contener el nombre de la ruta. Es responsabilidad de la persona que llama liberar este búfer. Un nombre de ruta relativa especifica una ruta a otra ubicación desde la ubicación actual (como el directorio de trabajo actual: "."). Un nombre de ruta absoluta es la expansión de un nombre de ruta relativo que indica la ruta completa necesaria para llegar a la ubicación deseada desde la raíz del sistema de archivos. A diferencia de _makepath , _fullpathse puede utilizar para obtener el nombre de la ruta absoluta para las rutas relativas (relPath) que incluyen "./" o "../" en sus nombres, podemos encontrar lo ya mencionado en [42]. Lo ya descrito es más sencillo de apreciar si nosotros nos enfocamos en entender como es que los parametros que se reciben en la llamada se organizan, con lo que ahora se puede dar una mejor imagen de como es que se puede implementar en términos de código, todo esto en la ilustración 37.

```
char *_fullpath(char *absPath,const char *relPath,size_t maxLength);
```

Ilustración 37 sintaxis para _fullpath.

Es importante aclarar que la parte de absPath es un puntero a un búfer que contiene el nombre de la ruta absoluta o completa, o NULL, posteriormente relPath es la parte del nombre de la ruta relativa y finalmente maxLength tiene que ver con la longitud máxima del búfer de nombre de ruta absoluta (absPath), esta longitud está en bytes para _fullpath pero en caracteres anchos (wchar_t) para _wfullpath.

fwrite

Esta función escribe en el flujo señalado por flujo num_elementos bloques de tamaño bytes de longitud, leyéndolos de memoria a partir de la dirección dada por ptro . Como resultado, la función

proporciona el número de bloques escritos. Si se produce un error, el resultado es un número de bloques menor que el esperado o cero. Esta función, al no hacer cambio de formato alguno, es la que escribe más rápidamente en disco, lo dicho se puede revisar en [43]. Posteriormente en la zona inferior al párrafo en donde hemos expuesto como es que empleamos fwrite podemos ver una imagen con como es que en términos del código se puede agregar cada uno de los parametros correspondientes, todo dentro de la ilustración 38.

```
size_t fwrite(const void *ptro, size_t tamaño, size_t num_elementos, FILE *flujo)
```

Ilustración 38 sintaxis para fwrite.

getc

Esto es una macro equivalente a fgetc(), que lee del flujo proporcionado un carácter, si existe; lo mencionado puede ser corroborado en [44]. Ahora bien, en la ilustración siguiente podemos apreciar como es que es la sintaxis básica para el uso de getc (ilustración 39).

```
int getc(FILE *flujo)
```

Ilustración 39 sintaxis para getc.

getcurdir

Obtiene el directorio actual para la unidad especificada y ademas getcurdir obtiene el nombre del directorio de trabajo actual para la unidad indicada por unidad. Cabe aclarar que getcurdir devuelve 0 en caso de éxito o -1 en caso de error, para esta funcionalidad hemos recurrido a la fuente extranjera que aparece a continuación [45]. Con esto un poco más claro, podemos proceder a entender como es que en código debemos ingresar los parametros dentro de esta funcionalidad todo dentro de la ilustración 40.

```
int getcurdir (unidad int, directorio char *);
```

Ilustración 40 sintaxis para getcurdir.

getcwd

Un programa puede determinar el directorio que se encuentra funcionando en ese momento solicitando la función getcwd, lo cual se puede verificar gracias a una consulta hecha en un popular foro de la programación como parte de una aportación de la comunidad [46]. Posteriormente en la ilustración que aparece a continuación podemos ver los parametros y la sintaxis para la funcionalidad anteriormente expuesta (ilustración 41).

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

Ilustración 41 sintaxis para getcwd.

La función getcwd escribe el nombre del directorio actual en una memoria dada, buf. Envía NULL si el nombre del directorio supera el tamaño de la memoria (error ERANGE), proporcionado por el parámetro size. Si tiene éxito envía buf. getcwd puede enviar también NULL si se elimina el directorio (EINVAL) o si se modifican los permisos (EACCESS) cuando el programa sigue ejecutándose.

getdisk

Getdisk devuelve el número de unidad actual. setdisk devuelve el número total de unidades disponibles. Si el parámetro de unidad para setdisk es un número de unidad no válido, el valor devuelto es 0, del mismo modo gracias a una aportación de la comunidad pudimos entender de mejor manera el funcionamiento [47]. Si bien, después de dejar en claro lo anterior podemos ver como utilizar la parte del setter y el getter dentro de la funcionalidad anteriormente expuesta, siendo esto mostrado en la ilustración 42.

```
int getdisk(void);
int setdisk(int drive);
```

Ilustración 42 sintaxis para getdisk.

_getdrive

Devuelve una máscara de bits que representa las unidades de disco disponibles actualmente del mismo modo si la función tiene éxito, el valor de retorno es una máscara de bits que representa las unidades de disco disponibles actualmente. La posición del bit 0 (el bit menos significativo) es la unidad A, la posición del bit 1 es la unidad B, la posición del bit 2 es la unidad C, y así sucesivamente. Si la función falla, el valor de retorno es cero. Para obtener información de error ampliada, llame a GetLastError, lo dicho puede ser corroborado en [48]. De manera posterior, es pertinente poder mostrar en la siguiente imagen como es que se implementa en código _getdrives (ilustración 43)

```
unsigned long _getdrives( void );
```

Ilustración 43 sintaxis para _getdrives.

locking

Bloquea o desbloquea bytes de un archivo, cabe aclarar que la función _locking bloquea o desbloquea nbytes bytes del archivo especificado por fd . Bloquear bytes en un archivo evita el acceso a esos bytes por parte de otros procesos. Todo bloqueo o desbloqueo comienza en la posición actual del puntero del archivo y continúa durante los siguientes nbytes bytes, de manera que lo dicho puede ser corroborado en la fuente que aparece a continuación [49]. Lo dicho puede verse en la ilustración 44 de la zona inferior.

```
int _locking(int fd,int mode,long nbytes);
```

Ilustración 44 sintaxis para _locking.

lseek

Lseek nos ayuda a reposicionar el puntero de lectura/escritura de un fichero y en términos del valor devuelto en caso de una ejecución correcta, lseek devuelve la posición del puntero resultante medida en bytes desde el principio del fichero. También cabe aclarar que algunos dispositivos son incapaces de buscar y POSIX no especifica qué dispositivos deben soportar la búsqueda, con el que lo dicho se pudo corroborar en [50]. Lo mencionado se aprecia en la imagen de la parte inferior para poder ver de forma más clara los parametros con los que cumplen (ilustración 45).

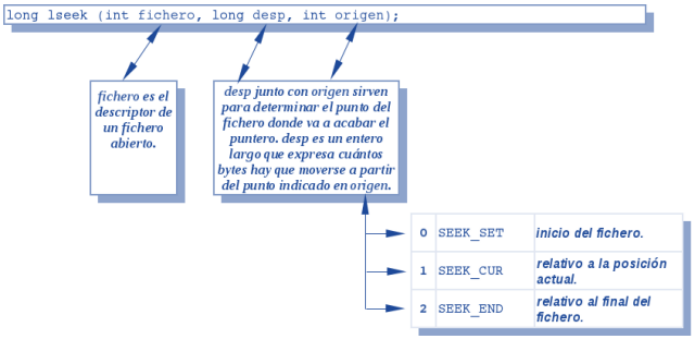


Ilustración 45 sintaxis para lseek.

_makepath

Ayudan a crear un nombre de ruta a partir de componentes, ahora bien los distintos parametros son path el cual nos dará la parte de la ruta del buffer, el parámetro drive contendrá una letra correspondiente a la unidad deseada y dos puntos finales opcionales, dir nos ayudará con la contención de los directorios del path, fname nos permitirá tener el archivo base con su extensión perfectamente referenciada mientras que ext contendrá la ruta actual de la extensión de nuestro archivo, se pudo hallar la información mencionada en [51]. Se puede ver en código lo dicho por medio de la ilustración 46 de la parte inferior.

```
void _makepath(char *path,const char *drive,const char *dir,const char *fname,const char *ext);
void _wmakepath(wchar_t *path,const wchar_t *drive,const wchar_t *dir,const wchar_t *fname,const wchar_t *ext);
```

Ilustración 46 sintaxis para _makepath.

mkdir

Los directorios pueden ser creados con la función mkdir, básicamente la función mkdir() creará un nuevo directorio con el nombre ruta . Los bits de permiso de archivo del nuevo directorio se inicializarán desde el modo . Estos bits de permiso de archivo del argumento de modo serán modificados por la máscara de creación de archivos del proceso del mismo modo una vez completado con éxito, mkdir () devolverá 0. De lo contrario, se devolverá -1, no se creará ningún directorio y se establecerá errno para indicar el error [52]. En la ilustración 47 podemos ver la implementación de la funcionalidad en términos de código.

```
#include <sys/stat.h>
int mkdir(const char *path, mode_t mode);
```

Ilustración 47 sintaxis para mkdir.

open

Se emplea para poder abrir el archivo para leerlo, escribirlo o ambos. En la imagen de abajo podemos ver la implementación en código de open (ilustración 48).

```
int open(const char *path, int oflags);
int open(const char *path, int oflags, mode_t mode);
```

Ilustración 48 sintaxis para open.

En la siguiente tabla podemos ver lo referente a los parametros acorde con la sintaxis establecida, ahora bien, const char *path nos dará la ruta relativa o absoluta al archivo que se va a abrir, int oflags nos brindará una lista de valores separados por bit a bit que determina el método en el que se abrirá el archivo (si debe ser de solo lectura, lectura / escritura, si debe borrarse al abrirlo, etc.). Consulte una lista de valores legales para este campo al final, mode_t mode nos permitirá tener una lista de valores separados por bit a bit que determina los permisos del archivo si se crea. Vea una lista de valores legales al final y finalmente en el retorno de valores obtendremos el descriptor de archivo para el nuevo archivo. El descriptor de archivo devuelto es siempre el número entero más pequeño mayor que cero que todavía está disponible. Si se devuelve un valor negativo, entonces hubo un error al abrir el archivo, con lo que ahora bien lo dicho fue hallado en [53].

perror

Esta función imprime en pantalla el mensaje de error del sistema asociado al valor de la variable errno . Si el argumento no tiene el valor NULL , la cadena se antepone al mensaje de error, empleando ' : ' como separador, con lo que en [54] podemos hallar lo mencionado. Lo dicho se aprecia en código en la ilustración 49.

```
void perror(const char *string)
```

Ilustración 49 sintaxis para perror.

putchar

Imprime en stdout el carácter que se le proporciona; equivale a putc() con stdout como argumento de salida, la explicación anterior fue encontrada en [55]. Ahora bien, podemos ver en términos del código la funcionalidad descrita en la ilustración 50.

```
int putchar(int c);
```

Ilustración 50 sintaxis para putchar.

read

Cabe aclarar que read es una llamada al sistema que se utiliza para leer datos en un búfer en donde int fildes es descriptor de archivo de dónde leer la entrada. Puede usar un descriptor de archivo obtenido de la llamada al sistema, para referirse a la entrada estándar, la salida estándar o el error estándar, respectivamente, ademas const void *buf nos permitirá acceder a la matriz de caracteres donde se almacenará el contenido leído, del mismo modo size_t nbytes será el número de bytes para leer antes de truncar los datos. Si los datos a leer son más pequeños que nbytes, todos los datos se guardan en el búfer y posteriormente el valor de retorno será el número de bytes leídos. Si el valor es negativo, la llamada al sistema devolvió un error, con lo cual del mismo modo que en las anteriores funciones nosotros podemos encontrar la información dicha en la siguiente cita [56]. Lo dicho lo vemos en la ilustración 51.

```
ssize_t read(int fildes, void *buf, size_t nbytes);
```

Ilustración 51 sintaxis para read.

remove

Esta función sirve para eliminar el directorio cuya ruta se le proporciona como argumento. Equivale a la llamada de sistema unlink() . La función remove() proporciona el valor 0 si concluye con éxito, o -1 en caso contrario. Si se produce un fallo (por las causas habituales al intentar eliminar un directorio), con lo que hemos podido fundamentar este conocimiento dentro de la referencia [57]. Lo mencionado puede apreciarse en la ilustración 52.

```
int remove(const char *path)
```

Ilustración 52 sintaxis para remove.

rename

Fundamentalmente la función rename() se usa para cambiar el nombre del archivo o directorio, es decir, de old_name a new_name sin cambiar el contenido presente en el archivo. Esta función toma el nombre del archivo como argumento, lo cual pudo se corroborado en [58]. Si bien, ahora en la ilustración 53 se puede ver lo mencionado.

```
int rename (const char *old_name, const char *new_name);
```

Ilustración 53 sintaxis para rename.

rewind

Esta función sirve para dar el valor cero al contador de archivo del flujo que se proporciona como argumento. Es equivalente a una llamada a fseek(fp, 0L, SEEK_SET), siendo precisamente la referencia [59] una base fuerte para hallar información acorde al tema. Lo dicho puede ser visto en la imagen de abajo (ilustración 54).

```
void rewind(FILE *stream)
```

Ilustración 54 sintaxis para rewind.

rmdir

La función rmdir () eliminará un directorio cuyo nombre viene dado por la ruta . El directorio se eliminará solo si es un directorio vacío. En términos de los valores retornados tendremos que una vez completada con éxito, la función rmdir () devolverá 0. De lo contrario, se devolverá -1 y errno se establecerá para indicar el error. Si se devuelve -1, el directorio nombrado no se cambiará, con lo que en [60] podremos encontrar la información ya expuesta. Lo dicho puede verse en la ilustración 55.

```
#include <unistd.h>
int rmdir(const char *path);
```

Ilustración 55 sintaxis para rmdir.

_rmtmp

La función _rmtmp limpia todos los archivos temporales en el directorio actual. La función elimina solo los archivos creados por tmpfile ; utilícelo solo en el mismo directorio en el que se crearon los archivos temporales, con lo que lo mencionado puede ser encontrado en la siguiente referencia [61]. En la siguiente imagen se ve como en la ilustración 56 como es la implementación en código.

```
int _rmtmp( void );
```

Ilustración 56 sintaxis para _rmtmp.

_rtl_creat

En términos de la funcionalidad del lenguaje _rtl_creat se afirma que abre el archivo especificado por ruta. El archivo siempre se abre en modo binario. Tras la creación exitosa del archivo, el puntero del archivo se establece al principio del archivo, en la siguiente referencia nos hemos basado [62]. Lo dicho se ve en la ilustración 57.

```
int _rtl_creat(const char *path, int attrib);
```

Ilustración 57 sintaxis para _rtl_creat.

_rtl_open

Este abre el archivo especificado por nombre de archivo, luego lo prepara para lectura o escritura, según lo determinado por el valor de oflags [63]. En la siguiente imagen se aprecia como es la implementación en términos de código (ilustración 58).

```
int _rtl_open(const char *filename, int oflags);
```

Ilustración 58 sintaxis para _rtl_open.

_rtl_read

En términos generales se sabe que _rtl_read nos ayuda a leer en bytes del archivo asociado con handle en el búfer al que apunta buf. Cuando se abre un archivo en modo texto, _rtl_read no elimina los retornos de carro, lo anteriormente mencionado fue hallado en la fuente que a continuación mostramos [64]. Lo dicho se puede ver en la ilustración 59.

```
int _rtl_read(int handle, void *buf, unsigned len);
```

Ilustración 59 sintaxis para _rtl_read.

_rtl_write

La función write () reescribe los bytes de recuento del búfer al que apunta el búfer en el archivo correspondiente al identificador del descriptor de archivo. El puntero de la posición del archivo se incrementa por el número de bytes escritos. Si el archivo se abre en modo texto, los avances de línea se complementan automáticamente con retornos de carro. Sin embargo, _rtl_write () no hace esto, lo dicho puede ser hallado en lo que respecta a la fuente [65]. Lo mencionado se aprecia en la ilustración 60.

```
int _rtl_write(int handle, void *buf, unsigned len);
```

Ilustración 60 sintaxis para _rtl_write.

_searchenv

En cuestión de la rutina _searchenv intenta localizar el archivo, buscando a lo largo de la ruta especificada por la variable de entorno del sistema operativo, la información puede ser encontrada en la cita siguiente [66]. Podemos ver con claridad en la siguiente imagen lo dicho (ilustración 61).

```
void _searchenv(const char *filename,const char *varname,char *pathname);
void _wsearchenv( const wchar_t *filename,const wchar_t *varname,wchar_t *pathname);
```

Ilustración 61 sintaxis para _searchchev.

setbuf

Si el sistema operativo admite búferes definidos por el usuario, setbuf()controla el almacenamiento en búfer para la secuencia especificada . La setbuf()función solo funciona en ILE C cuando se utiliza el sistema de archivos integrado. El puntero de flujo debe hacer referencia a un archivo abierto antes de que se haya realizado cualquier E/S o reposicionamiento, lo anteriormente mencionado fue hallado y resumido de la siguiente fuente [67]. Lo dicho se ve en la siguiente imagen (ilustración 62).

```
void setbuf(FILE *stream, char *buffer);
```

Ilustración 62 sintaxis para setdisk.

setdisk

En el caso de setdisk se establece que el disco actual en la unidad especificada por el parámetro de unidad. La unidad A es 0, la unidad B es 1 y así sucesivamente. La función devuelve el número total de discos del sistema, lo ya mencionado fue encontrado en la fuente siguiente [68]. En la siguiente ilustración 63 podemos ver la parte fundamental para implementación con código.

```
int getdisk(void);
int setdisk(int drive);
```

Ilustración 63 sintaxis para setdisk.

setmode

Esta función establece el modo del archivo dado en modo. También configurará el archivo en modo cocido o crudo en consecuencia, y configurará cualquier FILE* objeto que use este archivo en modo texto o binario, lo ya mencionado se puede encontrar en la siguiente referencia [69]. Lo dicho tendría su equivalente en código justo como se muestra en la siguiente imagen (ilustración 64).

```
int _setmode (int fd,int mode);
```

Ilustración 64 sintaxis para _setmode.

setvbuf

Las operaciones de entrada/salida de un flujo admiten tres modos de funcionamiento: sin memoria intermedia, por bloques o por líneas. Si no se usa memoria intermedia, la información enviada al flujo aparece inmediatamente en su destino. Si el funcionamiento es por bloques, los caracteres se almacenan en un bloque, que se envía a destino cuando se llena. Si el funcionamiento es por líneas, los caracteres se almacenan hasta que se escribe un carácter de nueva línea o se produce una lectura

en el flujo, lo ya mencionado pudo ser encontrado en la cita que aparece a continuación [70]. Lo mencionado se aprecia en el listado de abajo y en la ilustración 65.

Normalmente los archivos funcionan por bloques. La función setvbuf() sirve para modificar la configuración de memoria importante de un flujo. El parámetro de modo admite estos tres valores posibles:

- _IONBF Sin memoria intermedia
- _IOLBF Por líneas
- _IOFBF Por bloques, con acoplamiento completo.

```
int setvbuf(FILE *stream, char *buf, int modo, size_t tamaño)
```

Ilustración 65 sintaxis para setvbuf.

stat

Enfocándonos en la función stat() podemos decir que esta se utiliza básicamente para enumerar las propiedades de un archivo identificado por path, ademas cabe destacar que lee todas las propiedades del archivo y los vuelca a la subestructura. La función se define en el sys/stat.harchivo de encabezado, dentro de lo encontrado podemos remontarnos a la referencia [71]. Lo mencionado se aprecia en la ilustración 66 de la parte inferior.

```
int stat(const char *path, struct stat *buf);
```

Ilustración 66 sintaxis para stat.

system

Dentro de los lenguajes de programación tanto en C como en C++ tenemos la función system () es parte de la biblioteca estandarizada del lenguaje. Se utiliza para pasar los comandos que se pueden ejecutar en el procesador de comandos o en la terminal del sistema operativo, y finalmente devuelve el comando una vez completado, podemos hallar en dicha información en la siguiente referencia o cita bibliográfica [72]. Lo anterior puede verse en la parte inferior a este párrafo dentro de la ilustración 67.

```
int system(char command);
```

Ilustración 67 sintaxis para system.

_tempnam

La función tempnam () permite al usuario controlar la elección de un directorio. El argumento dir apunta al nombre del directorio en el que se creará el archivo. Si dir es un puntero nulo o apunta a una cadena que no es un nombre para un directorio apropiado, se utilizará el prefijo de ruta definido como _tmpdir en el encabezado <stdio.h>, la información que se ha mencionado se puede encontrar en la cita que aparece a continuación [73]. Lo dicho se ve con más claridad en el código dentro de la ilustración 68.

```
char *_tempnam( const char *dir, const char *prefix);
wchar_t *_wtempnam( const wchar_t *dir, const wchar_t *prefix);
```

Ilustración 68 sintaxis para _tempnam.

tmpfile

La función tmpfile () crea un archivo temporal en modo de actualización binaria en C. Se inicializa en el archivo de encabezado de un programa C. Siempre devuelve un puntero nulo si no se puede crear el archivo temporal. El archivo temporal se elimina automáticamente justo después de la finalización del programa, lo cual puede encontrarse en la cita que a continuación exponemos [74]. Lo mencionado se observa en la ilustración 69 de la parte inferior.

```
FILE *tmpfile(void)
```

Ilustración 69 sintaxis para tmpfile.

umask

En lo que respecta a umask tenemos que este cambia la máscara de creación de archivos del proceso. newmask especifica nuevos bits de permisos de archivo para la máscara de creación de archivos del proceso, esta máscara restringe la configuración de (o desactiva) los bits de permiso de archivo especificados en el argumento de modo que se usa con todosabierto(), creat(), mkdir() y mkfifo() funciones que emite el proceso actual. Los bits de permiso de archivo establecidos en 1 en la máscara de creación de archivos se establecen en 0 en los bits de permiso de archivo de los archivos creados por el proceso, lo mencionado lo podemos encontrar en la siguiente referencia [75]. Simplemente en la imagen de la parte inferior podemos ver el equivalente de lo que hemos mencionado en código (ilustración 70).

```
#define _POSIX_SOURCE
#include <sys/stat.h>

mode_t umask(mode_t newmask);
```

Ilustración 70 sintaxis para umask.

_unlink

La función unlink() eliminará un enlace a un archivo. Si la ruta nombra un enlace simbólico, unlink () eliminará el enlace simbólico nombrado por ruta y no afectará ningún archivo o directorio nombrado por el contenido del enlace simbólico. De lo contrario, unlink() eliminará el enlace nombrado por el nombre de ruta apuntado por ruta y disminuirá el recuento de enlaces del archivo al que hace referencia el enlace, lo cual puede ser corroborado en [76]. En la parte posterior (ilustración 71)podremos ver de forma clara la implementación de la funcionalidad en código.

```
int _unlink(const char *filename);
int _wunlink(const wchar_t *filename);
```

Ilustración 71 sintaxis para _unlink.

unlock

La función pthread_mutex_unlock() liberará el objeto mutex referenciado por mutex . La forma en que se libera un mutex depende del atributo de tipo del mutex. Si hay subprocesos bloqueados en el objeto mutex al que hace referencia mutex cuando se llama a pthread_mutex_unlock(), lo que hace que el mutex esté disponible, la política de programación determinará qué subproceso adquirirá el mutex, en términos del uso podemos utilizarlo para el desbloqueo de la utilización en este caso del lock, lo cual puede ser encontrado en [77]. En la parte inferior vemos una imagen relativa a la implementación del código en distintas circunstancias o estado (ilustración 72).

```
#include < pthread.h >

int pthread_mutex_lock (pthread_mutex_t * mutex );
int pthread_mutex_trylock (pthread_mutex_t * mutex );
int pthread_mutex_unlock (pthread_mutex_t * mutex );
```

Ilustración 72 uso de unlock, lock y tryunlock.

_utime

La función _utime establece la hora de modificación para el archivo especificado por nombre de archivo . El proceso debe tener acceso de escritura al archivo para cambiar la hora. En el sistema operativo Windows, puede cambiar la hora de acceso y la hora de modificación en la estructura _utimbuf , lo cual puede encontrarse en [78]. Ahora bien, en la siguiente parte podemos ver la implementación del código ya descrito (ilustración 73)

```
int _utime(const char *filename,struct _utimbuf *times);
```

Ilustración 73 sintaxis para _utime.

Tabla de resumen de funciones basadas en la fuente que se nos proporcionó.

En la siguiente tabla (tabla número 2), se muestran las distintas funciones disponibles en el lenguaje de programación C y del mismo modo en C++; cabe aclarar que la tabla que se muestra está basada en la fuente que usted nos proporcionó en el formato de la práctica, pero han sido detalladas en la parte anterior de forma específica y de manera mucho más concreta. [10]

Tabla 2 Resumen del listado, basado en el link que se nos compartió en el formato de la práctica número 8.

Nombre	Descripción	H.	UNIX	W32	C	C++
access	Determina la existencia del fichero y el tipo de acción posible sobre él (lectura, escritura, ejecución).	-	X	X	-	-
chdir	Cambiar el directorio actual.	-	X	X	-	-
chmod	Cambiar el tipo de acceso permitido a un fichero.	-	X	X	-	-
chsize	Cambiar (aumentar o disminuir) el tamaño de un fichero	I	-	X	-	-
closedir	Cerrar un directorio-stream.	-	X	X	-	-
_creat	Crear un fichero nuevo definido por su path-name.	I	X	X	-	-
creattemp	Crear un fichero de nombre único en un directorio.	I	-	X	-	-

dup	Crear un nuevo handle a partir de otro existente.	I	X	X	-	-
dup2	Similar al anterior	I	X	X	-	-
eof	Determinar si el handle ha alcanzado el final de fichero	I	-	X	-	-
fclose	Cerrar un fichero	F	X	X	X	X
_fdopen	Permite crear un nuevo flujo (definido por un handle F) y asociarlo con un handle obtenido.	F/I	X	X	-	-
feof	Detectar si se ha alcanzado el final de fichero después de una operación de lectura.	F	X	X	X	X
ferror	Detectar si se ha producido error después de una operación de L/E en el fichero.	F	X	X	X	X
fflush	Forzar el vaciado del buffer de salida y consiguiente escritura de datos al disco.	F	X	X	X	X
fgetc	Leer el próximo carácter del fichero	F	X	X	X	X
fgetpos	Obtener la posición actual del fp de un fichero.	F	-	X	X	X
fgets	Leer un número de caracteres de un fichero	F	X	X	X	X
findclose	Cierra cualquier handle y libera la memoria dinámica asociada con invocaciones previas a findfirst y findnext	-	-	X	-	-
findfirst	Comienza la búsqueda de ficheros definidos por atributos o comodines en un directorio de disco.	-	-	X	-	-
findnext	Continúa la búsqueda iniciada con findfirst.	-	-	X	-	-
filength	Obtener el tamaño en bytes de un fichero.	I	-	X	-	-
_fileno	Obtener el número de manejador asociado a un handle tipo FILE.	F/I	X	X	-	-
fnmerge	Construir un path-name a partir de sus componentes.	-	-	X	-	-
fnsplit	Desmembrar un path-name en sus componentes.	-	-	X	-	-
fopen	Abrir un fichero en diversos modos (lectura, escritura, crearlo si no existe, etc).	F	X	X	X	X
fprintf	Componer y formatear una cadena de caracteres y escribir el resultado en un fichero.	F	X	X	X	X
fputc	Escribir un carácter en un fichero.	F	X	X	-	-
fputs	Escribir una matriz C en un fichero.	F	X	X	X	X
fread	Leer n ítems de tamaño t bytes de un fichero almacenándolos en un buffer.	F	X	X	X	X
freopen	Asociar un nuevo fichero a un manejador (handle) previamente asociado a otro fichero abierto.	F	X	X	X	X
fscanf	Leer una serie de campos de un fichero formateando los datos recibidos.	F	X	X	X	X
fseek	Establecer una nueva posición para el fp de un fichero abierto.	F	X	X	X	X
fsetpos	Establecer el fp asociado a un fichero a una posición obtenida previamente con fgetpos.	F	-	X	X	X
fstat	Fijar datos (tamaño, último acceso, etc) relativos a un fichero o directorio.	I	X	X	-	-
ftell	Obtener la posición actual (bytes desde el origen) del fp de un fichero.	F	X	X	X	X
fullpath	Convertir un path-name relativo en absoluto.	-	-	X	-	-
fwrite	Escribir n items de tamaño t bytes de un buffer en un fichero.	F	X	X	X	X
getc	Lee el próximo carácter del fichero.	F	X	X	X	X
getcurdir	Obtener el nombre del directorio activo en un volumen lógico determinado.	-	-	X	-	-

getcwd	Obtener el nombre del directorio activo actual incluyendo nombre del disco.	-	-	X	-	-
getcdwd	Similar al anterior	-	-	X	-	-
getdisk	Obtener el número del disco actual.	-	-	X	-	-
_getdrive	Obtener el número del disco actual.	-	?	X	?	?
getftime	Obtener los datos de fecha y hora de un fichero.	I	-	X	-	-
lock	Bloquear un fichero.	I	-	X	-	-
locking	Bloquear y desbloquear una porción de un fichero.	I	-	X	-	-
lseek	Mueve el fp de un fichero a una nueva posición.	I	X	X	-	-
_makepath	Construir un path-name a partir de sus componentes.	-	-	X	-	-
mkdir	Crear un directorio.	-	X	X	-	-
open	Abrir un fichero en diversos modos.	I	X	X	-	-
perror	Escribir en el buffer estándar de salida.					
_pipe	Establece un fichero en memoria que puede servir para compartir datos entre procesos.	I	-	X	-	-
_popen	Crea un "pipe" con el procesador de comandos.		-	X	-	-
putc	Escribe un carácter en un fichero.	F	X	X	X	X
read	Leer n bytes de un fichero y almacenarlos en un buffer.	I	X	X	-	-
remove	Borrar un fichero definido por su path-name.	-	X	X	X	X
rename	Permite renombrar un fichero dentro de un directorio y/o moverlo de directorio	-	-	X	X	X
rewind	Posicionar el fp al principio del fichero.	F	X	X	X	X
rmdir	Borrar un directorio.	-	X	X	-	-
_rmtmp	Borrar los ficheros temporales previamente creados.	-	-	X	-	-
_rtl_chmod	Obtener/cambiar los atributos del fichero.	-	X	X	X	-
_rtl_creat	Crear un fichero y abrirlo para L/E binario (pueden ser ficheros ocultos o de Sistema)	I	X	X	X	-
_rtl_open	Abrir un fichero para L, E o L/E binario.	I	-	X	-	-
_rtl_read	Leer n bytes de un fichero.	I	X	X	X	-
_rtl_write	Escribir n bytes en un fichero.	I	X	X	X	-
_searchenv	Busca un fichero en un path de entorno.	-	-	X	-	-
_searchstr	Buscar un fichero en una lista de directorios.	-	-	X	-	-
setbuf	Utilizar un buffer específico para las operaciones de E/S de un flujo en lugar del establecido por el Sistema.	F	X	X	X	X
setdisk	Establecer el número del disco actual.	-	-	X	-	-
setftime	Establecer los datos de fecha y hora de un fichero.	I	-	X	-	-
setmode	Fijar el modo de L/E (binario/texto) de un fichero abierto previamente en otro modo.	I	-	X	-	-
setvbuf	Utilizar un buffer específico en lugar del proporcionado por el Sistema para operaciones de E/S asociadas a un fichero. Permite establecer la forma en que se manejará el buffer.	F	X	X	X	X
_seplitpath	Desmembrar un path-name en sus componente.	-	-	X	-	-
stat	Obtener información (tamaño, último acceso, etc) relativa a un fichero o directorio.	-	X	X	-	-
system	Invocar el intérprete de comandos del SO para ejecutar un comando.		X	X	-	-
_tempnam	Obtener un nombre de fichero no utilizado en un directorio. Es posible especificar los 5 primeros caracteres de este nombre; el resto son proporcionados aleatoriamente por el Sistema.	-	X	X	-	-

tmpfile	Crea un fichero temporal de nombre aleatorio y lo abre para lectura-escritura binaria. El fichero es destruido automáticamente al cerrarlo o al terminar el programa.	**F**	**X**	**X**	**X**	**X**
umask	Establecer los parámetros de L/E utilizados por las funciones open y _creat	**-**	**-**	**X**	**-**	**-**
_unlink	Borrar un fichero definido por su path-name.	**-**	**X**	**X**	**-**	**-**
unlock	Desbloquear un fichero previamente bloqueado con lock.	**I**	**-**	**X**	**-**	**-**
_utime	Permite establecer la fecha/hora de modificación de un fichero a un valor determinado.	**-**	**X**	**X**	**-**	**-**

Códigos y ventanas de ejecución

Programa81.c

Realizar un programa en lenguaje C en Linux que permita la creación, edición y borrado de un archivo de texto. En otras palabras, crear un editor de textos básico. El programa debe permitir crear un directorio nuevo si el usuario lo requiere.

Código completo y explicación en general

Código completo

En la siguiente parte de nuestro reporte de práctica podremos apreciar el código completo (ilustración 74) de nuestra solución para el problema planteado, cabe aclarar que de manera posterior podremos encontrar una descripción detallada parte por parte del código que aquí hemos presentado, con el fin de poder ser lo más claros y precisos en lo que respecta a clarificar el funcionamiento del código.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>


int nombreArchivoValido(char *nombreEntrada){
    char *ptnMovil=nombreEntrada;
    int tamEntrada=strlen(nombreEntrada);
    char const prefijoTxt[5]=".txt";
    int tamPrefijo=strlen(prefijoTxt);
    return (tamEntrada > tamPrefijo) && (0==strcmp(nombreEntrada+(tamEntrada-
tamPrefijo),prefijoTxt));
}

int nombreDirArchivoValido(char *nombreEntrada){
    char *ptnMovil=nombreEntrada;
    int tamEntrada=strlen(nombreEntrada);
    char const prefijoTxt[5]="/";
    int tamPrefijo=strlen(prefijoTxt);
    return (tamEntrada > tamPrefijo) && (0==strcmp(nombreEntrada+(tamEntrada-
tamPrefijo),prefijoTxt));
}

void crearArchivo(char *nombreArchivo){
  FILE *fp1;
  char c;
  fp1=fopen(nombreArchivo,"w");
  printf("\n\t Escriba el nombre de texto y presiona '.' para salvarlo \n\n\t");
```

```
  while(1){
    scanf("%c",&c);
    fputc(c,fp1);
   if(c=='.'){
     fclose(fp1);
     break;
   }
  }

}

void modificarArchivo(){
    FILE *fp1;
    char fn[20],c;
    printf("Escriba nombre de archivo\n ");
    scanf("%s",fn);
    fflush(stdin);
    fp1=fopen(fn,"r");
    if(fp1==NULL){
      printf("\nArchivo no encontrado!\n");
    }else{
        while(!feof(fp1)){
            c=getc(fp1);
            printf("%c",c);
        }
        fclose(fp1);
          printf("\n\t Escriba el nombre de texto y presiona '.' para salvarlo \
n\n\t");
          fp1=fopen(fn,"a");
          while(1){
            scanf("%c",&c);
            fputc(c,fp1);
           if(c=='.'){
             fclose(fp1);
             break;
           }
          }
     }
}

void eliminarArchivo(){
 FILE *fp1;
    char fn[20],c;
    printf("Escriba nombre de archivo\n ");
    scanf("%s",fn);
    fflush(stdin);
    fp1=fopen(fn,"r");
    if(fp1==NULL)
```

```
    {
      printf("\nArchivo no encontrado!\n");
    }else{
          remove(fn);
    }

}

int main(){
   int opcUsu=0;
   int opcDirArc=0;
   int salidaUsu=0;
   char nombreArchivo[40];
   int salidaNomArchivo=0;
   int salidaDirArchivo=0;
   int dirArchivoCorrecto=0;
   char dirArchivo[80];
   char archivoNombreFinal[120];
   while(salidaUsu==0){
          salidaNomArchivo=0;
          salidaDirArchivo=0;
          strcpy(nombreArchivo,"");
          strcpy(dirArchivo,"");
          strcpy(archivoNombreFinal,"");
       printf("\nBienvenido al editor de texto, escriba la opcción que desea hac
er:\n");
       printf("1-Hacer un nuevo archivo de texto\n");
       printf("2-Editar un archivo de texto ya creado\n");
       printf("3-Borrar un archivo de texto\n");
       printf("4-Salir\n");
       printf("\n");
       scanf("%d",&opcUsu);
       switch(opcUsu){
       case 1:
          while(salidaNomArchivo==0){
              printf("Escriba nombre de archivo que desea crear\n");
          scanf("%s",nombreArchivo);
          fflush(stdin);
          if(nombreArchivoValido(nombreArchivo)==1){
            salidaNomArchivo=1;
          }else{
            printf("Escriba nombre de archivo con teminación .txt\n");
          }
          }
          while(salidaDirArchivo==0){
             printf("Eliga una de las siguientes opcciones\n");
             printf("1-
Hacer un nuevo archivo de texto en el directorio actual\n");
```

```
            printf("2-
Hacer un nuevo archivo de texto en el directorio creado terminado como la siguie
nte forma 'directorio/' \n");
            scanf("%d",&opcDirArc);
            switch(opcDirArc){
               case 1:
               salidaDirArchivo=1;
               break;
               case 2:
               salidaDirArchivo=1;
               while(dirArchivoCorrecto==0){
                  printf("Escriba el nombre del directorio \n");
              scanf("%s",dirArchivo);
              fflush(stdin);
              if(nombreDirArchivoValido(dirArchivo)==1){
            dirArchivoCorrecto=1;
                 strcat(archivoNombreFinal,dirArchivo);
                 mkdir(dirArchivo,0700);
             }else{
            printf("Escriba dirección de archivo valido\n");
            }
                }
               break;
               default:
               printf("Escriba una opcción valida\n");
               break;
            }
            strcat(archivoNombreFinal,nombreArchivo);
            crearArchivo(archivoNombreFinal);
         }
          break;
      case 2:
          modificarArchivo();
         break;
      case 3:
         eliminarArchivo();
         break;
      case 4:
         printf("\nAdios\n");
         salidaUsu=1;
         break;
      default:
        printf("\nEliga una opcción valida\n");
    }
  }
   return 0;
}
```

Ilustración 74 Código completo de la práctica número 8.

Explicación código por partes

En esta parte del código se importan las bibliotecas estándar para el programa. La biblioteca de stdio.h es para las operaciones estándar del lenguaje C. La librería string.h se usa para hacer las operaciones con string. Las librerías de sys/types.h y sys/stat.h es para poder tener una estructura de datos del sistema para usarlas en el código. La última librería unistd.h es para poder tener acceso a la API del sistema operativo POSIX. Lo ya mencionado, se puede apreciar en la imagen de la parte inferior (ilustración 75).

```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

Ilustración 75 Implementación de bibliotecas estándar.

Esta función llamada nombreArchivoValido evalúa si el nombre de un archivo de texto es válido regresa un valor integer dependiendo del resultado. Regresa cero si no es válido y uno si es válido. Esta función recibe un puntero char llamado nombreEntrada.Se asigna un valor a tamEntrada con la función de strlen donde se obtiene el tamaño del string nombreEntrada. Luego se guarda el prefijo del archivo a buscar en una variable string constante llamada prefijoTxt en este caso ".txt". Se asigna un valor a tamPrefijo con la función de strlen donde se obtiene el tamaño del string prefijoTxt. Al final se retornó el valor de aplicar la operación lógica and a la comparación de que tamEntrada es mayor que tamPrefijo y que la función strcmp, que dice si dos string son iguales, da igual a cero si esta recibe dos parámetros en sí. El primero siendo el puntero de archivo del texto del archivo, llamado nombreEntrada, movido un cierto número de pasos , que son dependientes del resultado de restar a tamEntrada el valor de tamPrefijo. El segundo valor es el string prefijoText. Es importante aclarar que lo expuesto es claramente visible en la imagen que podemos ver en la parte de abajo (ilustración 76).

```c
int nombreArchivoValido(char *nombreEntrada){
    char *ptnMovil=nombreEntrada;
    int tamEntrada=strlen(nombreEntrada);
    char const prefijoTxt[5]=".txt";
    int tamPrefijo=strlen(prefijoTxt);
    return (tamEntrada > tamPrefijo) && (0==strcmp(nombreEntrada+(tamEntrada-tamPrefijo),prefijoTxt));
}
```

Ilustración 76 implementación de nombreArchivoValido.

Esta función llamada nombreDirArchivoValido evalúa si el nombre de un directorio es válido regresa un valor integer dependiendo del resultado. Regresa cero si no es válido y uno si es válido. Esta función recibe un puntero char llamado nombreEntrada.Se asigna un valor a tamEntrada con la función de strlen donde se obtiene el tamaño del string nombreEntrada. Luego se guarda el prefijo del directorio a buscar en una variable string constante llamada prefijoTxt en este caso "/". Se asigna un valor a tamPrefijo con la función de strlen donde se obtiene el tamaño del string prefijoTxt. Al final se retornó el valor de aplicar la operación lógica and a la comparación de que tamEntrada es mayor que tamPrefijo y que la función strcmp, que dice si dos string son iguales, da igual a cero si esta recibe dos parámetros en sí. El primero siendo el puntero de nombre del directorio , llamado nombreEntrada, movido un cierto número de pasos , que son dependientes del resultado de restar a tamEntrada el valor de tamPrefijo. El segundo valor es el string prefijoText.

Si bien, es importante declarar que en la siguiente imagen podemos apreciar lo expuesto anteriormente, la imagen tiene el identificador de ilustración 77.

```
int nombreDirArchivoValido(char *nombreEntrada){
    char *ptnMovil=nombreEntrada;
    int tamEntrada=strlen(nombreEntrada);
    char const prefijoTxt[5]="/";
    int tamPrefijo=strlen(prefijoTxt);
    return (tamEntrada > tamPrefijo) && (0==strcmp(nombreEntrada+(tamEntrada-tamPrefijo),prefijoTxt));
}
```

Ilustración 77 implementación de nombreDirArchivoValido.

La función de crearArchivo es la usada para poder crear un archivo. Esta recibe un puntero char nombreArchivo que es el nombre del archivo a crear. Primero se crea un puntero de archivo llamado fp1 y un carácter llamado c. Se asigna el valor de fp1 con la función fopen para crear un archivo con el nombre del archivo y en el modo de escritura con el parámetro "w". Luego se escribe las instrucciones que con el printf. Luego se hace un ciclo while infinito donde con la función scanf se asigna el valor tecleado por el usuario a c y luego se escribe este valor en el archivo creado con la función fputc en el archivo de fp1. Si el valor de c es el del carácter "." Se cierra el archivo fp1 con la función fclose y se sale del ciclo y termina la función. Se debe tener en cuenta que en la siguiente imagen de nuestro código se muestra lo explicado de manera cabal (ilustración 78).

```
void crearArchivo(char *nombreArchivo){
  FILE *fp1;
  char c;
  fp1=fopen(nombreArchivo,"w");
  printf("\n\t Escriba el nombre de texto y presiona '.' para salvarlo \n\n\t");
  while(1){
    scanf("%c",&c);
    fputc(c,fp1);
   if(c=='.'){
      fclose(fp1);
      break;
    }
  }

}
```

Ilustración 78 implementación de crearArchivo.

La función modificarArchivo para modificar un archivo crea un puntero de archivo fp1, un string de caracteres llamado fn y un dato tipo carácter llamado c. Luego se imprime la petición al usuario de escribir el nombre del archivo. Con la función scanf se asigna el valor del usuario a la variable fn y se limpia la el buffer en entrada con la función fflush. Se asigna el valor de fp1 con la función fopen con el nombre de archivo con el valor de fn y el parámetro de lectura "r". Se evalúa si fp1 está vacío y en caso de estarlo da el mensaje de que no se encontró el archivo. Si no está vacío se entra en un ciclo que sigue hasta que la negación de la función feof de un valor verdadero indicando que se terminó de leer el archivo. Dentro del ciclo while se usa la función getc para obtener el carácter en turno del archivo de fp1 e imprimirlo en la pantalla. Después del while se cierra el archivo fp1 para luego imprimir la orden al usuario de escribir un nuevo texto a agregar. Luego se le asigna a fp1 el valor del valor de retorno de la función fopen con el valor de fn y el parámetro de adjuntar "a". Se pasa luego en un ciclo while infinito donde se usa la función scanf para asigna r a la variable c el valor del carácter que teclea el usuario. El valor de fputc le imprime al archivo del puntero fp1 el

valor de c y si c es igual al carácter “.” con fclose se deja el archivo del puntero fp1 ,se sale del ccilo while para al final terminar la función. Si bien, llegado a este punto es importante declarar que lo desglosado previamente se puede apreciar con mayor detalle en la siguiente imagen (ilustración 79).

```
void modificarArchivo(){
    FILE *fp1;
    char fn[20],c;
    printf("Escriba nombre de archivo\n ");
    scanf("%s",fn);
    fflush(stdin);
    fp1=fopen(fn,"r");
    if(fp1==NULL){
      printf("\nArchivo no encontrado!\n");
    }else{
        while(!feof(fp1)){
            c=getc(fp1);
            printf("%c",c);
        }
        fclose(fp1);
        printf("\n\t Escriba el nombre de texto y presiona '.' para salvarlo \n\n\t");
        fp1=fopen(fn,"a");
        while(1){
          scanf("%c",&c);
          fputc(c,fp1);
         if(c=='.'){
          fclose(fp1);
          break;
         }
        }
    }
}
```

Ilustración 79 implementación de modificarArchivo.

La función eliminarArchivo es para eliminar un archivo seleccionado. Primero se crea un puntero de archivo, un string denominado fn y una variable de tipo carácter llamada c. Se imprime el mensaje para que el usuario escriba el nombre del archivo a eliminar. Se usa la función scanf para asignar el valor del usuario a la variable fn y luego se limpia el buffer de entrada con la función fflush. La variable fp1 se le asigna el valor devuelto por la función fopen con el valor del archivo de fn y en el modo de lectura “r”. Si fp1 es igual a nulo imprime el mensaje de que no se encontró el archivo , en caso contrario usa la función de remove para eliminar el archivo. A continuación, en la imagen presentada a continuación (ilustración 80) se ve como lo descrito fue implementado.

```
void eliminarArchivo(){
  FILE *fp1;
    char fn[20],c;
    printf("Escriba nombre de archivo\n ");
    scanf("%s",fn);
    fflush(stdin);
    fp1=fopen(fn,"r");
    if(fp1==NULL)
    {
      printf("\nArchivo no encontrado!\n");
    }else{
          remove(fn);
    }

}
```

Ilustración 80 implementación de eliminarArchivo.

La función main es para darle al usuario el menú para poder crear, editar y eliminar un archivo de texto, lo cual puede apreciarse en la ilustración número 81 de la zona de abajo.

```
int main(){
   int opcUsu=0;
   int opcDirArc=0;
   int salidaUsu=0;
   char nombreArchivo[40];
   int salidaNomArchivo=0;
   int salidaDirArchivo=0;
   int dirArchivoCorrecto=0;
   char dirArchivo[80];
   char archivoNombreFinal[120];
   while(salidaUsu==0){
          salidaNomArchivo=0;
          salidaDirArchivo=0;
          strcpy(nombreArchivo,"");
          strcpy(dirArchivo,"");
          strcpy(archivoNombreFinal,"");
       printf("\nBienvenido al editor de texto, escriba la opcción que desea hac
er:\n");
       printf("1-Hacer un nuevo archivo de texto\n");
       printf("2-Editar un archivo de texto ya creado\n");
       printf("3-Borrar un archivo de texto\n");
       printf("4-Salir\n");
       printf("\n");
       scanf("%d",&opcUsu);
       switch(opcUsu){
       case 1:
          while(salidaNomArchivo==0){
              printf("Escriba nombre de archivo que desea crear\n");
          scanf("%s",nombreArchivo);
          fflush(stdin);
          if(nombreArchivoValido(nombreArchivo)==1){
            salidaNomArchivo=1;
          }else{
            printf("Escriba nombre de archivo con teminación .txt\n");
          }
          }
          while(salidaDirArchivo==0){
             printf("Eliga una de las siguientes opcciones\n");
             printf("1-
Hacer un nuevo archivo de texto en el directorio actual\n");
             printf("2-
Hacer un nuevo archivo de texto en el directorio creado terminado como la siguie
nte forma 'directorio/' \n");
             scanf("%d",&opcDirArc);
             switch(opcDirArc){
```

```
                case 1:
                salidaDirArchivo=1;
                break;
                case 2:
                salidaDirArchivo=1;
                while(dirArchivoCorrecto==0){
                   printf("Escriba el nombre del directorio \n");
               scanf("%s",dirArchivo);
               fflush(stdin);
               if(nombreDirArchivoValido(dirArchivo)==1){
            dirArchivoCorrecto=1;
                 strcat(archivoNombreFinal,dirArchivo);
                 mkdir(dirArchivo,0700);
             }else{
            printf("Escriba dirección de archivo valido\n");
            }
                }
                break;
                default:
                printf("Escriba una opcción valida\n");
                break;
             }
             strcat(archivoNombreFinal,nombreArchivo);
             crearArchivo(archivoNombreFinal);
          }
           break;
      case 2:
           modificarArchivo();
          break;
      case 3:
          eliminarArchivo();
          break;
      case 4:
          printf("\nAdios\n");
          salidaUsu=1;
          break;
      default:
        printf("\nEliga una opcción valida\n");
    }
  }
   return 0;
}
```

Ilustración 81 Implementación del contenido de main en la implementación de la práctica 8.

Primero se declara las variables integer opcUsu (para saber la opción del menú seleccionada por el usuario), opcDirArc (para saber como quiere tratar la dirección del archivo si desea crearlo),salidaUsu (para controlar el while que da el ciclo del todo el menú),salidaNomArchivo (para controlar el bucle que asegura que cuando se ingresa el nombre del archivo sea correcto), salidaDirArchivo (para

controlar el bucle que asegura que cuando se ingresa una opción de creación de directorio valida) y dirArchivoCorrecto (para controlar el bucle que asegura que cuando se escriba la dirección del directorio de un archivo sea correcta). También se crea los string de nombreArchivo (string del nombre del archivo),dirArchivo (nombre del directorio del archivo si se desea crear uno) y dirArchivoCorrecto (nombre del archivo y del directorio si es necesario).Luego se entra en el ciclo que despliega el menú controlado por la variable de salidaUsu. Cabe destacar que lo dicho se ve con más claridad en la ilustración 82.

```
int main(){
    int opcUsu=0;
    int opcDirArc=0;
    int salidaUsu=0;
    char nombreArchivo[40];
    int salidaNomArchivo=0;
    int salidaDirArchivo=0;
    int dirArchivoCorrecto=0;
    char dirArchivo[80];
    char archivoNombreFinal[120];
    while(salidaUsu==0){
      /*Código del menú*/
    }
    return 0;
}
```

Ilustración 82 Declaraciones para el programa.

Dentro del ciclo while del menú, se ponen las variables de salidaNomArchivo y salidaDirArchivo se iguala a cero para tener valores iguales a como eran al principio del programa. La función strcpy aplicada a las variables nombreArchivo,dirArchivo y archivoNombreFinal hace lo mismo. Luego se imprime las opciones que puede seleccionar el usuario y se usa la función scanf para asignar la opción seleccionada por el usuario a la variable opcUsu. Podemos ver en la imagen de abajo, como se lleva a cabo la implementación de nuestro menú (ilustración 83).

```
salidaNomArchivo=0;
salidaDirArchivo=0;
strcpy(nombreArchivo,"");
strcpy(dirArchivo,"");
strcpy(archivoNombreFinal,"");
printf("\nBienvenido al editor de texto, escriba la opcción que desea hacer:\n")
;
printf("1-Hacer un nuevo archivo de texto\n");
printf("2-Editar un archivo de texto ya creado\n");
printf("3-Borrar un archivo de texto\n");
printf("4-Salir\n");
printf("\n");
scanf("%d",&opcUsu);
switch(opcUsu){
```

```
    //código de opciones del menú
}
```

Ilustración 83 Integración del menú del programa.

Para el caso de 1 se entra en un ciclo while controlado por la variable salidaNomArchivo que seguirá mientras esta variable sea igual a cero donde se le dice al usuario que escriba el nombre del archivo a escribir para luego guardar ese valor mediante la función scanf la variable nombreArchivo y limpiar el buffer de entrada del teclado con fflush. Se evalúa con la función nombreArchivoValido la variable nombreArchivo de manera que si regresa uno cambia el valor de salidaNomArchivo a uno para salir del bucle while y si no le dice al usuario que el nombre de archivo no tiene el formato correcto.

Luego se entra a un ciclo controlado por la variable salidaDirArchivo que seguirá mientras esta tenga el valor de uno. En este ciclo imprime si el usuario quiere crear un directorio propio para el archivo. Si elige uno solo la variable salidaDirArchivo se iguala a uno, pero si es dos entra a otro bucle de while que seguirá mientras la variable dirArchivoCorrecto sea igual a cero. En el bucle se pregunta por el nombre del directorio a crear, se guarda el valor del usuario con la función scanf en la variable dirArchivoCorrecto y luego se limpia el buffer de entrada con la función fflush. Luego se evalúa con la función nombreDirArchivoValido la variable dirArchivo que en caso de regresar uno iguala la variable dirArchivoCorrecto a uno, concatena las cadenas archivoNombreFinal y dirArchivo con la función strcat y con la función mkdir se crea la dirección que indica la variable dirArchivo. Si no se cumple la función manda el mensaje de que el nombre del directorio es incorrecto. Para el default del switch para escribir un directorio dice que escriba un directorio imprime el mensaje de que el usuario debe elegir una opción válida.

Al final se concatena las cadenas archivoNombreFinal y nombreArchivo con la función strcat, y se llama la función crearArchivo con la cadena archivoNombreFinal. Lo dicho puede ser visto en la ilustración 84.

```
case 1:
          while(salidaNomArchivo==0){
              printf("Escriba nombre de archivo que desea crear\n");
          scanf("%s",nombreArchivo);
          fflush(stdin);
          if(nombreArchivoValido(nombreArchivo)==1){
            salidaNomArchivo=1;
          }else{
            printf("Escriba nombre de archivo con teminación .txt\n");
          }
          }
          while(salidaDirArchivo==0){
             printf("Eliga una de las siguientes opcciones\n");
             printf("1-
Hacer un nuevo archivo de texto en el directorio actual\n");
             printf("2-
Hacer un nuevo archivo de texto en el directorio creado terminado como la siguie
nte forma 'directorio/' \n");
             scanf("%d",&opcDirArc);
             switch(opcDirArc){
                case 1:
```

```
            salidaDirArchivo=1;
            break;
            case 2:
            salidaDirArchivo=1;
            while(dirArchivoCorrecto==0){
               printf("Escriba el nombre del directorio \n");
           scanf("%s",dirArchivo);
           fflush(stdin);
           if(nombreDirArchivoValido(dirArchivo)==1){
      dirArchivoCorrecto=1;
            strcat(archivoNombreFinal,dirArchivo);
            mkdir(dirArchivo,0700);
       }else{
      printf("Escriba dirección de archivo valido\n");
      }
            }
            break;
            default:
            printf("Escriba una opcción valida\n");
            break;
       }
       strcat(archivoNombreFinal,nombreArchivo);
       crearArchivo(archivoNombreFinal);
    }
     break;
```

Ilustración 84 Implementación del primer caso.

Para el caso de 2 se llama a la función modificarArchivo, esto puede ser visto en la ilustración 85.

```
case 2:
    modificarArchivo();
    break;
```

Ilustración 85 Implementación del segundo caso.

Para el caso de 3 se llama a la función eliminarArchivo, lo mencionado puede verse en la ilustración 86.

```
case 3:
    eliminarArchivo();
    break;
```

Ilustración 86 Implementación del tercer caso.

Para el caso de 4 se imprime un mensaje de adiós y se asigna la variable salidaUsu con el valor uno, que puede apreciarse en la ilustración 87

```
case 4:
    printf("\nAdios\n");
    salidaUsu=1;
    break;
```

Ilustración 87 Implementación del cuarto caso.

Finalmente, la opción default imprime el mensaje de que se elija una opción válida en el menú principal. Del mismo modo que en los anteriores casos, en la parte inferior podemos ver el caso por defecto en la ilustración 88.

```
    default:
        printf("\nEliga una opcción valida\n");
    }
}
```

Ilustración 88 Implementación del caso por defecto.

Resumen del código mostrado

Este código es un editor de texto sencillo que permite crear, editar y eliminar archivos de texto. Este imprime un menú con las siguientes opciones:

1. Hacer un nuevo archivo de texto
2. Editar un archivo de texto ya creado
3. Borrar un archivo de texto
4. Salir

Para esto se tienen las funciones nombreArchivoValido para verificar que el nombre de archivo este correcto, nombreDirArchivoValido para verificar la dirección de un directorio de un archivo, crearArchivo para crear un archivo, modificarArchivo para poder modificar un archivo y eliminarArchivo para eliminar un archivo.

Explicación de la ejecución para la práctica número 8

En la presente imagen (ilustración 89) se aprecia la compilación y ejecución del programa 81, la primera parte de la ejecución en donde se muestra el menú principal con 3 opciones para trabajar con los archivos y una más para salir de la aplicación.

```
ulises@UlisesJE:~$ cd Escritorio/misProgramas
ulises@UlisesJE:~/Escritorio/misProgramas$ gcc -o Programa81 Programa81.c
ulises@UlisesJE:~/Escritorio/misProgramas$ ./Programa81

Bienvenido al editor de texto, escriba la opcción que desea hacer:
1-Hacer un nuevo archivo de texto
2-Editar un archivo de texto ya creado
3-Borrar un archivo de texto
4-Salir
```

Ilustración 89. Menú principal

Si se selecciona la opción de hacer un nuevo archivo de texto, lo primero se solicita es ingresar el nombre que le queremos dar, debe tener terminación .txt para que sea válido. Lo dicho puede apreciarse en la ilustración 90.

```
ulises@UlisesJE:~$ cd Escritorio/misProgramas
ulises@UlisesJE:~/Escritorio/misProgramas$ gcc -o Programa81 Programa81.c
ulises@UlisesJE:~/Escritorio/misProgramas$ ./Programa81

Bienvenido al editor de texto, escriba la opcción que desea hacer:
1-Hacer un nuevo archivo de texto
2-Editar un archivo de texto ya creado
3-Borrar un archivo de texto
4-Salir

1
Escriba nombre de archivo que desea crear
prueba.txt
```

Ilustración 90. Nombre del archivo

Una vez ingresado un nombre valido, aparecerán dos opciones para guardar el archivo, se puede guardar en el directorio actual o podemos guardarlo en un directorio de nuestra preferencia, para esta ocasión crearemos un nuevo directorio llamado archivo, tal como se aprecia en la imagen el nombre del directorio tiene que terminar con / para ser considerado valido. Esto puede apreciarse en la imagen de la parte de abajo (ilustración 91).

```
Eliga una de las siguientes opcciones
1-Hacer un nuevo archivo de texto en el directorio actual
2-Hacer un nuevo archivo de texto en el directorio creado terminado como la sig
uiente forma 'directorio/'
2
Escriba el nombre del directorio
archivos
Escriba dirección de archivo valido
Escriba el nombre del directorio
archivos/
```

Ilustración 91. Nombre del directorio

A continuación, se nos pide escribir el texto que contendrá el archivo que ya ha sido guardado en el directorio y con el nombre asignado, en esta ocasión lo que vamos a guardar en el archivo es la palabra texto, es importante colocar un punto al final del texto a guardar para que funcione correctamente. Lo expuesto anteriormente se puede apreciar en la ilustración 92.

```
archivos/

         Escriba el nombre de texto y presiona '.' para salvarlo

        texto.

Bienvenido al editor de texto, escriba la opcción que desea hacer:
1-Hacer un nuevo archivo de texto
2-Editar un archivo de texto ya creado
3-Borrar un archivo de texto
4-Salir
```

Ilustración 92. Guardando la palabra texto

En la siguiente imagen (ilustración 93) se aprecia que en el directorio actual del programa se creó el directorio archivos que ingresamos en el programa.

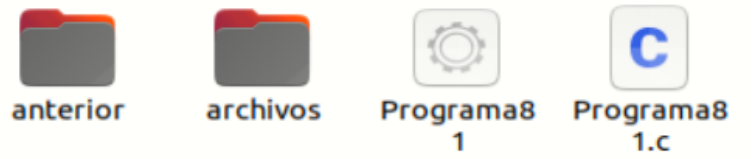


Ilustración 93. Directorio archivos.

Dentro del directorio archivos se creó el archivo de texto con el nombre que ingresamos en la ejecución del programa es decir prueba.txt, lo cual se aprecia claramente en la ilustración 94.

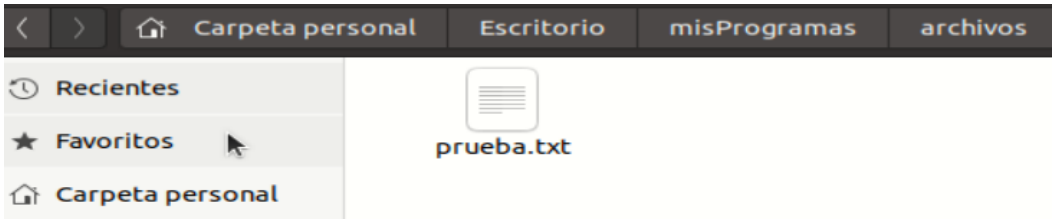


Ilustración 94. Archivo en el directorio

Si abrimos el archivo prueba.txt con un editor de texto podemos apreciar que efectivamente se guardó el texto que ingresamos durante el programa en este caso texto con terminación con ".", justo como se ve en la ilustración 95.

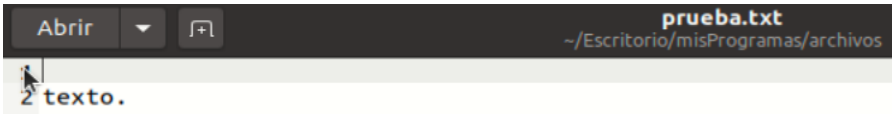


Ilustración 95. Contenido del archivo prueba.txt

Ahora seleccionamos la opción de editar un archivo ya creado desde el menú, entonces nos pedirá ingresar la ruta y el nombre del archivo a editar, aparece entonces el contenido de ese archivo e ingresamos el texto que queremos agregar, en este caso se agregaron las palabras "nuevo texto", del mismo modo que en la creación del texto original se debe poner un punto al final del texto para que se agregue de manera correcta. Lo mencionado se aprecia con claridad en la ilustración 96.

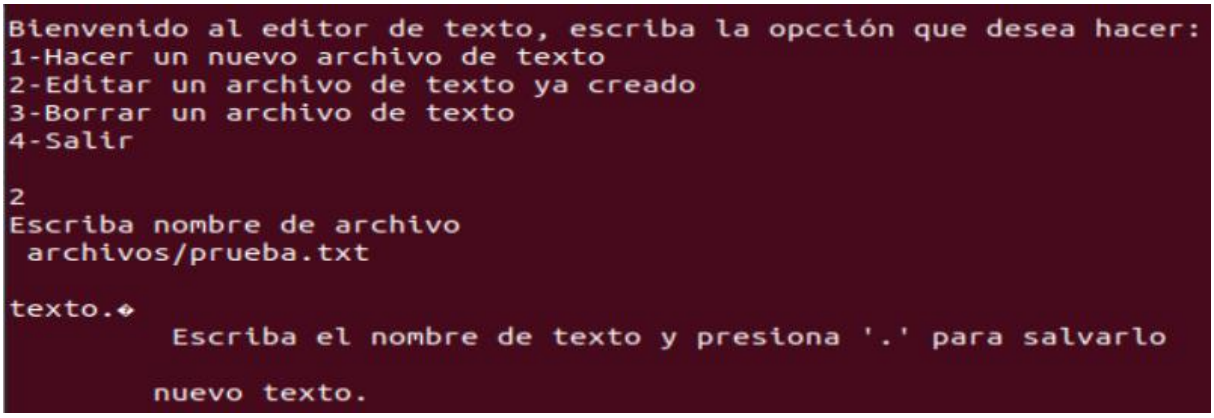


Ilustración 96. Agregando nuevo texto

Ahora abrimos el archivo prueba.txt con un editor de texto y vemos que efectivamente se agregaron las palabras “nuevo texto” al archivo existente. Observamos lo dicho en la ilustración 97.

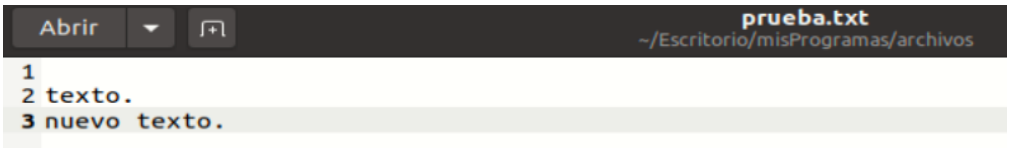


Ilustración 97. Nuevo contenido del archivo prueba.txt

Ahora desde el menú seleccionamos la opción de eliminar un archivo de texto y escribimos la ruta y el nombre del archivo a eliminar, en este caso con el que hemos estado trabajando que es el archivo prueba.txt, lo anterior puede verse en la ilustración 98.

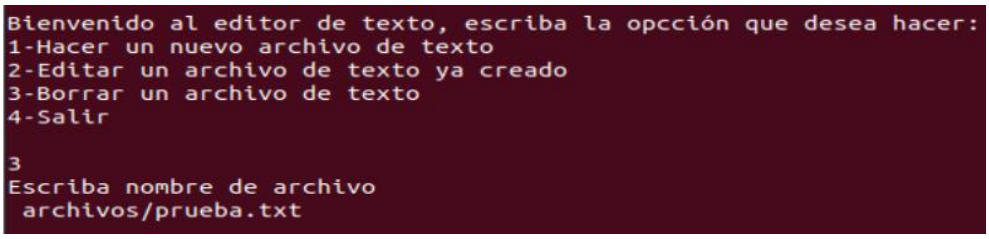


Ilustración 98. Eliminación de un archivo

Ahora abrimos el directorio archivo, como solo teníamos el archivo prueba.txt y se ha eliminado ahora está vacío, tal cual esto puede ser visto en la ilustración 99.

Ilustración 99. Contenido del directorio

Finalmente salimos del programa utilizando la opción desde el menú, como puede apreciarse en la ilustración 100.

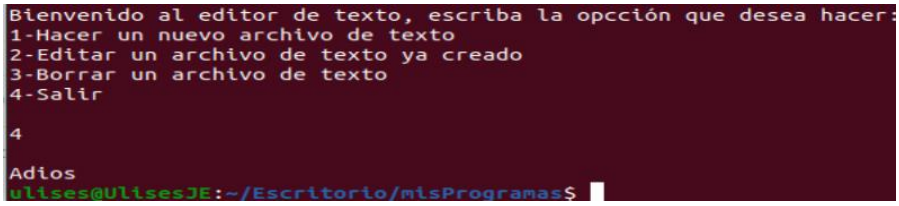


Ilustración 100. Salir del programa

Conclusiones

Chavarría Vázquez Luis Enrique

A lo largo del desarrollo de la práctica me fue posible aprender una barbaridad sobre las múltiples funcionalidades que se tienen tanto en lenguaje de programación C y C++, con lo cual puedo decir que me siento muy satisfecho por este compilado que hemos realizado, porque estoy seguro de que en el momento que yo o mis compañeros lo requiramos, podremos recurrir a la información recabada para poder encontrar soluciones a nuestras implementaciones mucho más eficientes y desde luego efectivas.

Del mismo modo, considero que la práctica me ha permitido poner en práctica aspectos referentes al manejo de archivos dentro del sistema operativo, con lo cual he de decir que me siento con mucha más confianza para poder plantearme poner manos a la obra en implementaciones de software mucho más desafiantes para mis proyectos futuros; cabe destacar que el dinamismo de Ubuntu es genial en términos de todas las posibilidades nos ofrece para el manejo de archivos, su edición e inclusive los permisos con que podemos contar para poder ofrecer soluciones a nuestros clientes de manera mucho más efectiva y sin ningún tipo de limitación al cumplimiento de los requerimiento que estos tengan, con lo cual bajo ningún caso, se afirma que un sistema operativo sea mejor que otro para estos menesteres, pero sin duda a lo largo de las prácticas que he desarrollado con mi equipo, puedo afirmar que me he quedado con un mucho mejor sabor de boca al poder trabajar con Ubuntu y el ecosistema en general.

Si nos remontamos a la parte técnica de la implementación del editor de texto, se puede destacar que durante todo el desarrollo procuramos trabajar de forma muy modular, con la finalidad de agilizar el tiempo de desarrollo y poder iterar en caso de que requiriéramos hacer algun tipo de ajuste o de que tuviéramos problemas en determinada parte de la implementación del editor; la verdad uno de mis aspectos favoritos de poder trabajar las prácticas como se ha venido haciendo a lo largo de este curso, es que hemos podido poner en marcha mecanismo de planeación estratégica para el desarrollo y solución de los problemas planteados en la unidad de aprendizaje, lo cual al final del día considero que, puede representar una ventaja muy grande en términos de experiencia en el campo laboral e inclusive en el terreno de la relaciones, debido a que como equipo nos hemos podido acoplar bien al ritmo de trabajo.

Un aspecto que valoro mucho, es que se pudo aprender los fundamentos del manejo con las llamadas al sistema para poder lidiar con el manejo de directorios, lo cual desde mi punto de vista resulta bastante útil, porque siento que de muy poco sirve hacer programas que ejecutan determinadas rutinas si bajo ningún termino se es capaz de poder tener una interacción mucho más profunda con los directorios y ficheros del sistema, por lo que considero que ahora , teniendo estos conocimientos, se puede tener un mejor panorama y muchas más herramientas para solucionar problemas que se nos vayan presentando en el camino a lo largo de nuestra carrera y sobre todo en el campo laboral, en donde no sabemos a ciencia cierta que tipo de soluciones tengamos que dar a los clientes en determinado momento.

Finalmente, es digno de mencionarse que, los conocimientos que se obtuvieron aquí nos estan permitiendo resolver algunas problemáticas que estábamos teniendo en nuestro proyecto final del sistema de gestión para pacientes atendidos por especialistas de la salud y del mismo modo me ayudó a poder solucionar una problemática que se me presentó en un proyecto que de manera personal estoy elaborando, para poder realizar la gestión efectiva en empresas de archivos legales y para la gestión administrativa de pequeños y medianos negocios; con lo cual puedo aseverar que me hace muy feliz

poder ver como los conocimientos adquiridos en cierto rubro de programación siempre se enlazan con problemas que se presentan y al final del día, prácticas como esta, de la mano con investigaciones e indagaciones en concreto nos potencian tremendamente de forma integral.

Juárez Espinoza Ulises

En esta práctica me familiaricé con un concepto importante en los sistemas operativos, los archivos, comprendí que no son más que un sistema de abstracción que permite almacenar información en el disco y leerla después, aprendí que son de vital importancia ya que con ayuda del sistema de archivos integrado en el sistema operativo nos permiten el manejo de nuestro equipo en gran medida.

Estos archivos tienen diferentes extensiones que hacen que sean diferentes y tenga funcionalidades diferentes, específicamente hablando de la práctica, se trabajó con archivos de texto que entre otras funcionalidades nos permiten almacenar, modificar, y eliminar información o datos a nuestro antojo.

Fue interesante ver la flexibilidad que brinda el sistema operativo Linux que fue en el que se desarrolló la práctica, para permitir las funcionalidades anteriormente mencionas, en mi caso siempre me había limitado a trabajar mis archivos de texto con editores de texto y a crear mis directorios con ayuda de la interfaz gráfica, recientemente ya los había hecho desde la terminal tanto en Linux como en Windows, aunque ya sabía que se podía hacer nunca me había interesado en probarlo por cuenta propia, estoy hablando de programar estas funcionalidades con un lenguaje de programación tal es el caso de C.

La lógica de cómo debía funcionar el programa fue sencilla, ya que no es nada que sepamos hacer de manera manual, solo nos quedó revisar la documentación de como trabajar con archivos desde lenguaje C y claro al final orientarlo al sistema operativo que estamos trabajando, es decir adicionalmente se investigó acerca de las llamadas al sistema para la creación de directorios en Linux, sin duda quede muy satisfecho con la práctica ya que ha sido un repaso de cosas que ya habíamos visto antes, complementándolo con cosas nuevas, me ayudó mucho a proyectar la utilidad que puede tener el lenguaje C de la mano con funcionalidades con archivos y directorios como en esta práctica. Al decir verdad en mi proyecto de equipo estamos trabajando con archivos de texto y esta práctica me dio nuevas ideas y me aclaro muchas otras.

Machorro Vences Ricardo Alberto

Una parte muy fundamental de la programación es el manejo de datos, y aunque ya existen las base de datos muchas veces se tienen que guardar datos con un formato en específico de tal forma que no se pueden guardar de una forma convencional como un tipo de dato primitivo por eso es que existen los archivos. En este caso se utilizó para la práctica los más simple que son los de texto. Afortunadamente en el lenguaje de programación de C tiene ya librerías, métodos y estructuras que pueden manejar archivos de manera sencilla. Aún con esta sencillez para manejar archivos hay que saber como manejarlos por el hecho de que los archivos al final y acabo son como una especie de dato que se tiene que manejar.

En esta práctica aprendí que, si es relativamente sencillo poder manejar archivos, pero hay que poder planificar que se va hacer con estos y ver como usar los diferentes métodos de un lenguaje de programación para manipularlos correctamente. Esto para mí se me complico un poco porque yo verdaderamente porque algunas veces cuando visualizo un proceso o lo simplifico demasiado al grado de que muchas veces partes que yo pensé que eran sencillas les faltan pasos o me voy al otro extremo sobre prensando como hacer una función para realizar algo cuando la solución era simplemente dividir el problema en más pequeños, para al final unirlos en uno solo que sea sencillo.

Afortunadamente por el hecho de que muchas de las acciones relacionadas con archivos ya se han planteado de muchas formas se puede tomar estas como base para poder ver cuál de esta es la que se adapta más a las necesidades que el código, dejando así un mayor tiempo para ver cosas más específicas del código.

En resumen, esta práctica ayudo a ver como tratar archivos en el lenguaje C de una manera correcta y usando los métodos correctos.

Pastrana Torres Victor Norberto

Los archivos son uno de los elementos más importantes en cualquier sistema operativo, las aplicaciones que tiene los archivos son bastas, se utilizan para muchas cosas. Una de las cosas que puede ser similar a los archivos son las bases de datos ya que ambos sirven para garantizar la supervivencia de la información una vez que el proceso finalice, pero una de las ventajas de los archivos es que son más dinámicos de tratar y se pueden utilizar acorde a lo que necesitamos, si solo se necesita cierta información se puede trabajar el archivo para que únicamente se obtenga la información necesaria.

De primera instancia esta práctica me pareció un poco complicada porque pensar en desarrollar un editor de texto por cuenta propia podría parecer algo complicado, de hecho, una de las opciones que tuvimos fue revisar la documentación de Linux acerca de su editor de texto que vine preinstalado pero ese editor ya esta muy completo si lo comparamos con el básico obtenido en la práctica. A pesar de ser un editor de texto básico nos fue de gran ayuda para comprender el uso de los archivos no solo de forma convencional como es solo lectura o escritura si no incluso editar directorios y validar que cada tipo de fichero cumpla con las características adecuadas en lo que respecta al nombre.

Al introducirme al mundo de la programación una de las cosas que más me costó trabajo aprender fue el manejo de archivos, además si mencionamos que para utilizar cualquier archivo necesitamos de un puntero, esto me complicaba más las cosas. Durante el desarrollo de las practicas vimos la aplicación de diferentes tecnologías, una de las que más me llamo la atención fue la implementación de los hilos y de semáforos, en parte eso fue porque nunca había utilizado esas herramientas disponibles en el lenguaje C. El uso de hilos puede mejorar la implementación de un programa, pero también si no está bien desarrollado puede empeorar el comportamiento del programa.

Para terminar, el uso de archivos es importante para la creación de cualquier sistema, no solo de sistemas operativos si no en cualquier tipo de sistema y/o programa que requiera del almacenamiento, análisis y documentación de información.

Bibliografía

[1] udg, «udg,» udg, [En línea]. Available: https://www.udg.co.cu/cmap/sistemas_operativos/sistema_archivo/directorios/directorios.html. [Último acceso: Diciembre 2020].

[2] TANNENBAUM, Sistemas Operativos Modernos, México: Prentice-Hall, 2003.

[3] sotecnology351, «sotecnology351,» sotecnology351, 25 Octubre 2010. [En línea]. Available: http://sotecnology351.blogspot.com/2010/10/sistemas-de-un-directorio-de-un-solo.html. [Último acceso: Diciembre 2020].

[4] STALLINGS, Sistemas Operativos, México: Prentice Hall, 2003.

[5] J. CARRETERO Pérez, Sistemas operativos, México: Mc. Graw-Hill, 2000.

[6] unne, «unne,» unne, [En línea]. Available: http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/SO0.htm. [Último acceso: Diciembre 2020].

[7] ticportal, «ticportal,» ticportal, [En línea]. Available: https://www.ticportal.es/temas/sistema-gestion-documental/que-es-sistema-gestion-documental. [Último acceso: 3 Enero 2021].

[8] definicionabc, «definicionabc,» definicionabc, [En línea]. Available: https://www.definicionabc.com/general/fichero.php. [Último acceso: Enero 2021].

[9] Baúl de Linux, «Baúl de Linux,» [En línea]. Available: https://baulderasec.wordpress.com/programacion/primeros-pasos-con-linux/cambio-de-privilegios-chmod/. [Último acceso: 2021].

[10] zator, «zator,» [En línea]. Available: https://www.zator.com/Cpp/E5_5_2.htm#filength. [Último acceso: 1 Diciembre 2021].

[11] pubs, «pubs.opengroup.org,» pubs, [En línea]. Available: https://pubs.opengroup.org/onlinepubs/009695399/functions/access.html. [Último acceso: 2021].

[12] stackoverflow, «stackoverflow,» [En línea]. Available: https://es.stackoverflow.com/questions/34281/lenguaje-c-en-linux-chdir-devuelve-valor-1-siempre. [Último acceso: 2021].

[13] QNX, «QNX,» [En línea]. Available: http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_lib_ref%2Fc%2Fchsize.html. [Último acceso: 2021].

[14] die.net, «die.net,» [En línea]. Available: https://linux.die.net/man/3/closedir. [Último acceso: Enero 2021].

[15] Microsoft, «Microsoft,» [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/creat-wcreat?view=msvc-160. [Último acceso: Enero 2021].

[16] stackoverflow.com, «stackoverflow.com,» [En línea]. Available: https://stackoverflow.com/questions/9717271/create-temporary-file-in-c-ms-windows-system. [Último acceso: Enero 2021].

[17] Baúl de Linux, «Baúl de Linux,» [En línea]. Available: https://baulderasec.wordpress.com/programacion/programacion-con-linux/3-trabajando-con-los-archivos/acceso-de-bajo-nivel-a-archivos/dup-y-dup2/. [Último acceso: Enero 2021].

[18] w3big, «w3big.com,» w3big, [En línea]. Available: http://www.w3big.com/es/cprogramming/c-function-feof.html. [Último acceso: Enero 2021].

[19] aprendeaprogramar, «aprendeaprogramar.com,» [En línea]. Available: https://www.aprendeaprogramar.com/referencia/view.php?f=fclose&leng=C. [Último acceso: Enero 2021].

[20] pubs, «pubs,» [En línea]. Available: https://pubs.opengroup.org/onlinepubs/009695399/functions/fdopen.html. [Último acceso: 2021].

[21] tutorialspoint, «tutorialspoint,» [En línea]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_ferror.htm. [Último acceso: 2021].

[22] carlospes, «carlospes,» [En línea]. Available: http://www.carlospes.com/curso_de_lenguaje_c/01_11_la_funcion_fflush.php. [Último acceso: Enero 2021].

[23] aprendeaprogramar, «aprendeaprogramar,» aprendeaprogramar, [En línea]. Available: https://www.aprendeaprogramar.com/referencia/view.php?f=fgetc&leng=c. [Último acceso: 2021].

[24] cplusplus, «cplusplus,» cplusplus, [En línea]. Available: http://www.cplusplus.com/reference/cstdio/fgetpos/. [Último acceso: 2021].

[25] aprendeaprogramar, «aprendeaprogramar,» aprendeaprogramar, [En línea]. Available: https://www.aprendeaprogramar.com/referencia/view.php?f=fgets&leng=c. [Último acceso: Diciembre 2020].

[26] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-findclose. [Último acceso: Diciembre 2020].

[27] msdn, «msdn,» msdn, [En línea]. Available: https://social.msdn.microsoft.com/Forums/vstudio/en-US/a25f92f5-4a9f-4b5b-931a-a6750733fc95/how-to-use-findfirst-and-findnext-in-c?forum=vcgeneral. [Último acceso: Diciembre 2020].

[28] opengroup, «opengroup,» opengroup, [En línea]. Available: https://pubs.opengroup.org/onlinepubs/009695399/functions/fileno.html. [Último acceso: Diciembre 2020].

[29] embarcadero, «embarcadero,» embarcadero, [En línea]. Available: http://docwiki.embarcadero.com/RADStudio/Sydney/en/Fnsplit,_wfnsplit. [Último acceso: Diciembre 2020].

[30] wikibooks, «wikibooks,» wikibooks, [En línea]. Available: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_de_archivos. [Último acceso: Diciembre 2020].

[31] aprendeaprogramar, «aprendeaprogramar,» aprendeaprogramar, [En línea]. Available: https://www.aprendeaprogramar.com/referencia/view.php?f=fprintf&leng=C. [Último acceso: 2021].

[32] tutorialspoint, «tutorialspoint,» tutorialspoint, [En línea]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_fputs.htm. [Último acceso: Diciembre 2020].

[33] aprenderaprogramar, «aprenderaprogramar,» aprenderaprogramar, [En línea]. Available: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=936:escribir-guardar-datos-en-ficheros-o-archivos-en-lenguaje-c-fputc-putc-fputs-fprintf-ejemplos-cu00537f&catid=82&Itemid=210. [Último acceso: Diciembre 2020].

[34] sourceforge, «sourceforge,» sourceforge, [En línea]. Available: http://cdiv.sourceforge.net/cdivhlp/0225.htm. [Último acceso: 26 Diciembre 2020].

[35] tutorialspoint, «tutorialspoint,» tutorialspoint, [En línea]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_freopen.htm. [Último acceso: Diciembre 2020].

[36] riptutorial, «riptutorial,» riptutorial, [En línea]. Available: https://riptutorial.com/es/c/example/25365/fscanf---. [Último acceso: Diciembre 2020].

[37] aprendeaprogramar, «aprendeaprogramar,» aprendeaprogramar, [En línea]. Available: https://www.aprendeaprogramar.com/referencia/view.php?f=fseek&leng=c. [Último acceso: 2 Enero 2021].

[38] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-160. [Último acceso: Diciembre 2020].

[39] w3big, «w3big,» w3big, [En línea]. Available: http://www.w3big.com/es/cprogramming/c-function-fsetpos.html. [Último acceso: Diciembre 2020].

[40] opengroup, «opengroup,» opengroup, [En línea]. Available: https://pubs.opengroup.org/onlinepubs/009696699/functions/fstat.html. [Último acceso: Enero 2021].

[41] sourceforge, «sourceforge,» sourceforge, [En línea]. Available: http://cdiv.sourceforge.net/cdivhlp/0228.htm. [Último acceso: Diciembre 2020].

[42] oreilly, «oreilly,» oreilly, [En línea]. Available: https://www.oreilly.com/library/view/c-cookbook/0596007612/ch10s16.html. [Último acceso: 4 Enero 2021].

[43] sourceforge, «sourceforge,» sourceforge, [En línea]. Available: http://cdiv.sourceforge.net/cdivhlp/0226.htm#:~:text=La%20funci%C3%B3n%20fwrite()%20requiere,memoria%20que%20ocupan%20dichos%20datos.. [Último acceso: 6 Diciembre 2020].

[44] ubuntu, «ubuntu,» ubuntu, [En línea]. Available: http://manpages.ubuntu.com/manpages/bionic/es/man3/gets.3.html. [Último acceso: 26 Diciembre 2020].

[45] baidu, «baidu,» baidu, [En línea]. Available: https://wenku.baidu.com/view/bed7ef096ad97f192279168884868762caaebba0.html?re=view. [Último acceso: 30 Diciembre 2020].

[46] stackoverflow, «stackoverflow,» stackoverflow, [En línea]. Available: https://stackoverflow.com/questions/298510/how-to-get-the-current-directory-in-a-c-program. [Último acceso: 30 Diciembre 2020].

[47] stackoverflow, «stackoverflow,» stackoverflow, [En línea]. Available: https://stackoverflow.com/questions/60666200/how-can-get-disk-usage-in-linux-using-c-program. [Último acceso: 2 Enero 2021].

[48] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/getdrive?view=msvc-160. [Último acceso: Enero 2021].

[49] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/locking?view=msvc-160. [Último acceso: Diciembre 2020].

[50] sopaulpgc, «sopaulpgc,» sopaulpgc, [En línea]. Available: http://sopa.dis.ulpgc.es/prog_c/FICHER.HTM. [Último acceso: Enero 2021].

[51] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/makepath-s-wmakepath-s?view=msvc-160. [Último acceso: Diciembre 2020].

[52] opengroup, «opengroup,» opengroup, [En línea]. Available: https://pubs.opengroup.org/onlinepubs/009695399/functions/mkdir.html. [Último acceso: Enero 2021].

[53] codewiki, «codewiki,» codewiki, [En línea]. Available: http://codewiki.wikidot.com/c:system-calls:open. [Último acceso: Diciembre 2020].

[54] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/perror-wperror?view=msvc-160. [Último acceso: Enero 2021].

[55] aprendeaprogramar, «aprendeaprogramar,» aprendeaprogramar, [En línea]. Available: https://www.aprendeaprogramar.com/referencia/view.php?f=putchar&leng=c. [Último acceso: Diciembre 2020].

[56] geeksforgeeks, «geeksforgeeks,» geeksforgeeks, [En línea]. Available: https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/. [Último acceso: 3 Enero 2021].

[57] tutorialspoint, «tutorialspoint,» tutorialspoint, [En línea]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_remove.htm. [Último acceso: Diciembre 2020].

[58] techonthenet, «techonthenet,» techonthenet, [En línea]. Available: https://www.techonthenet.com/c_language/standard_library_functions/stdio_h/rename.php. [Último acceso: 4 Enero 2021].

[59] programacionwebs, «programacionwebs,» programacionwebs, [En línea]. Available: https://www.programacionwebs.com/c/funcion-rewind-en-c/. [Último acceso: Enero 2021].

[60] «opengroup,» opengroup, [En línea]. Available: https://pubs.opengroup.org/onlinepubs/007904875/functions/rmdir.html. [Último acceso: Enero 2021].

[61] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/posix-rmtmp?view=msvc-160. [Último acceso: 28 Diciembre 2020].

[62] www.c-cpp.ru, «www.c-cpp.ru,» www.c-cpp.ru, [En línea]. Available: http://www.c-cpp.ru/content/creat-rtlcreat-creatnew-creattemp. [Último acceso: 29 Diciembre 2020].

[63] www.c-cpp.ru, «www.c-cpp.ru,» www.c-cpp.ru, [En línea]. Available: http://www.c-cpp.ru/content/open-rtlopen. [Último acceso: Enero 2021].

[64] appmethod, «appmethod,» appmethod, [En línea]. Available: http://docwiki.appmethod.com/appmethod/1.14/topics/en/Rtl_write. [Último acceso: Diciembre 2020].

[65] www.c-cpp.ru, «www.c-cpp.ru,» www.c-cpp.ru, [En línea]. Available: http://www.c-cpp.ru/content/write-rtlwrite. [Último acceso: Enero 2021].

[66] embarcadero, «embarcadero,» embarcadero, [En línea]. Available: http://docwiki.embarcadero.com/RADStudio/Sydney/en/Searchenv,_wsearchenv. [Último acceso: Diciembre 2020].

[67] IBM, «IBM,» IBM, [En línea]. Available: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_74/rtref/setbuf.htm. [Último acceso: Diciembre 2020].

[68] itlnet, «itlnet,» itlnet, [En línea]. Available: https://www.itlnet.net/programming/program/Reference/tc/ng4cb17.html. [Último acceso: 29 Diciembre 2020].

[69] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/posix-setmode?view=msvc-160. [Último acceso: Enero 2021].

[70] tutorialspoint, «tutorialspoint,» tutorialspoint, [En línea]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_setvbuf.htm. [Último acceso: Diciembre 2020].

[71] codeforwin, «codeforwin,» codeforwin, [En línea]. Available: https://codeforwin.org/2018/03/c-program-find-file-properties-using-stat-function.html#:~:text=stat()%20function%20in%20C&text=stat()%20function%20is%20used,character%20pointing%20to%20file%20path.. [Último acceso: 3 Enero 2021].

[72] tutorialspoint, «tutorialspoint,» tutorialspoint, [En línea]. Available: https://www.tutorialspoint.com/system-function-in-c-cplusplus#:~:text=The%20system()%20function%20is,after%20it%20has%20been%20completed.. [Último acceso: Enero 2021].

[73] geeksforgeeks, «geeksforgeeks,» geeksforgeeks, [En línea]. Available: https://www.geeksforgeeks.org/tmpnam-function-c/. [Último acceso: Enero 2021].

[74] geeksforgeeks, «geeksforgeeks,» geeksforgeeks, [En línea]. Available: https://www.geeksforgeeks.org/tmpfile-function-c/. [Último acceso: 27 Diciembre 2020].

[75] stackoverflow, «stackoverflow,» stackoverflow, [En línea]. Available: https://stackoverflow.com/questions/6198623/when-is-umask-useful. [Último acceso: Diciembre 2020].

[76] microsoft, «microsoft,» microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/unlink-wunlink?view=msvc-160. [Último acceso: Enero 2021].

[77] «cplusplus,» cplusplus, [En línea]. Available: http://www.cplusplus.com/reference/mutex/mutex/unlock/. [Último acceso: Diciembre 2020].

[78] «IBM,» IBM, [En línea]. Available: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.bpxbd00/rtuti.htm. [Último acceso: Diciembre 2020].