Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Operativos

“Práctica 2. Introducción al sistema operativo Linux y Windows (2)”

Grupo: 2CM9

Integrantes:

- Martínez Coronel Brayan Yosafat.
- Monteros Cervantes Miguel Angel.
- Ramírez Olvera Guillermo.
- Sánchez Méndez Edmundo Josue.

Profesor: Cortés Galicia Jorge

ESCOM

Práctica 2. Introducción al sistema operativo Linux y Windows (2)

Introducción

Como se ha visto en las lecturas, el sistema operativo responde mediante rutinas a las interrupciones, las direcciones de estas rutinas (que son fragmentos de código) está en las primeras celdas de la memoria principal, para conveniencia y eficiencia del sistema operativo. Como se vio en la teoría, las interrupciones pueden provenir de diversos lugares, como el propio sistema operativo, los dispositivos de E / S, o código que se desea ejecutar desde el modo usuario para el modo kernel.

Una llamada al sistema es la forma en la que un programa de computadora solicita un servicio del núcleo del sistema operativo en el que se ejecuta, también es una forma de que los programas interactúen con el sistema operativo. Un programa de computadora realiza una llamada al sistema cuando realiza una solicitud al kernel del sistema operativo. La llamada al sistema proporciona los servicios del sistema operativo a los programas de usuario a través de la interfaz de programa de aplicación (API). Proporciona una interfaz entre un proceso y un sistema operativo para permitir que los procesos de nivel de usuario soliciten servicios del sistema operativo. Las llamadas al sistema son los únicos puntos de entrada al sistema del núcleo. Todos los programas que necesitan recursos deben utilizar llamadas al sistema.

Servicios proporcionados por llamadas al sistema:

- Creación y gestión de procesos.
- Gestión de la memoria principal.
- Gestión de acceso a archivos, directorio y sistema de archivos.
- Manejo de dispositivos (E / S).
- Protección
- Redes, etc.

Tipos de llamadas al sistema: hay 5 categorías diferentes de llamadas al sistema:

- Control de procesos: finalizar, abortar, crear, terminar, asignar y liberar memoria.
- Gestión de archivos: crear, abrir, cerrar, eliminar, leer archivos, etc.
- Gestión de dispositivos.
- Mantenimiento de información.
- Comunicación.

En esta practica veremos llamadas al sistema en los dos sistemas operativos que hemos estado ocupando, los cuales son Linux (distribución Ubuntu) y Windows, veremos como tienen llamadas al sistema con la misma funcionalidad y evidentemente con diferente forma de invocación y diferentes parámetros, aunque veremos que no todas las llamadas al sistema en Linux hay una homologa en Windows.

1. Competencias.

El alumno aprende a familiarizarse con los sistemas operativos Linux y Windows (en su funcionalidad básica), mediante el desarrollo de programas bajo el lenguaje C para la invocación de llamadas al sistema propias de cada sistema operativo.

2. Desarrollo.

2.1. Compatibilidad de archivos

2.1.1. Creación de documentos y almacenamiento es la memoria USB

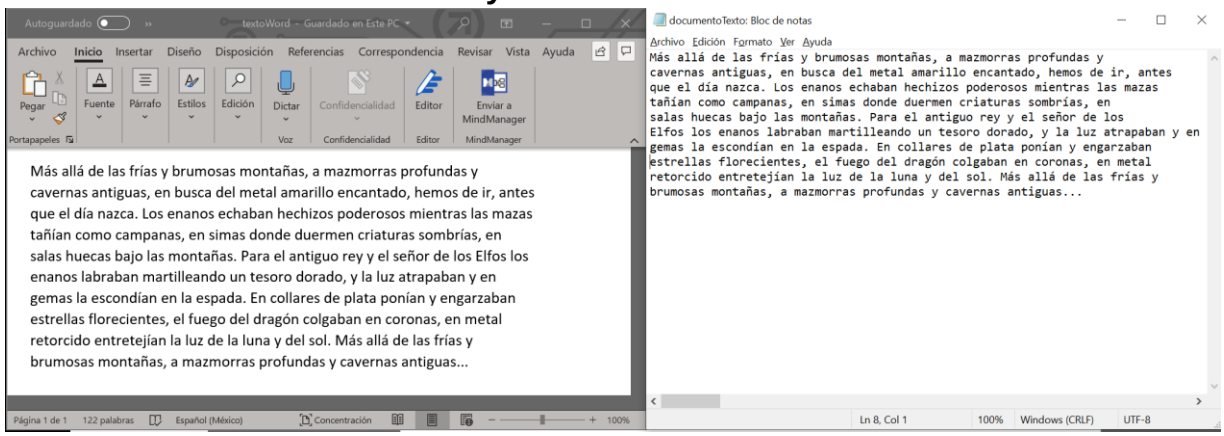


Figura 1: Creación de documentos en Word y en bloc de notas en texto plano.

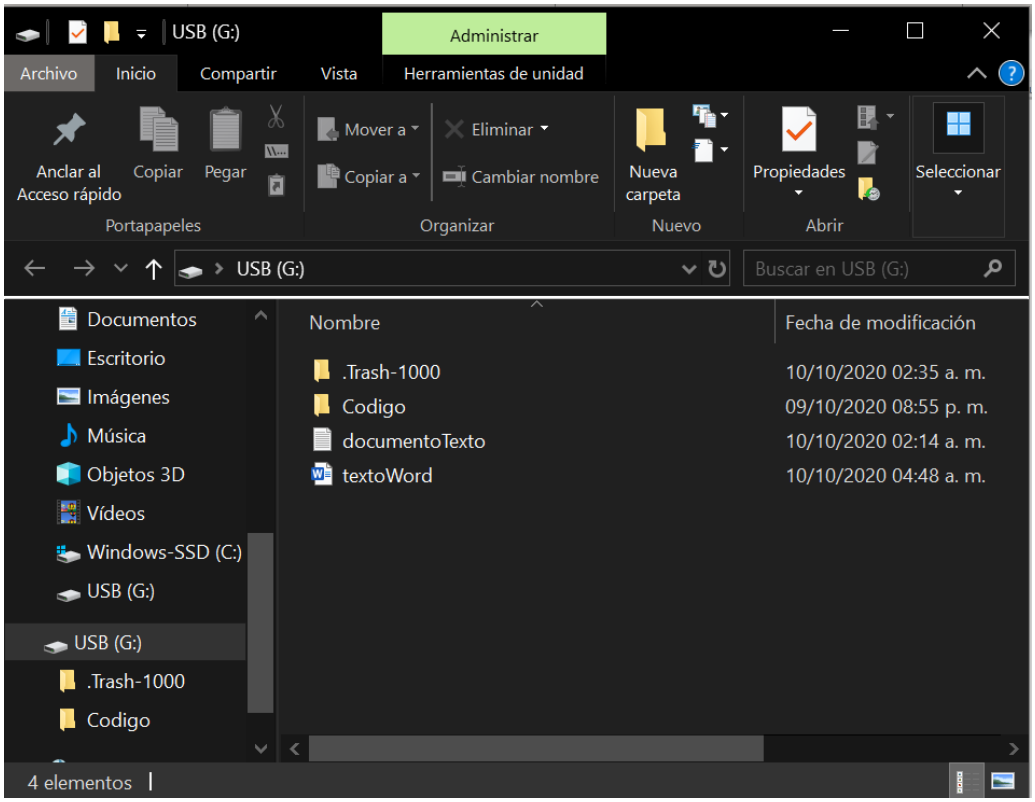


Figura 2: Archivos guardados correctamente en la memoria USB.

2.1.2. Inicio y montaje de la USB en Linux

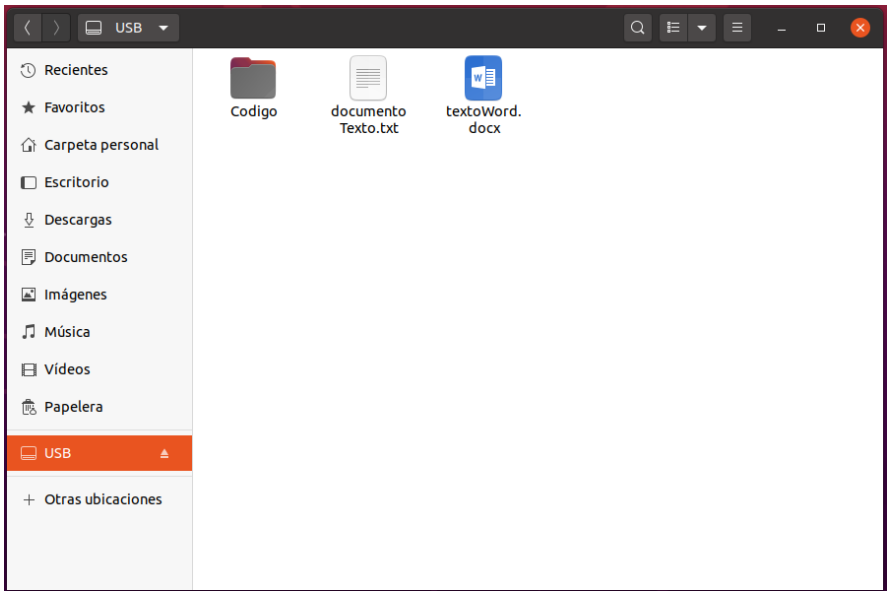


Figura 3: La memoria USB se montó de forma automática al conectarla.

2.1.3. Edición de los documentos creados en Windows en Linux

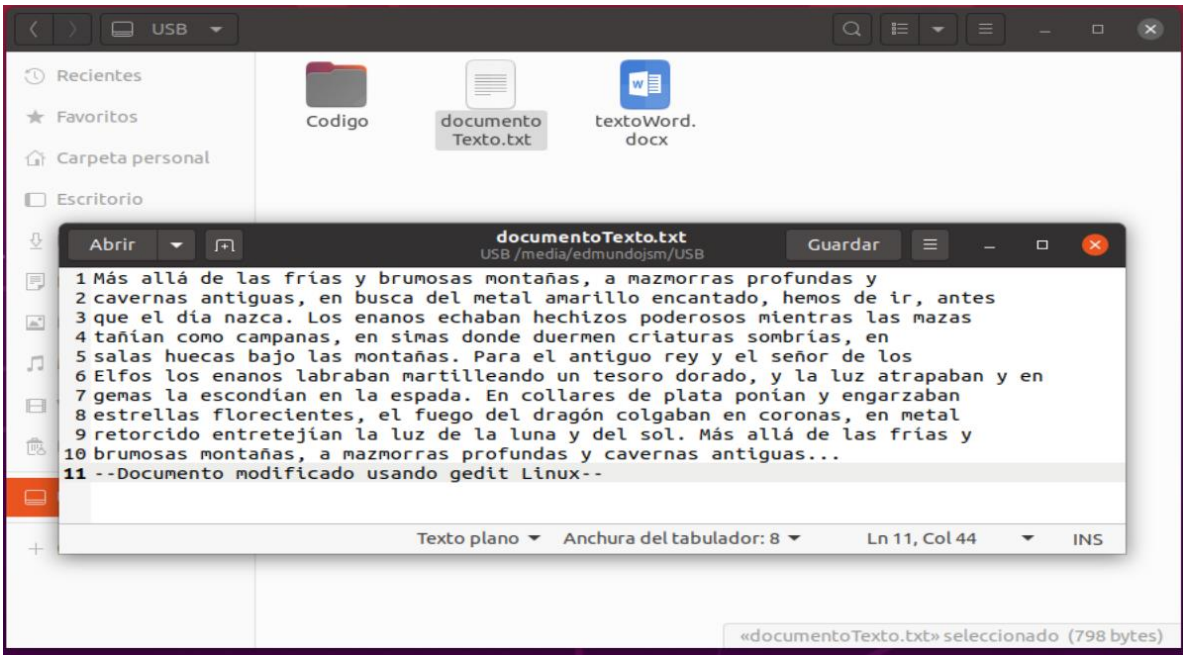


Figura 4: Al abrir el documento de texto plano (con extensión .txt) con ***gedit*** podemos editarlo normalmente y sin ningún tipo de errores.

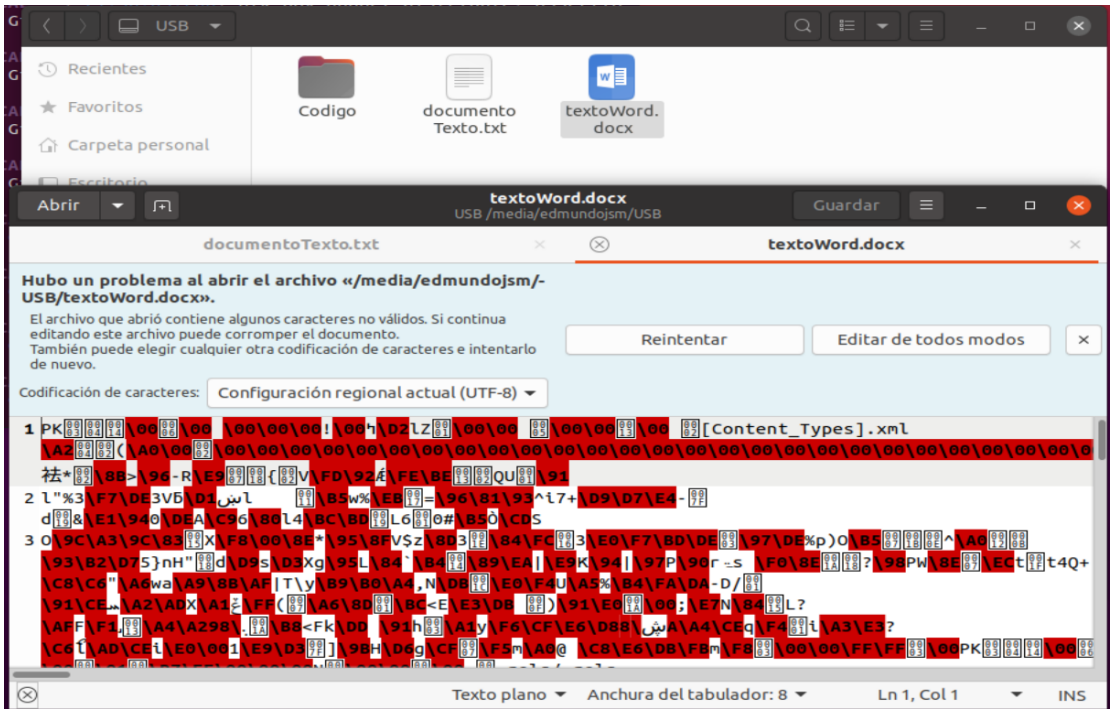


Figura 5: Por el contrario, al intentar abrir el documento creado con Word en ***gedit*** nos lanza una advertencia en la cual nos dice que no hay caracteres válidos, aun así, nos deja editar el archivo corriendo el riesgo de perder la información.

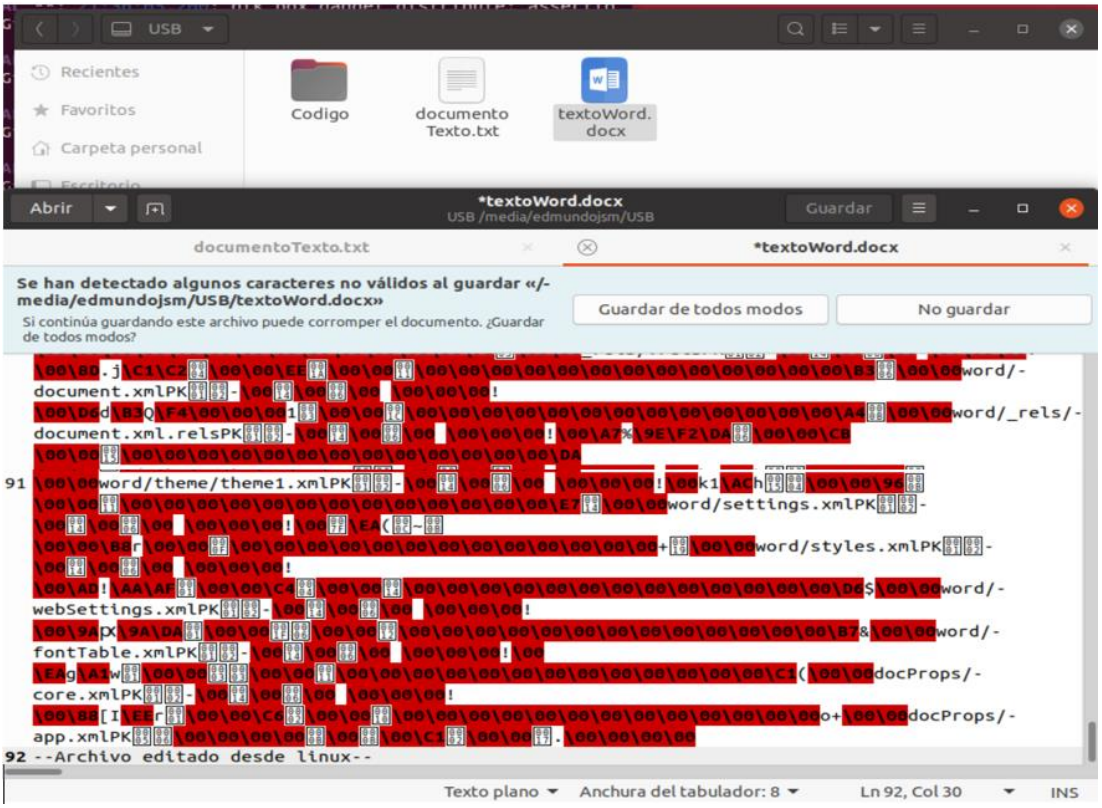


Figura 6: Al editarlo y tratar de guardarlo nos vuelve a lanzar la advertencia de que podemos corromper el archivo, aun así, aceptamos para poder ver qué pasa en Windows.

2.1.4. Visualización de resultados en Windows

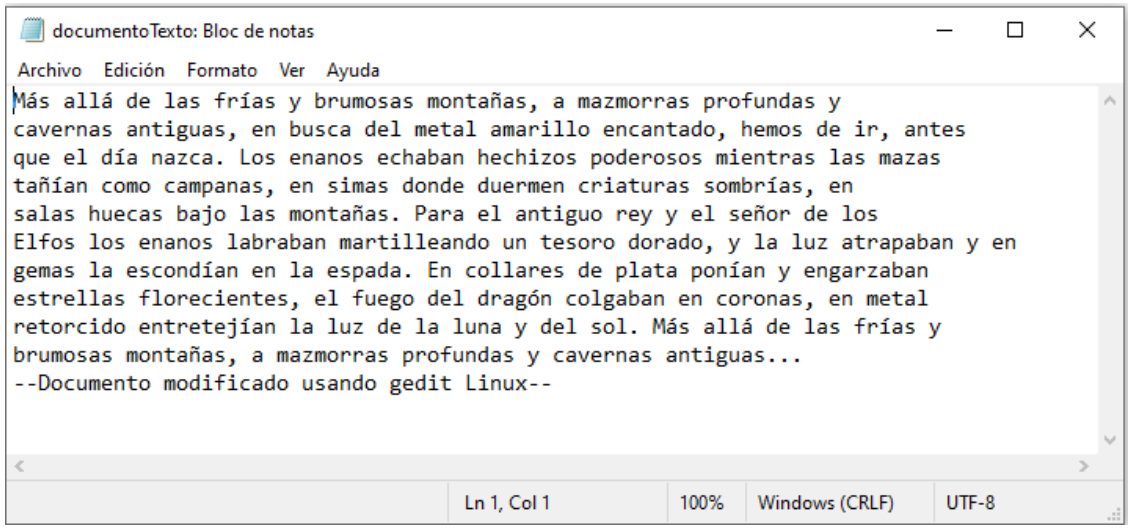


Figura 7: De vuelta en Windows, vemos que el archivo de texto plano (con extensión .txt) la modificación realizada en Linux aparece exactamente igual a lo realizado en Windows, es visible ya que ***gedit*** en ningún momento nos mostró alguna advertencia.

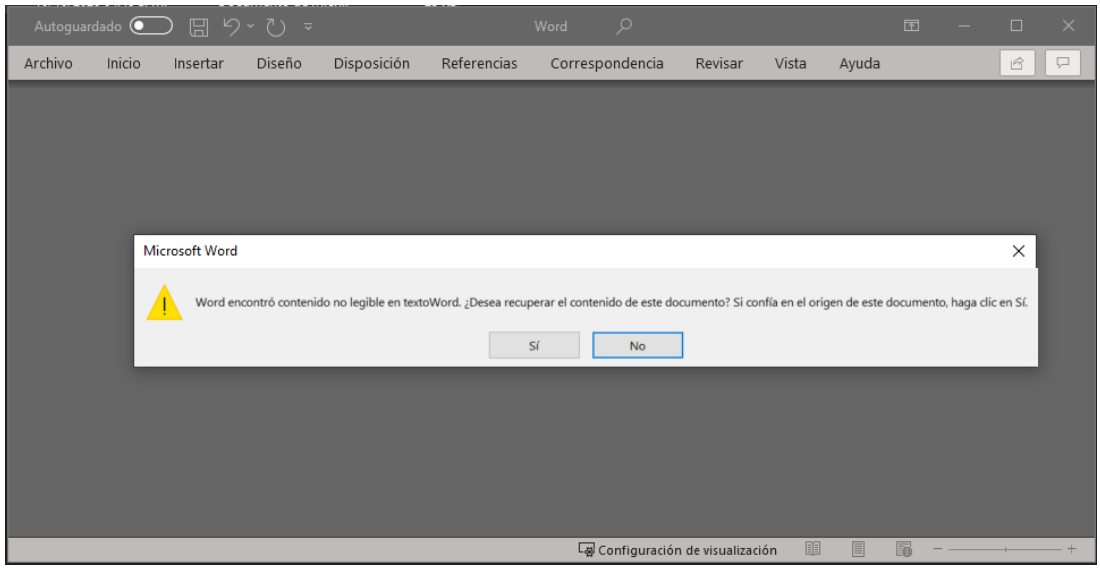


Figura 8: Al momento de querer abrir el archivo en Word nos aparece esta advertencia, la cual nos indica que el documento fue corrompido, esto debido a la advertencia que ***gedit*** nos indicó.

2.2. Llamadas al sistema en el sistema operativo Linux

2.2.1. open

```
01. #include <sys/types.h>
02. #include <sys/stat.h>
03. #include <fcntl.h>
04.
05. int open(const char *pathname, int flags);
06. int open(const char *pathname, int flags, mode_t mode);
```

La llamada al sistema open() abre el archivo especificado por su ruta de acceso. Devuelve un descriptor de dicho archivo.

- **const char *pathname**: La ruta absoluta o relativa al archivo que va a ser abierto.
- **int flags**: Una lista de valores que determinan de qué forma se va a abrir el archivo, separados mediante un OR bit a bit. Los valores posibles son:
    - **O_RDONLY**: Solo lectura.
    - **O_WRONLY**: Solo escritura.
    - **O_RDWR**: Lectura y escritura.
    - **O_APPEND**: Añadir nuevo contenido al final del archivo.
    - **O_TRUNC**: Borrar todo el contenido del archivo al abrirlo.
    - **O_CREAT**: Si el archivo no existe, se crea.
    - **O_EXCL**: Combinada con la opción **O_CREAT**, se asegura de que realmente se cree el archivo. Si ya existe, la llamada fallara.
    - **O_DIRECTORY**: Si *pathname* no es un directorio, la apertura falla. Esta bandera se agregó para evitar la denegación de problemas de servicio si se llama a **opendir**.
    - **O_NOFOLLOW**: Si el componente final (es decir, el nombre de base) de la ruta es un enlace simbólico, entonces la apertura fallara. Los enlaces simbólicos en componentes anteriores del nombre de ruta todavía se seguirán.
    - **O_TMPFILE**: Crea un archivo regular temporal sin nombre. El argumento del nombre de ruta especifica un directorio; se creará un nodo índice sin nombre en el sistema de archivos de ese directorio. Cualquier cosa escrita al archivo resultante se perderá cuando el último descriptor de archivo sea cerrado, a menos que el archivo tenga un nombre.
- **mode_t mode**: Una lista de valores que determinan los permisos del archivo en caso de que se cree, separados mediante un OR bit a bit. Los nueve valores posibles son **S_Iou**, donde:
    - **o** representa la operación, puede ser **R** (lectura), **W** (escritura) o **X** (ejecución).
    - **u** representa a quien aplica el permiso, puede ser USR (propietario), GRP (grupo) o OTH (otros usuarios).
- El valor devuelto por **open** es el descriptor para el archivo, el cual es el mínimo entero positivo disponible. Si se devuelve un numero negativo, hubo un error al abrir el archivo.

2.2.2. close

```
01. #include <unistd.h>
02.
03. int close(int fd);
```

La llamada al sistema **close()** se usa para cerrar un descriptor de archivo abierto, de tal forma que ya no haga referencia a ningún archivo y pueda ser reutilizado.

- **int fd**: El descriptor de archivo a ser cerrado.
- El valor devuelto por **close** es 0 en caso de éxito, -1 en caso de falla.

2.2.3. read

```
01. #include <unistd.h>
02.
03. ssize_t read(int fd, void *buf, size_t count);
```

La llamada al sistema **read()** es una llamada que lee una cantidad determinada de bytes de un descriptor de archivo en un buffer.

- **int fd**: El descriptor de archivo de donde se va a leer el contenido. Se puede usar uno obtenido desde open o usar 0, 1 o 2 para usar la entrada estándar, la salida estándar o la salida estándar de error, respectivamente.
- **void *buf**: Un arreglo de caracteres en donde el contenido leído se va a guardar.
- **size t count**: El número de bytes a leer antes de truncar los datos. Si el tamaño de lo que se va a leer es menor que **count**, se guarda todo en el buffer.
- El valor devuelto por **read** es el número de bytes que fueron leídos. Si es negativo, ocurrió un error. Si es menor a **count**, no necesariamente ocurrió un error, pues no todos los bytes solicitados estaban disponibles.

2.2.4. write

```
01. #include <unistd.h>
02.
03. ssize_t write(int fd, const void *buf, size_t count);
```

La llamada al sistema **write()** es una llamada que escribe una cantidad determinada de bytes proveniente de un buffer en un descriptor de archivo.

- **int fd**: El descriptor de archivo a donde se va a escribir el contenido. Se puede usar uno obtenido desde open o usar 0, 1 o 2 para usar la entrada estándar, la salida estándar o la salida estándar de error, respectivamente.
- **const void *buf**: Un apuntador al buffer de al menos **count** bytes, el cual será escrito al archivo.
- **size t count**: El número de bytes a escribir. Si es menor que el tamaño del buffer se trunca.
- El valor devuelto por **write** es el número de bytes que fueron escritos. Si es negativo, ocurrió un error. Si es menor a **count**, no necesariamente ocurrió un error, pues puede que el almacenamiento ya estaba lleno, por ejemplo.

2.2.5. creat

```
01. #include <sys/types.h>
02. #include <sys/stat.h>
03. #include <fcntl.h>
04.
05. int creat(const char *pathname, mode_t mode);
```

creat() es una llamada al sistema que permite crear y abrir un nuevo archivo. Por lo tanto, es equivalente a llamar a **open()** con los flags **O_CREAT | O_WRONLY | O_TRUNC**. El valor devuelto es el mismo que el de **open()**.

2.2.6. lseek

```
01. #include <sys/types.h>
02. #include <unistd.h>
03.
04. off_t lseek(int fd, off_t offset, int whence);
```

La llamada al sistema **lseek()** es una llamada que cambia la posición del puntero de lectura/escritura de un descriptor de archivo. Dicha posición puede ser absoluta o relativa.

- **int fd**: El descriptor de archivo del cual se va a mover su puntero.
- **off_t offset**: El desplazamiento del puntero en bytes.
- **int whence**: La forma en la cual el argumento offset se va a interpretar. Hay tres posibles valores:
    - **SEEK_SET**: El offset se mide en términos absolutos, es decir, el puntero se coloca en la posición indicada respecto al comienzo del archivo.
    - **SEEK_CUR**: El offset se mide relativo a la posición actual del puntero, es decir, a la oposición actual le sumamos el valor indicado.
    - **SEEK_SET**: El offset se mide relativo al fin del archivo, es decir, retrocedemos el valor indicado comenzando desde el fin del archivo.
- El valor devuelto por **lseek** es la nueva posición del puntero respecto al comienzo del archivo. Si es negativo, ocurrió un error al cambiar la posición.

2.2.7. access

```
01. #include <unistd.h>
02.
03. int access(const char *pathname, int mode);
```

La llamada al sistema **access()** es una llamada que nos permite saber si el proceso actual puede acceder a un archivo especifico de un modo especifico.

- **const char *pathname**: La ruta absoluta o relativa al archivo que va a ser verificado.
- **int mode**: Especifica el o los tests de accesibilidad que se van a revisar, separados por un OR bit a bit. Los valores posibles son:
    - **F_OK**: Revisa la existencia del archivo.
    - **R_OK**: Revisa si tenemos acceso de lectura al archivo.
    - **W_OK**: Revisa si tenemos acceso de escritura al archivo.
    - **X_OK**: Revisa si tenemos acceso de ejecución al archivo.
- El valor devuelto por **access** es 0 si todos los tests solicitados fueron satisfactorios, y -1 si al menos uno de los tests fallo.

2.2.8. stat

```
01. #include <sys/types.h>
02. #include <sys/stat.h>
03. #include <unistd.h>
04.
05. int stat(const char *pathname, struct stat *statbuf);
06. int fstat(int fd, struct stat *statbuf);
07. int lstat(const char *pathname, struct stat *statbuf);
```

Las llamadas al sistema **stat()**, **lstat()** y **fstat()** son llamadas que devuelven información de un archivo dada su ruta de acceso. La única diferencia entre las tres es que **stat()** recibe una ruta absoluta o relativa, **fstat()** recibe un descriptor de archivo previamente creado y **lstat()** en el caso de que el *pathname* es un *Symbolic Link* regresara información sobre el link y no del archivo que el link refiera.

- **const char *pathname**: La ruta absoluta o relativa del archivo a obtener información.
- **int fd**: El descriptor de archivo a obtener información.
- **struct stat *statbuf**: Un apuntador a una estructura de tipo **stat** en donde la información sobre el archivo solicitado se guardará. Sus campos son:
    - **st_mode**: Permisos actuales del archivo.
    - **st_ino**: El "inode" del archivo.
    - **st_dev**: El dispositivo en el cual se encuentra el archivo.
    - **st_uid**: El ID de usuario del archivo.
    - **st_gid**: El ID de grupo del archivo.
    - **st_atime**: La fecha más reciente en la que el archivo fue accedido.
    - **st_ctime**: La fecha más reciente en la que se modificaron los permisos del archivo.
    - **st_mtime**: La fecha más reciente en la que se modificó el contenido del archivo.
    - **st_nlink**: El número de enlaces existentes hacia este archivo.
    - **st_size**: El tamaño del archivo en bytes.
- El valor devuelto por **stat** es 0 en caso de éxito, -1 en caso de error.

2.2.9. chmod

```
01. #include <sys/stat.h>
02.
03. int chmod(const char *pathname, mode_t mode);
04. int fchmod(int fd, mode_t mode);
```

Las llamadas al sistema **chmod()** y **fchmod()** sirven para modificar los permisos de un archivo dado por su ruta de acceso o por su descriptor de archivo.

- **const char *pathname**: La ruta absoluta o relativa del archivo al que se le modificaran los permisos.
- **int fd**: El descriptor de archivo al que se le modificaran los permisos.
- **mode_t mode**: Representa los 9 bits que indican los permisos. Los tres bits menos significativos indican los permisos para los "otros usuarios", los tres

que le siguen para el “grupo” y los tres que le siguen para el propietario. Por lo tanto, podemos combinar cualquiera de los siguientes 9 valores con operaciones OR bit a bit (los valores están en octal):

- **S_IRUSR**: 400, leer por propietario.
- **S_IWUSR**: 200, escribir por propietario.
- **S_IXUSR**: 100, ejecutar por propietario.
- **S_IRGRP**: 040, leer por grupo.
- **S_IWGRP**: 020, escribir por grupo.
- **S_IXGRP**: 010, ejecutar por grupo.
- **S_IROTH**: 004, leer por otros.
- **S_IWOTH**: 002, escribir por otros.
- **S_IXOTH**: 001, ejecutar por otros.

- El valor devuelto por **chmod** es 0 en caso de éxito, -1 en caso de error.

2.2.10. chown

```
01. #include <unistd.h>
02.
03. int chown(const char *pathname, uid_t owner, gid_t group);
04. int fchown(int fd, uid_t owner, gid_t group);
05. int lchown(const char *pathname, uid_t owner, gid_t group);
```

Las llamadas al sistema **chown()**, **lchown()** y **fchown()** sirven para modificar el propietario y el grupo de un archivo dado por su ruta de acceso o por su descriptor de archivo, las diferencias residen en que **chown()** cambia la propiedad del archivo especificado por el pathname que es desreferenciado si es un *symbolic link,* **fchown()** cambia la propiedad del archivo del archivo referenciado a ser abierto en el descriptor de archivos fd y **lchown()** es como **chown()** pero no desreferencia *symbolic links*.

- **const char *pathname**: La ruta absoluta o relativa del archivo al que se le modificaran los permisos.
- **int fd**: El descriptor de archivo al que se le modificaran los permisos.
- **uid_t owner**: ID del propietario, -1 para no cambiar.
- **gid_t group**: ID del grupo, -1 para no cambiar.
- El valor devuelto por **chmod** es 0 en caso de éxito, -1 en caso de error.

2.2.11. fcntl

```
01. #include <unistd.h>
02. #include <fcntl.h>
03.
04. int fcntl(int fd, int cmd, ... /* arg */ );
```

La llamada al sistema **fcntl()** realiza una operación sobre un descriptor de archivo.

2.2.12. opendir

```
01. #include <sys/types.h>
02. #include <dirent.h>
03.
04. DIR *opendir(const char *name);
05. DIR *fdopendir(int fd);
```

La llamada al sistema **opendir()** abre una secuencia para obtener información sobre una carpeta dada.

- **const char *name**: Es la ruta absoluta o relativa a la carpeta.
- **int fd**: Descriptor de archivo de la carpeta.
- El valor devuelto por **opendir** es un apuntador a la secuencia de la carpeta. En caso de error, se devuelve NULL.

2.2.13. readdir

```
01. #include <dirent.h>
02.
03. struct dirent *readdir(DIR *dirp);
```

La llamada al sistema **readdir()** devuelve información sobre una carpeta, dado que ya se abrió una secuencia de dicha carpeta con la llamada anterior.

- **DIR *dirp**: Es un apuntador a una secuencia de la carpeta de interés.
- El valor devuelto por **readdir** es un apuntador a una estructura **dirent** que representa la siguiente entrada en la carpeta. Devuelve NULL si ya hemos llegado al final de la secuencia o si ocurrió algún error. Los campos de la estructura **dirent** son:
    - **Ino_t d_ino**: El número del “inodo” del archivo.
    - **off_t d_off**: La ubicación del elemento actual en la secuencia.
    - **unsigned short d_reclen**: El tamaño en bytes del elemento actual.
    - **unsigned char d_type**: Indica el tipo del elemento actual, sus valores posibles son:
        - **DT_BLK**: Dispositivo de bloque (*block device*)
        - **DT_CHR**: Dispositivo de carácter (*character device*)
        - **DT_DIR**: Carpeta
        - **DT_FIFO**: Tubería nombrada
        - **DT_LNK**: Enlace simbólico (*symbolic link*)
        - **DT_REG**: Archivo
        - **DT_SOCK**: Socket de dominio UNIX (*UNIX domain socket*)
        - **DT_UNKNOWN**: Desconocido, indeterminado
    - **char d_name[256]:** El nombre del elemento, con fin de cadena.

2.3. Llamadas al sistema en el sistema operativo Windows

Nota importante: El sitio MSDN paso a ser https://docs.microsoft.com/en-us/ y los ejemplos de código están orientados a lenguaje C++.

2.3.1. OpenFile

```
#include <winbase.h>

HFILE OpenFile(
  LPCSTR     lpFileName,
  LPOFSTRUCT lpReOpenBuff,
  UINT       uStyle
);
```

La llamada el sistema **OpenFile()** permite crear, abrir, volver a abrir o eliminar un archivo.

- **lpFileName**: El nombre del archivo.
- **lpReOpenBuff**: Un apuntador a una estructura de tipo OFSTRUCT que contendrá información sobre el archivo cuando se abra por primera vez.
- **uStyle**: Acción a realizar con el archivo. Las más comunes son:
  - **OF_READ**: Solo lectura.
  - **OF_CREATE**: Crea un nuevo archivo, si existe se borra su contenido.
  - **OF_WRITE**: Solo escritura.
  - **OF_READWRITE**: Lectura y escritura.
  - **OF_DELETE**: Borra el archivo.
- En caso de éxito, la funcion devuelve una estructura **HFILE** (identificador) que se usara para manipular el archivo. En caso de error, devuelve **HFILE_ERROR.**

Nota: Microsoft no recomienda el uso de **OpenFile()** ya que tiene capacidades limitadas, en lugar de esta funcion recomienda el uso de la llamada al sistema **CreateFile.**

2.3.2. _lclose / CloseHandle

Nota: No existe una llamada al sistema con el nombre de **CloseFile,** para versiones de Windows con 16-bit se usa **_lclose**, para las demás versiones se debe de usar **CloseHandle**, ambas sirven para lo mismo, lo único que cambia es la versión de Windows en la que se deben de utilizar y el código de cómo usarla.

Caso **_lclose:**

```
#include <winbase.h>

HFILE _lclose(
  HFILE hFile
);
```

- **hFile**: Un identificador del archivo a ser cerrado. Este manejador es regresado por la funcion que creo o la última que abrió el archivo.
- El valor de regreso es un manejador del archivo para cerrar

Caso **CloseHandle**

```
#include <handleapi.h>

BOOL CloseHandle(
  HANDLE hObject
);
```

- **hObject**: Un identificador correcto del archivo.

- En caso de éxito, devuelve un valor distinto de cero. En caso de error, devuelve cero.

Ambas llamadas al sistema cierran un identificador de archivo.

2.3.3. ReadFile

```
#include <fileapi.h>

BOOL ReadFile(
  HANDLE       hFile,
  LPVOID       lpBuffer,
  DWORD        nNumberOfBytesToRead,
  LPDWORD      lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

La llamada al sistema **ReadFile()** lee datos del archivo especificado o del dispositivo E/S.

- **hFile**: Identificador del archivo, el cual debió haberse creado con permisos de lectura.
- **lpBuffer**: Apuntador al buffer que recibirá los datos leídos del archivo.
- **nNumberOfBytesToRead**: El número máximo de bytes a leer.
- **lpNumberOfBytesRead**: Puntero a una variable que guardara los bytes leídos.
- Devuelve TRUE en caso de éxito, FALSE en caso contrario.

2.3.4. WriteFile

```
#include <fileapi.h>

BOOL WriteFile(
  HANDLE       hFile,
  LPCVOID      lpBuffer,
  DWORD        nNumberOfBytesToWrite,
  LPDWORD      lpNumberOfBytesWritten,
  LPOVERLAPPED lpOverlapped
);
```

La llamada al sistema **WriteFile()** escribe datos al archivo especificado o al dispositivo E/S.

- **hFile**: Identificador de archivo, el cual debió haberse creado con permisos de escritura.
- **lpBuffer**: Apuntador al buffer que contiene la información que se escribira al archivo.
- **nNumberOfBytesToWrite**: El número de bytes a escribir en el archivo. Un valor 0 indica una escritura nula.
- Devuelve TRUE en caso de éxito, FALSE en caso contrario.

2.3.5. CreateFileA

Nota: No se encontró una llamada al sistema **CreateFile()**, sin embargo, se encontraron **CreateFile2, CreateFileA** y **CreateFileW**, sin embargo, las tres funciones son equivalentes.

```
#include <fileapi.h>

HANDLE CreateFileA(
  LPCSTR                lpFileName,
  DWORD                 dwDesiredAccess,
  DWORD                 dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD                 dwCreationDisposition,
  DWORD                 dwFlagsAndAttributes,
  HANDLE                hTemplateFile
);
```

La llamada al sistema **CreateFileA()** crea o abre un archivo o dispositivo E/S.

- **lpFileName**: Nombre del archivo. Se pueden usar diagonales (/) o diagonales invertidas (\).
- **dwDesiredAccess**: El tipo de acceso que se le dará al archivo, el cual puede ser lectura (**GENERIC_READ**), escritura (**GENERIC_WRITE**) o ambos (**GENERIC_READ** | **GENERIC_WRITE**).
- **dwShareMode**: Especifica cómo se va a compartir el archivo con el resto de las aplicaciones mientras se está usando aquí. Los valores pueden ser: bloqueo total (0), se puede eliminar (**FILE_SHARE_DELETE**), se puede leer (**FILE_SHARE_READ**), se puede escribir en el (**FILE_SHARE_WRITE**), o una combinación de las anteriores con una OR bit a bit.
- **dwCreationDisposition**: La acción que se realizara si el archivo no existe. Los valores posibles son: crear siempre, aunque exista (**CREATE_ALWAYS**), crear solo si no existe (**CREATE_NEW**), abrir siempre (**OPEN_ALWAYS**), abrir solo si existe (**OPEN_EXISTING**), abrir y truncar a 0 bytes solo si existe (**TRUNCATE_EXISTING**).
- **dwFlagsAndAttributes**: Los atributos del archivo, siendo **FILE_ATTRIBUTE_NORMAL** el más común. Se permiten atributos del tipo **FILE_ATTRIBUTE_*.**
- Devuelve un identificador al archivo en caso de éxito, **INVALID_HANDLE_VALUE** en caso de error.

2.3.6. SetFilePointer

```
#include <fileapi.h>

DWORD SetFilePointer(
  HANDLE hFile,
  LONG   lDistanceToMove,
  PLONG  lpDistanceToMoveHigh,
  DWORD  dwMoveMethod
);
```

La llamada al sistema **SetFilePointer()** mueve el apuntador de la posición de un archivo.

- **hFile**: Identificador del archivo, el cual debió haberse creado con permisos de lectura o escritura.
- **lDistanceToMove**: Número de bytes a moverse. Valor de 32 bits signado.
- **lpDistanceToMoveHigh**: Número de bytes a moverse. Valor de 32 o 64 bits. Si no se requiere, se debe de poner en NULL.
- **dwMoveMethod**: El punto de partida para el movimiento del puntero de la posición del archivo. Los valores posibles son: del comienzo del archivo (**FILE_BEGIN**), desde la posición actual (**FILE_CURRENT**), o desde el final del archivo (**FILE_END**).
- Devuelve un valor **DWORD** de la nueva posición del archivo en caso de éxito, **INVALID_SET_FILE_POINTER** en caso de error.

2.3.7. stat

```
01. #include <sys/stat.h>
02.
03. int stat(const char *restrict path, struct stat *restrict buf);
```

La función **stat()** devuelve información sobre el archivo o carpeta especificado.

- **path**: Cadena que contiene la ruta del archivo o carpeta existente.
- **buffer**: Apuntador de tipo estructura stat, que contendrá la información solicitada. Sus campos son casi los mismos a la llamada stat en Linux.
- Devuelve 0 si la información sí se pudo obtener, -1 si ocurrió un error.

2.3.8. opendir y readdir

```
01. #include <fileapi.h>
02.
03. HANDLE FindFirstFileA(
04.   LPCSTR             lpFileName,
05.   LPWIN32_FIND_DATAA lpFindFileData
06. );
```

```
01. #include <fileapi.h>
02.
03. BOOL FindNextFileA(
04.   HANDLE             hFindFile,
05.   LPWIN32_FIND_DATAA lpFindFileData
06. );
```

Lo más parecido en Windows es la llamada al sistema **()**, la cual busca en una carpeta archivos o subcarpetas de acuerdo con un término de búsqueda.

- **lpFileName**: La ruta a la carpeta, incluyendo el nombre del archivo, el cual puede contener comodines, como asterisco (*****) o signo de interrogación (**?**)
- **lpFindFileData**: Un puntero a una estructura de tipo **WIN32_FIN_DATA** que contendrá la información sobre el primer elemento encontrado.

- En caso de éxito, devuelve un identificador de búsqueda que se usara para las llamadas a las funciones **FindNextFileA()** o **FindClose()**. Si la carpeta no existe, devuelve **INVALID_HANDLE_VALUE**. Si no se encontró nada, devuelve **ERROR_FILE_NOT_FOUND**.

2.3.9. Equivalente de chmod en Windows

No existe un equivalente exacto de chmod en Windows, pues las definiciones de los permisos que Linux establece no son compatibles en Windows, ademas de que el modelo de seguridad es diferente. El comando más parecido es **icacls**.

2.4. Programas desarrollados en Linux

2.4.1. Creación aleatoria de archivos en una ruta dada

Código(PLinux1.c)

```
01. #include <stdio.h>
02. #include <stdlib.h>
03. #include <string.h>
04. #include <unistd.h>
05. #include <fcntl.h>
06. #include <time.h>
07. #include <sys/stat.h>
08. #include <sys/types.h>
09. 
10. char ruta[300];
11. void crearArchivos(){
12.     const char cadena[8][30]={{"Sistemas Operativos\n"},
13.                     {"Escuela Superior de Computo\n"},
14.                     {"2CM9\n"},
15.                     {"Cortes Galicia Jorge\n"},
16.                     {"Yosafat Martinez\n"},
17.                     {"Miguel Monteros\n"},
18.                     {"Guillermo Ramirez\n"},
19.                     {"Edmundo Sanchez\n"}};
20. 
21.     char nombreArchivo[18]="ArchivoLinux .txt\0";
22.     time_t t;
23.     srand((unsigned) time(&t));
24.     strcat(ruta,"/");
25.     for(char i = 'A'; i < 'H';i++){
26.         int random = rand() % 8;
27.         nombreArchivo[12]=i;
28.         char tempruta[300]="";
29.         strcat(tempruta,ruta);
30.         strcat(tempruta,nombreArchivo);
31.         int archivo = open(tempruta, O_WRONLY | O_CREAT | O_TRUNC, 0644);
32.         write(archivo,cadena[random], strlen(cadena[random]));
33.         close(archivo);
34.     }
35. }
36. 
```

```c
int main(){
    int verificaruta;
    char directorio[200];
    do{
        printf("Ingrese la ruta en donde se creara el directorio (Terminar con una /)\n");
        scanf("%s",ruta);
        printf("Ingrese el nombre del directorio a crear\n");
        scanf("%s",directorio);
        strcat(ruta,directorio);
        verificaruta = mkdir(ruta,200 | 400 | 100);
        if(verificaruta!=0){
            printf("Ruta no valida\n");
        }
    }while(verificaruta!=0);
    crearArchivos();
    return 0;
}
```

```
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ gcc PLinux1.c
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ ./a.out
Ingrese la ruta en donde se creara el directorio (Terminar con una /)
/home/edmundojsm/
Ingrese el nombre del directorio a crear
ArchivosCreadosLinux
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ cd
edmundojsm@edmundojsm-VB:~$ cd ArchivosCreadosLinux/
edmundojsm@edmundojsm-VB:~/ArchivosCreadosLinux$ ls
ArchivoLinuxA.txt  ArchivoLinuxB.txt  ArchivoLinuxC.txt  ArchivoLinuxD.txt  ArchivoLinuxE.txt  ArchivoLinuxF.txt  ArchivoLinuxG.txt
edmundojsm@edmundojsm-VB:~/ArchivosCreadosLinux$
```

2.4.2. Modificación de permisos dado un archivo

Código(PLinux2.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int opcion=0;
char rutaArchivo[200]="";

int main();
void menu();
void modificarArchivo();

void modificarArchivo(){

    switch (opcion){
        case 1:
            chmod(rutaArchivo,777);
            break;
        case 2:
            chmod(rutaArchivo,755);
            break;
        case 3:
            chmod(rutaArchivo,711);
            break;
        case 4:
            chmod(rutaArchivo,640);
            break;
        case 5:
            chmod(rutaArchivo,600);
            break;
        case 6:
            chmod(rutaArchivo,400);
            break;
```

```
33.             case 7:
34.                 chmod(rutaArchivo,200);
35.                 break;
36.             case 8:
37.                 chmod(rutaArchivo,100);
38.                 break;
39.             case 9:
40.                 chmod(rutaArchivo,040);
41.                 break;
42.             case 10:
43.                 chmod(rutaArchivo,020);
44.                 break;
45.             case 11:
46.                 chmod(rutaArchivo,010);
47.                 break;
48.             case 12:
49.                 chmod(rutaArchivo,007);
50.                 break;
51.             case 13:
52.                 chmod(rutaArchivo,006);
53.                 break;
54.             case 14:
55.                 chmod(rutaArchivo,005);
56.                 break;
57.             case 15:
58.                 chmod(rutaArchivo,004);
59.                 break;
60.             case 16:
61.                 chmod(rutaArchivo,003);
62.                 break;
63.             case 17:
64.                 chmod(rutaArchivo,002);
65.                 break;
66.             case 18:
67.                 chmod(rutaArchivo,001);
68.                 break;
69.             case 19:
70.                 chmod(rutaArchivo,000);
71.                 break;
72.             case 20:
73.                 main();
74.                 break;
75.             case 21:
76.                 exit(0);
77.                 break;
78.             default:
79.                 printf("Opcion no valida");
80.         }
81.         menu();
82.     }
```

```
83.
84.  void menu(){
85.
86.      printf("Opciones de permisos para archivos\n");
87.      printf("Ingrese el numero de la opcion que desee realizar\n");
88.      printf("1.- El archivo puede ser escrito, leido y ejecutado por quien sea\n");
89.      printf("2.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion y lectura\n");
90.      printf("3.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion\n");
91.      printf("4.- Lectura y escritura por propietario, el grupo solo puede leer y otros no pueden hacer nada\n");
92.      printf("5.- Leer y escribir por propietario\n");
93.      printf("6.- Lectura por propietario, los demas ni el puede modificarlo o ejecutarlo\n");
94.      printf("7.- Escribir por propietario\n");
95.      printf("8.- Ejecutar por propietario\n");
96.      printf("9.- Leer por grupo\n");
97.      printf("10.- Escribir por grupo\n");
98.      printf("11.- Ejecutar por grupo\n");
99.      printf("12.- Todos los permisos, lectura, escritura y ejecucion\n");
100.     printf("13.- Solo permisos de lectura y escritura\n");
101.     printf("14.- Solo permisos de lectura y ejecucion\n");
102.     printf("15.- Solo permiso de lectura\n");
103.     printf("16.- Solo permisos de escritura y ejecucion\n");
104.     printf("17.- Solo permiso de escritura\n");
105.     printf("18.- Solo permiso de ejecucion\n");
106.     printf("19.- No se tiene ningun permiso\n");
107.     printf("20.- Seleccionar otro archivo\n");
108.     printf("21.- Finalizar el programa\n");
109.     scanf("%i",&opcion);
110.     modificarArchivo();
111. }
112.
113. int main(){
114.     printf("Por favor ingrese la ruta del archivo al cual se le modificaran los permisos\n");
115.     scanf("%s", rutaArchivo);
116.     menu();
117.     return 0;
118. }
```

Compilación y ejecución donde la ruta es del directorio LINUX, modificando los archivos: ArchivoLinuxD.txt y ArchivoLinuxA.txt

```
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ gcc PLinux2.c
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ ./a.out
Por favor ingrese la ruta del archivo al cual se le modificaran los permisos
/home/edmundojsm/ArchivosCreadosLinux/ArchivoLinuxD.txt
Opciones de permisos para archivos
Ingrese el numero de la opcion que desee realizar
1.- El archivo puede ser escrito, leido y ejecutado por quien sea
2.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion y lectura
3.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion
4.- Lectura y escritura por propietario, el grupo solo puede leer y otros no pueden hacer nada
5.- Leer y escribir por propietario
6.- Lectura por propietario, los demas ni el puede modificarlo o ejecutarlo
7.- Escribir por propietario
8.- Ejecutar por propietario
9.- Leer por grupo
10.- Escribir por grupo
11.- Ejecutar por grupo
12.- Todos los permisos, lectura, escritura y ejecucion
13.- Solo permisos de lectura y escritura
14.- Solo permisos de lectura y ejecucion
15.- Solo permiso de lectura
16.- Solo permisos de escritura y ejecucion
17.- Solo permiso de escritura
18.- Solo permiso de ejecucion
19.- No se tiene ningun permiso
20.- Seleccionar otro archivo
21.- Finalizar el programa
18
```

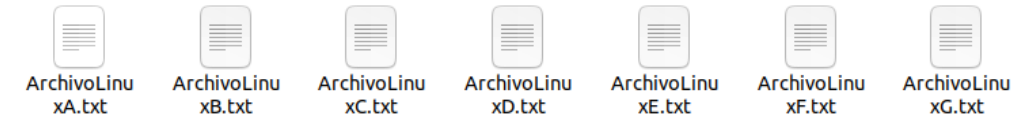
```
Opciones de permisos para archivos
Ingrese el numero de la opcion que desee realizar
1.- El archivo puede ser escrito, leido y ejecutado por quien sea
2.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion y lectura
3.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion
4.- Lectura y escritura por propietario, el grupo solo puede leer y otros no pueden hacer nada
5.- Leer y escribir por propietario
6.- Lectura por propietario, los demas ni el puede modificarlo o ejecutarlo
7.- Escribir por propietario
8.- Ejecutar por propietario
9.- Leer por grupo
10.- Escribir por grupo
11.- Ejecutar por grupo
12.- Todos los permisos, lectura, escritura y ejecucion
13.- Solo permisos de lectura y escritura
14.- Solo permisos de lectura y ejecucion
15.- Solo permiso de lectura
16.- Solo permisos de escritura y ejecucion
17.- Solo permiso de escritura
18.- Solo permiso de ejecucion
19.- No se tiene ningun permiso
20.- Seleccionar otro archivo
21.- Finalizar el programa
17
```

```
Opciones de permisos para archivos
Ingrese el numero de la opcion que desee realizar
1.- El archivo puede ser escrito, leido y ejecutado por quien sea
2.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion y lectura
3.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion
4.- Lectura y escritura por propietario, el grupo solo puede leer y otros no pueden hacer nada
5.- Leer y escribir por propietario
6.- Lectura por propietario, los demas ni el puede modificarlo o ejecutarlo
7.- Escribir por propietario
8.- Ejecutar por propietario
9.- Leer por grupo
10.- Escribir por grupo
11.- Ejecutar por grupo
12.- Todos los permisos, lectura, escritura y ejecucion
13.- Solo permisos de lectura y escritura
14.- Solo permisos de lectura y ejecucion
15.- Solo permiso de lectura
16.- Solo permisos de escritura y ejecucion
17.- Solo permiso de escritura
18.- Solo permiso de ejecucion
19.- No se tiene ningun permiso
20.- Seleccionar otro archivo
21.- Finalizar el programa
20
Por favor ingrese la ruta del archivo al cual se le modificaran los permisos
/home/edmundojsm/ArchivosCreadosLinux/ArchivoLinuxA.txt
```

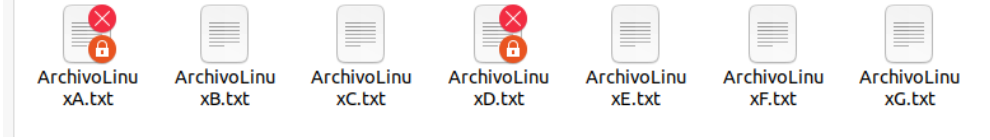
```
Opciones de permisos para archivos
Ingrese el numero de la opcion que desee realizar
1.- El archivo puede ser escrito, leido y ejecutado por quien sea
2.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion y lectura
3.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion
4.- Lectura y escritura por propietario, el grupo solo puede leer y otros no pueden hacer nada
5.- Leer y escribir por propietario
6.- Lectura por propietario, los demas ni el puede modificarlo o ejecutarlo
7.- Escribir por propietario
8.- Ejecutar por propietario
9.- Leer por grupo
10.- Escribir por grupo
11.- Ejecutar por grupo
12.- Todos los permisos, lectura, escritura y ejecucion
13.- Solo permisos de lectura y escritura
14.- Solo permisos de lectura y ejecucion
15.- Solo permiso de lectura
16.- Solo permisos de escritura y ejecucion
17.- Solo permiso de escritura
18.- Solo permiso de ejecucion
19.- No se tiene ningun permiso
20.- Seleccionar otro archivo
21.- Finalizar el programa
19
```

```
Opciones de permisos para archivos
Ingrese el numero de la opcion que desee realizar
1.- El archivo puede ser escrito, leido y ejecutado por quien sea
2.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion y lectura
3.- Lectura, escritura y ejecucion por propietario, el grupo y otros solo ejecucion
4.- Lectura y escritura por propietario, el grupo solo puede leer y otros no pueden hacer nada
5.- Leer y escribir por propietario
6.- Lectura por propietario, los demas ni el puede modificarlo o ejecutarlo
7.- Escribir por propietario
8.- Ejecutar por propietario
9.- Leer por grupo
10.- Escribir por grupo
11.- Ejecutar por grupo
12.- Todos los permisos, lectura, escritura y ejecucion
13.- Solo permisos de lectura y escritura
14.- Solo permisos de lectura y ejecucion
15.- Solo permiso de lectura
16.- Solo permisos de escritura y ejecucion
17.- Solo permiso de escritura
18.- Solo permiso de ejecucion
19.- No se tiene ningun permiso
20.- Seleccionar otro archivo
21.- Finalizar el programa
21
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$
```

Contenido del directorio ArchivosCreadosLinux (Antes de la ejecución):



Contenido del directorio ArchivosCreadosLinux (Después de la ejecución):



2.4.3. Información de archivos en un directorio (Tamaño, fecha y hora de acceso)

Código(PLinux3.c)

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

char rutaArchivo[200]="";
DIR *dirh;
struct dirent *dirp;
struct stat sb;

void InfoArchivo(){
    for(dirp = readdir (dirh);dirp !=NULL; dirp = readdir(dirh)){
        char tempruta[300]="";
        strcat(tempruta,rutaArchivo);
        strcat(tempruta,dirp->d_name);
        stat(tempruta,&sb);

        printf("Tamaño del archivo: %lld bytes \n", (long long) sb.st_size);
        printf("Ultimo acceso: %s", ctime(&sb.st_atime));
    }
}
```

```
26.
27.
28.  int main(){
29.      printf("Por favor ingrese la ruta del directorio\n");
30.      scanf("%s", rutaArchivo);
31.      if((dirh = opendir(rutaArchivo))==NULL){
32.          perror("opendir");
33.      }
34.      strcat(rutaArchivo,"/");
35.      InfoArchivo();
36.      return 0;
37.  }
```

Compilación y ejecución donde se muestra el tamaño, fecha y hora de acceso a los archivos en el directorio ArchivosCreadosLinux.

```
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ gcc PLinux3.c
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ ./a.out
Por favor ingrese la ruta del directorio
/home/edmundojsm/ArchivosCreadosLinux
Tamaño del archivo: 18 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 4096 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 4096 bytes
Ultimo acceso: Mon Oct 12 01:43:21 2020
Tamaño del archivo: 21 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 16 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 17 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 20 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 20 bytes
Ultimo acceso: Mon Oct 12 01:29:35 2020
Tamaño del archivo: 18 bytes
Ultimo acceso: Mon Oct 12 10:30:43 2020
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$
```

2.4.4. Mostrar contenido de archivo y copiar ese archivo u otros pertenecientes al mismo directorio

Código(PLinux4.c)

```
01.  #include <stdio.h>
02.  #include <stdlib.h>
03.  #include <unistd.h>
04.  #include <string.h>
05.  #include <sys/types.h>
06.  #include <sys/stat.h>
07.  #include <fcntl.h>
08.
09.  int fd,fd2,decision;
10.  char c, rutaOrigen[200], rutaOrigenTemp[200],rutaDestino[200],nomArchivo1[50],nomArchivo2[50];
11.
12.  int main();
13.  void ArchivoaCopiar();
14.  void CopiaArchivos();
15.
16.  void ArchivoaCopiar(){
17.      rutaOrigenTemp[0]='\0';
18.      strcat(rutaOrigenTemp,rutaOrigen);
19.      printf("\n\nDigite el nombre del archivo a copiar\n");
20.      scanf("%s", nomArchivo1);
21.      strcat(rutaOrigenTemp,nomArchivo1);
22.      fd = open(rutaOrigenTemp,O_RDONLY);
23.      if(fd == -1)
24.          printf("\nEl archivo que desea ver su contenido no existe ");
25.      else
26.          CopiaArchivos();
27.  }
```

```
28.
29. void CopiaArchivos(){
30.
31.     printf("\n\nIngrese la ruta en donde quiere guardar la copia del archivo (Terminar con una /)\n");
32.     scanf("%s",rutaDestino);
33.     printf("\n\nIngrese el nombre del nuevo archivo\n");
34.     scanf("%s",nomArchivo2);
35.     strcat(rutaDestino,nomArchivo2);
36.     fd2= open(rutaDestino,O_WRONLY | O_CREAT | O_TRUNC, 0644);
37.     while(read(fd,&c,sizeof(c)!=0)){
38.         write(fd2,&c,sizeof(c));
39.     }
40.     close(fd);
41.     close(fd2);
42.     printf("\n\nDesea copiar otro archivo de la misma ruta?\n0.-No 1.-Si\n");
43.     scanf("%i",&decision);
44.     if(decision){
45.         ArchivoaCopiar();
46.     }
47. }
48.
49. void VerArchivo(){
50.     fd = open(rutaOrigenTemp,O_RDONLY);
51.     if(fd == -1){
52.         printf("\nEl archivo que desea ver su contenido no existe ");
53.     }else{
54.         printf("\n Contenido de %s\n\n", nomArchivo1);
55.         while(read(fd,&c,sizeof(c)!=0)){
56.             printf("%c",c);
57.         }
58.         printf("\n\nDesea copiar este archivo?\n0.-No 1.-Si\n");
59.         scanf("%i",&decision);
60.         close(fd);
61.         if(decision){
62.             fd = open(rutaOrigenTemp,O_RDONLY);
63.             CopiaArchivos();
64.         }else{
65.             ArchivoaCopiar();
66.         }
67.     }
68. }
70. int main(){
71.     printf("Ingrese la ruta en donde se enecuentre el archivo a ver su contenido o copiar tanto como ese como otros archivos (Terminar con una /)\n");
72.     scanf("%s", rutaOrigen);
73.     printf("Digite el nombre del archivo que desea ver su contenido\n");
74.     scanf("%s", nomArchivo1);
75.     strcat(rutaOrigenTemp,rutaOrigen);
76.     strcat(rutaOrigenTemp,nomArchivo1);
77.     VerArchivo();
78.     return 0;
79. }
```

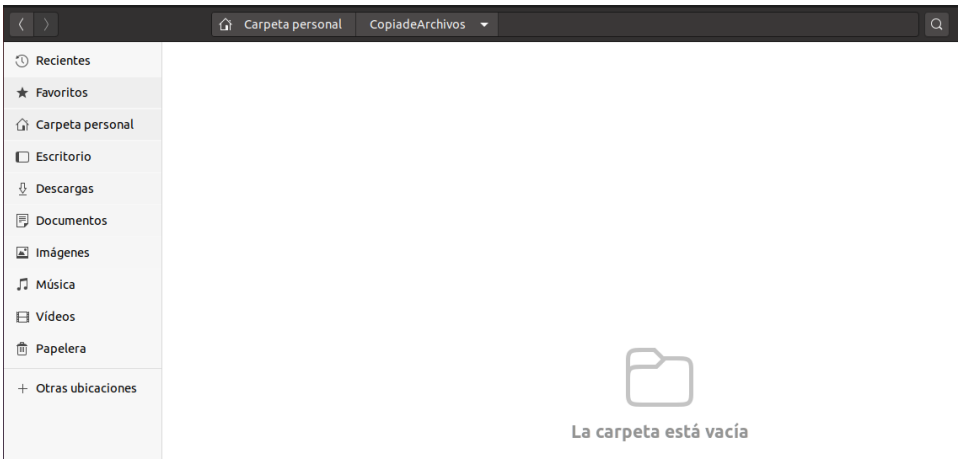
Compilación y ejecución donde se elige un archivo que se muestra su contenido en pantalla y su nuevo destino en un directorio con un nuevo nombre, ademas de copiar otro archivo.

```
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ gcc PLinux4.c
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ ./a.out
Ingrese la ruta en donde se enecuentre el archivo a ver su contenido o copiar tanto como ese como otros archivos (Terminar con una /)
/home/edmundojsm/ArchivosCreadosLinux/
Digite el nombre del archivo que desea ver su contenido
ArchivoLinuxA.txt

 Contenido de ArchivoLinuxA.txt

Escuela Superior de Computo


Desea copiar este archivo?
0.-No 1.-Si
1


Ingrese la ruta en donde quiere guardar la copia del archivo (Terminar con una /)
/home/edmundojsm/CopiadeArchivos/


Ingrese el nombre del nuevo archivo
CopiaA.txt


Desea copiar otro archivo de la misma ruta?
0.-No 1.-Si
1


Digite el nombre del archivo a copiar
ArchivoLinuxB.txt


Ingrese la ruta en donde quiere guardar la copia del archivo (Terminar con una /)
/home/edmundojsm/CopiadeArchivos/


Ingrese el nombre del nuevo archivo
CopiaB.txt


Desea copiar otro archivo de la misma ruta?
0.-No 1.-Si
0
```

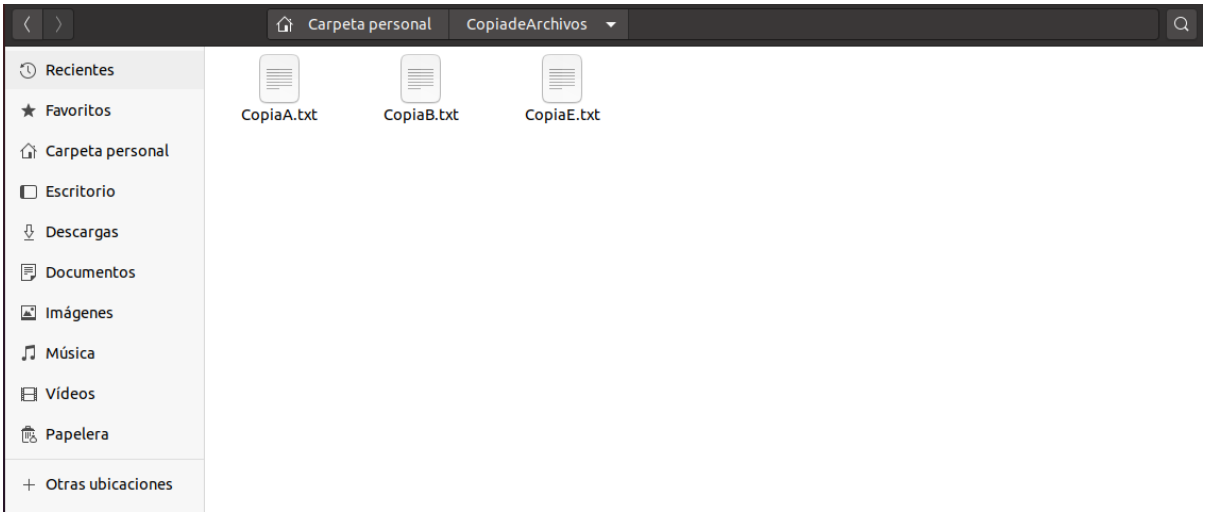
Ejecución donde se elige un archivo que se muestra su contenido, pero no se copia, ademas se copia otro archivo.

```
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$ ./a.out
Ingrese la ruta en donde se enecuentre el archivo a ver su contenido o copiar tanto como ese como otros archivos (Terminar con una /)
/home/edmundojsm/ArchivosCreadosLinux/
Digite el nombre del archivo que desea ver su contenido
ArchivoLinuxD.txt

 Contenido de ArchivoLinuxD.txt

Sistemas Operativos


Desea copiar este archivo?
0.-No 1.-Si
0


Digite el nombre del archivo a copiar
ArchivoLinuxE.txt


Ingrese la ruta en donde quiere guardar la copia del archivo (Terminar con una /)
/home/edmundojsm/CopiadeArchivos/


Ingrese el nombre del nuevo archivo
CopiaE.txt


Desea copiar otro archivo de la misma ruta?
0.-No 1.-Si
0
edmundojsm@edmundojsm-VB:~/Escritorio/Sistemas Operativos/Practica 2$
```

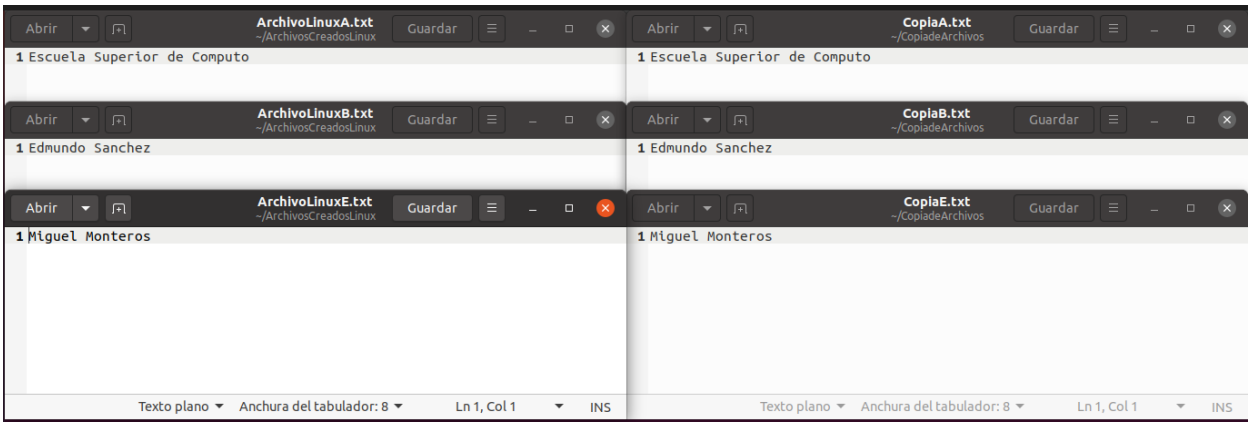
Contenido del directorio CopiadeArchivos (Antes de la ejecución).



Contenido del directorio CopiadeArchivos (Después de la ejecución).



Comparación de los contenidos de los archivos.



2.5. Programas en Windows

A comparación del programa realizado en Linux, este programa no crea un nuevo directorio debido a los requisitos.

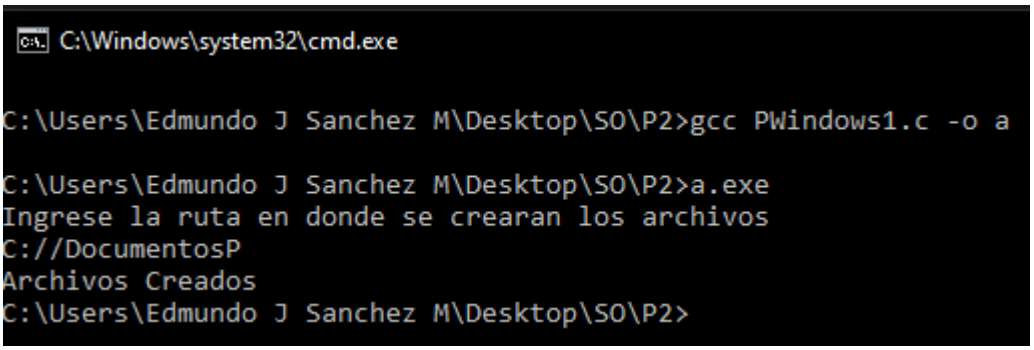
Nota: En los ejemplos de código para cada llamada al sistema investigado están orientados al lenguaje C++, para poder hacer esas mismas llamadas en C solo se llama al paquete <windows.h>

2.5.1. Creación aleatoria de archivos en una ruta dada

Código(PWindows1.c)

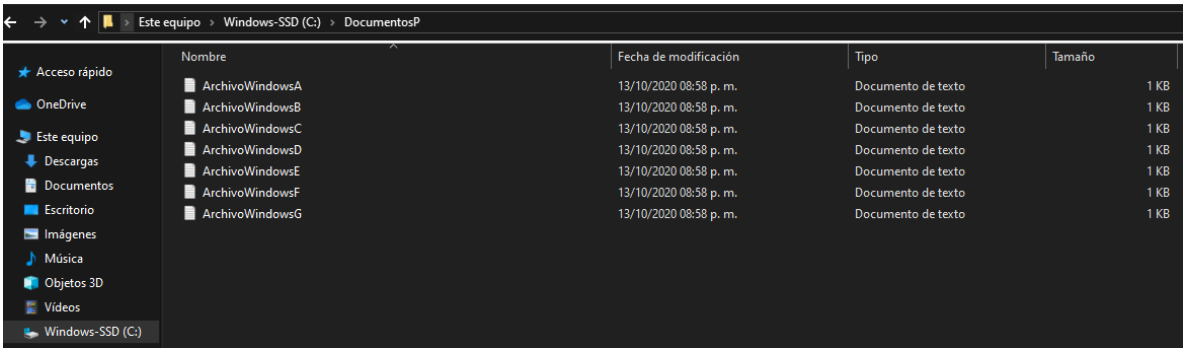
```
01. #include <windows.h>
02. #include <stdio.h>
03. #include <stdlib.h>
04. #include <string.h>
05. #include <time.h>
06. 
07. char ruta[300];
08. void crearArchivos(){
09.     const char cadena[8][30]={{"Sistemas Operativos\n"},
10.                     {"Escuela Superior de Computo\n"},
11.                     {"2CM9\n"},
12.                     {"Cortes Galicia Jorge\n"},
13.                     {"Yosafat Martinez\n"},
14.                     {"Miguel Monteros\n"},
15.                     {"Guillermo Ramirez\n"},
16.                     {"Edmundo Sanchez\n"}};
17. 
18.     char nombreArchivo[20]="ArchivoWindows .txt\0";
19.     time_t t;
20.     srand((unsigned) time(&t));
21.     strcat(ruta,"/");
22.     for(char i = 'A'; i < 'H';i++){
23.         int random = rand() % 8;
24.         HANDLE hfile;
25.         DWORD bytesWritten;
26.         nombreArchivo[14]=i;
27.         char tempruta[300]="";
28.         strcat(tempruta,ruta);
29.         strcat(tempruta,nombreArchivo);
30.         hfile = CreateFileA(tempruta, GENERIC_WRITE | GENERIC_READ,
31.         FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, CREATE_NEW,
32.         FILE_ATTRIBUTE_NORMAL,NULL);
33.         WriteFile(hfile,cadena[random],strlen(cadena[random]), &bytesWritten,NULL);
34.         CloseHandle(hfile);
35.     }
36. }
37. 
38. int main(){
39.     char directorio[200];
40.     printf("Ingrese la ruta en donde se crearan los archivos\n");
41.     scanf("%s",ruta);
42.     strcat(ruta,"\\");
43.     crearArchivos();
44.     printf("Archivos Creados");
45.     return 0;
46. }
```

Compilación y ejecución en donde se da la ruta (ya existente) en donde se crearán los archivos.



Contenido del directorio DocumentosP (Antes de la ejecución).

Contenido del directorio DocumentosP (Después de la ejecución).

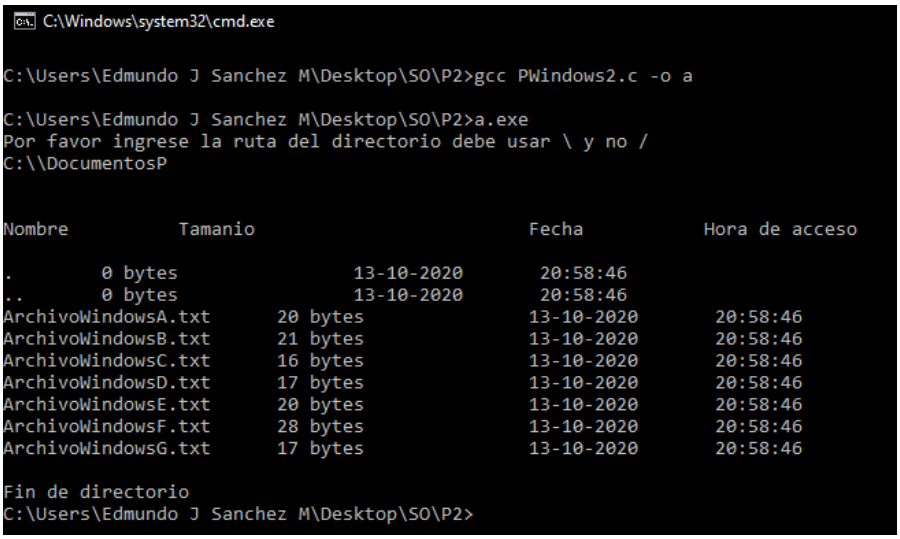


2.5.2. Información de archivos en un directorio (Tamaño, fecha y hora de acceso)

Código(PWindows2.c)

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

char rutaArchivo[200] = "";
HANDLE hFile;
WIN32_FIND_DATA data;
LARGE_INTEGER tamanioArchivo;

void Conversion(){
    FILETIME localFiletime;
    SYSTEMTIME sysTime;
    if(FileTimeToLocalFileTime(&data.ftLastAccessTime, &localFiletime)){
        if(FileTimeToSystemTime(&localFiletime, &sysTime)){
            printf("%.2d-%.2d-%4d \t %.2d:%.2d:%.2d\n",
            sysTime.wDay,sysTime.wMonth,sysTime.wYear,
            sysTime.wHour, sysTime.wMinute, sysTime.wSecond);
        }
    }
}

void printInfoArchivo(){
    printf("%s \t %d bytes\t\t",
        data.cFileName,
        data.nFileSizeLow);
    Conversion();
}

void InfoArchivo(){
    hFile = FindFirstFile(rutaArchivo, &data);
    printf("\n\nNombre\t\tTamanio\t\t\tFecha\t\tHora de acceso\n\n");
    printInfoArchivo(data);
    while(FindNextFile(hFile, &data)){
        printInfoArchivo(data);
    }
    if(GetLastError() == ERROR_NO_MORE_FILES){
        printf("\nFin de directorio");
    }
    FindClose(hFile);
}

int main(){
    printf("Por favor ingrese la ruta del directorio debe usar \\ y no /\n");
    scanf("%s", rutaArchivo);
    strcat(rutaArchivo,"\\*.*");
    InfoArchivo();
    return 0;
}
```

Compilación y ejecución donde se muestra el tamaño, fecha y hora de acceso a los archivos en el directorio DocumentosP.



```
C:\Windows\system32\cmd.exe

C:\Users\Edmundo J Sanchez M\Desktop\SO\P2>gcc PWindows2.c -o a

C:\Users\Edmundo J Sanchez M\Desktop\SO\P2>a.exe
Por favor ingrese la ruta del directorio debe usar \ y no /
C:\\DocumentosP


Nombre          Tamanio                         Fecha           Hora de acceso

.        0 bytes                13-10-2020       20:58:46
..       0 bytes                13-10-2020       20:58:46
ArchivoWindowsA.txt      20 bytes                13-10-2020       20:58:46
ArchivoWindowsB.txt      21 bytes                13-10-2020       20:58:46
ArchivoWindowsC.txt      16 bytes                13-10-2020       20:58:46
ArchivoWindowsD.txt      17 bytes                13-10-2020       20:58:46
ArchivoWindowsE.txt      20 bytes                13-10-2020       20:58:46
ArchivoWindowsF.txt      28 bytes                13-10-2020       20:58:46
ArchivoWindowsG.txt      17 bytes                13-10-2020       20:58:46

Fin de directorio
C:\Users\Edmundo J Sanchez M\Desktop\SO\P2>
```

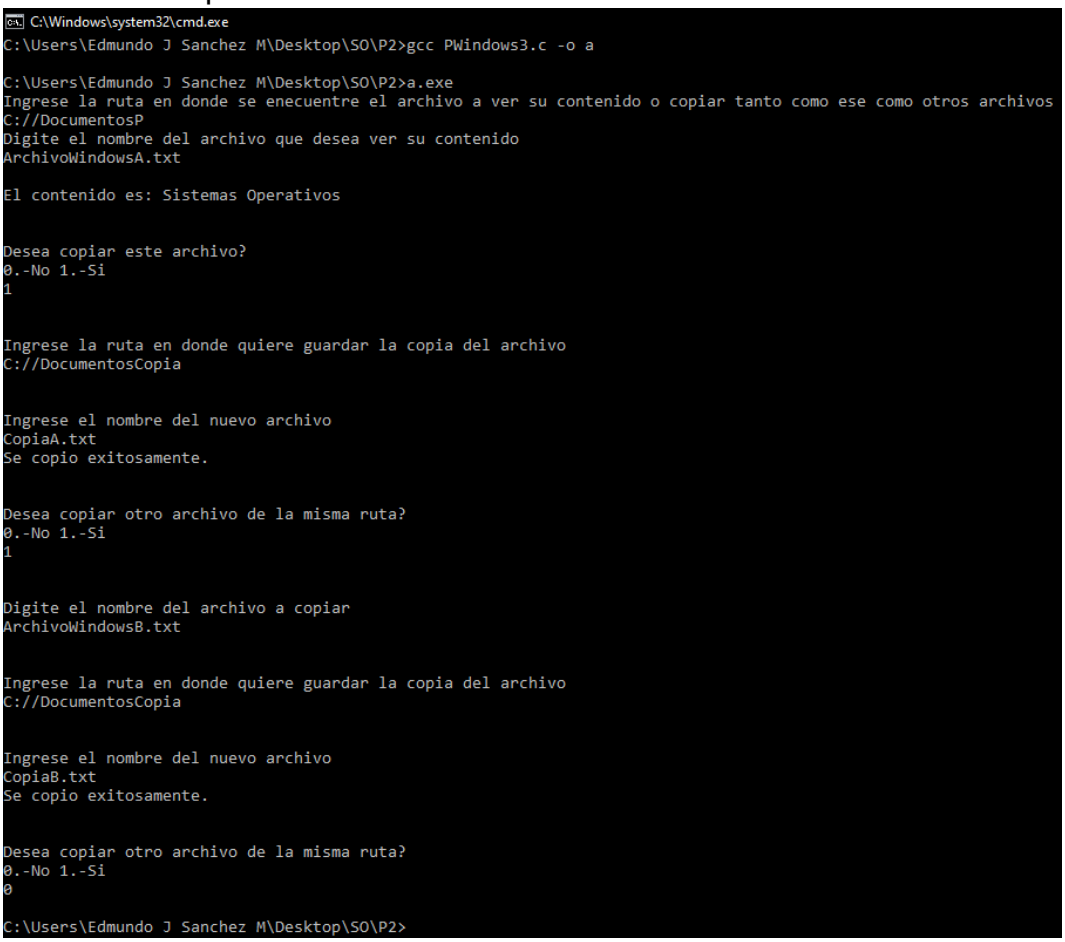
2.5.3. Mostrar contenido de archivo y copiar ese archivo u otros pertenecientes al mismo directorio

```
01. #include <windows.h>
02. #include <stdio.h>
03. #include <stdlib.h>
04. #include <string.h>
05. 
06. int decision;
07. HANDLE fd,fd2;
08. DWORD dwBytesLectura,dwBytesEscritura;
09. char rutaOrigen[200], rutaOrigenTemp[200],rutaDestino[200],nomArchivo1[50],nomArchivo2[50];
10. char BufferArchivo[500]="";
11. 
12. int main();
13. void ArchivoaCopiar();
14. void CopiaArchivos();
15. 
16. void ArchivoaCopiar(){
17.     rutaOrigenTemp[0]='\0';
18.     strcat(rutaOrigenTemp,rutaOrigen);
19.     printf("\n\nDigite el nombre del archivo a copiar\n");
20.     scanf("%s", nomArchivo1);
21.     strcat(rutaOrigenTemp,nomArchivo1);
22.     fd = CreateFileA(rutaOrigenTemp,
23.           GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
24.     CopiaArchivos();
25. }
```

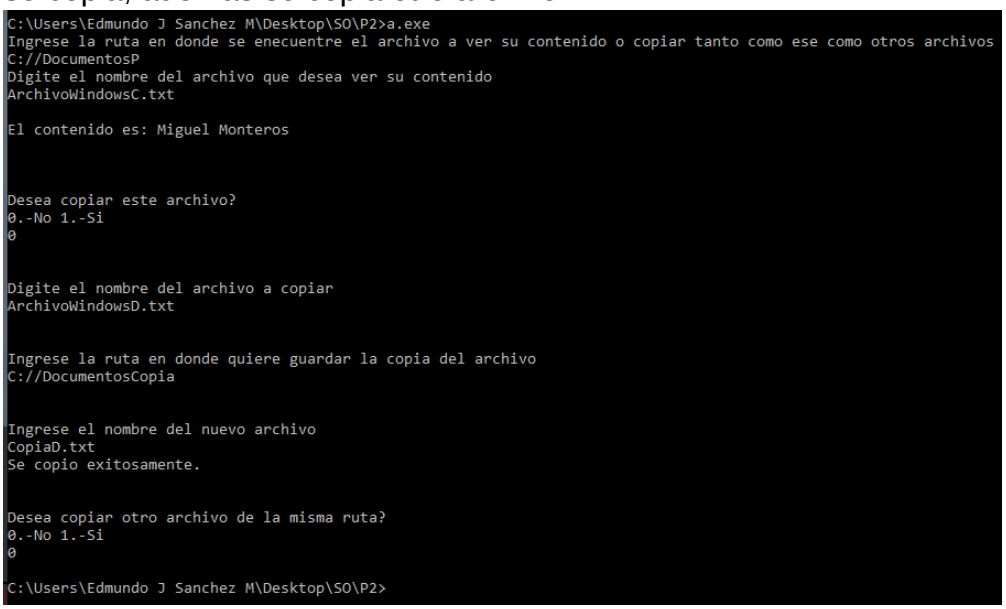
```
26.
27. void CopiaArchivos(){
28.     printf("\n\nIngrese la ruta en donde quiere guardar la copia del archivo\n");
29.     scanf("%s",rutaDestino);
30.     strcat(rutaDestino,"\\");
31.     printf("\n\nIngrese el nombre del nuevo archivo\n");
32.     scanf("%s",nomArchivo2);
33.     strcat(rutaDestino,nomArchivo2);
34.     fd2= CreateFileA(rutaDestino, GENERIC_WRITE | GENERIC_READ,
35.         FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, CREATE_NEW,
36.         FILE_ATTRIBUTE_NORMAL,NULL);
37.     ReadFile(fd, BufferArchivo, sizeof(BufferArchivo), &dwBytesLectura, NULL);
38.     if(WriteFile(fd2, BufferArchivo, sizeof(BufferArchivo), &dwBytesEscritura, NULL)){
39.         printf("Se copio exitosamente.\n");
40.     }
41.     BufferArchivo[0]='\0';
42.     CloseHandle(fd);
43.     CloseHandle(fd2);
44.     printf("\n\nDesea copiar otro archivo de la misma ruta?\n0.-No 1.-Si\n");
45.     scanf("%i",&decision);
46.     if(decision){
47.         ArchivoaCopiar();
48.     }
49. }
50.
```

```
51. void VerArchivo(){
52.     fd = CreateFileA(rutaOrigenTemp,
53.           GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
54.     if(ReadFile(fd, BufferArchivo, sizeof(BufferArchivo), &dwBytesLectura, NULL)){
55.         printf("\nEl contenido es: %s\n", BufferArchivo);
56.     }
57.     printf("\n\nDesea copiar este archivo?\n0.-No 1.-Si\n");
58.     scanf("%i",&decision);
59.     BufferArchivo[0]='\0';
60.     CloseHandle(fd);
61.     if(decision){
62.         fd = CreateFile(rutaOrigenTemp,
63.           GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
64.         CopiaArchivos();
65.     }else{
66.         ArchivoaCopiar();
67.     }
68. }
69.
70. int main(){
71.     printf("Ingrese la ruta en donde se enecuentre el archivo a ver su contenido o copiar tanto como ese como otros archivos\n");
72.     scanf("%s", rutaOrigen);
73.     strcat(rutaOrigen,"\\");
74.     printf("Digite el nombre del archivo que desea ver su contenido\n");
75.     scanf("%s", nomArchivo1);
76.     strcat(rutaOrigenTemp,rutaOrigen);
77.     strcat(rutaOrigenTemp,nomArchivo1);
78.     VerArchivo();
79.     return 0;
80. }
```

Compilación y ejecución donde se elige un archivo que se muestra su contenido en pantalla y su nuevo destino en un directorio con un nuevo nombre, ademas de copiar otro archivo.

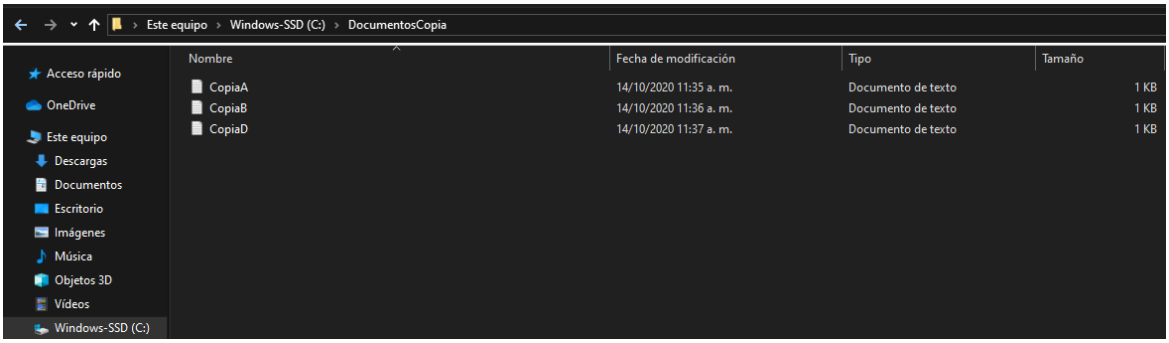


Ejecución donde se elige un archivo que se muestra su contenido, pero no se copia, ademas se copia otro archivo.

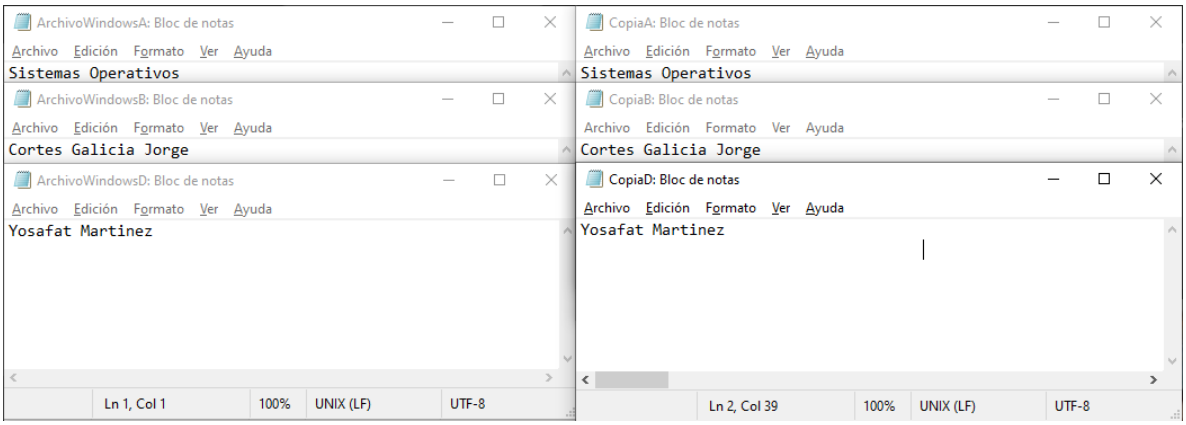


Contenido del directorio DocumentosCopia (Antes de la ejecución).

Contenido del directorio DocumentosCopia (Después de la ejecución).



Comparación de los contenidos de los archivos.



3. Análisis de la practica

En esta práctica tuvimos la oportunidad de comunicarnos con el sistema operativo mediante una interfaz a más bajo nivel: las llamadas al sistema, las cuales están agrupadas en una API que, dependiendo el sistema operativo, vamos a poder usar en algún lenguaje de programación, en esta práctica fue C, para poder incluir su funcionalidad en nuestras aplicaciones.

Las llamadas al sistema se usan por medio de funciones en C, las cuales es necesario hacer uso ciertas bibliotecas para su invocación. Las llamadas al sistema siempre retornan un valor de información acerca del éxito que tuvo al producir su

ejecución, este valor puede ser usado o no. Como se vio en la práctica hay llamadas al sistema equivalentes entre sistemas operáticos, pero con una estructura diferente de invocación, también se pueden simular con llamadas al sistema funciones similares que algunos comandos.

4. Observaciones

4.1. Compatibilidad entre archivos

- La memoria USB se montó automáticamente en Linux al igual que Windows.
- El explorador de archivos de Linux nos permite montar y desmontar las USB con un solo clic, ademas, de que nos aparece un icono exclusivo para la USB.
- Word no guarda sus archivos en texto plano, usa un formato propio, no compatible con gedit, pero si con LibreOffice.
- Al abrir archivos que contienen caracteres no ASCII en gedit, no se ven correctamente.
- Al tratar de modificar dichos archivos en gedit, se corrompen.
- Si se va a usar gedit, que sea solo para archivos de texto plano.

4.2. Llamadas al sistema

- En Linux, usualmente un valor de retorno 0 indica éxito en la función, mientras que en Windows indica error. Hay que tener cuidado con eso.
- Las rutas de archivos o directorios en Linux y Windows tienen diferente sintaxis.
- Todas las llamadas al sistema tienen un valor de retorno que indica el éxito que tuvo al ejecutarse la llamada.
- Linux cuenta con una llamada al sistema chmod con la funcionalidad para modificar los accesos de permisos a los usuarios, mientras que Windows no cuenta con una llamada al sistema con la misma funcion.
- Hay llamadas al sistema para crear archivos y directorios, entre otras más las cuales suelen trabajar siempre con las rutas.

5. Conclusiones

Tanto Linux como Windows cuentan con distintas llamadas al sistema, las cuales son similares, sin embargo, tienen una distinta forma de invocación y diferentes parámetros.

La mayoría de las llamadas al sistema en ambos sistemas operativos trabajan con la ruta del archivo o directorio como un parámetro de la llamada, mencionar que la forma de escribir la ruta es diferente.

Pudimos observar que las llamadas al sistema simulan la misma funcion que un comando, nada más que la llamada al sistema tiene un proceso más largo para su invocación que un comando ejecutado en la terminal. También cabe destacar que se pueden ejecutar varias llamadas al sistema al mismo tiempo, es una característica que los comandos no tienen.